The calx, xintexpr and allied packages

Jean-François Burnol
jfbu (at) free (dot) fr
Package version: 2.0a2 (2026/11/26).
From source file xint-calx.dtx. Time-stamp: <26-11-2026 at 20:46:25 CET>.

Part I. The calx and xintexpr packages

1 **The calx package** 3

1.1 Usage 3
1.2 What is calx? 4
1.3 Timing comparisons versus \fpeval for sin(), ln(), exp(), ... 5
1.4 User interface 5
1.5 Arithmetic signals, exceptional results and error handling 6

2 **Introduction to the rest of this manual (which is the legacy 1.4 documentation)** 8

2.1 This documentation 9
2.2 Compatible engines and formats 9
2.3 Usage 9
2.4 xintsession 10
2.5 zeckendorf 11
2.6 polexpr 11
2.7 bnumexpr 11
2.8 Printing big numbers on the page 11
2.9 Repository 12
2.10 License and installation instructions 12

3 **Syntax reference and user guide** 13

3.1 Introduction: the three-parsers way 13
3.2 Output customization 16
3.3 Built-in operators and their precedences 20
Table of precedence levels of operators 21
3.4 Built-in functions 25
Table of functions in expressions 26
3.5 Generators of arithmetic progressions 43
3.6 Python slicing and indexing of one-dimensional sequences 44
3.7 NumPy like nested slicing and indexing for arbitrary oples and nutples 45
3.8 Tacit multiplication 46
3.9 User defined variables 47
3.10 User defined functions 51
3.11 Examples of user defined functions 58
3.12 Links to some (old) examples within this document 62
3.13 Oples and nutples: the 1.4 terminology 62
3.14 Expansion (for geeks only) 67
3.15 Known bugs/features (last updated at 1.4n) 68

4 **The macros of xintexpr (ancient documentation, mostly)** 71

5 **The calxtrig.tex library** 89

6 **The calxlog.tex library** 93

7 **Macros of the xinttools package** 95

8 **Additional (old) examples with xinttools or xintexpr or both** 116

Part II. The macro layer for expandable computations: xintcore, xint, xintfrac, and some extras

9 **Introduction: the xint bundle** 135

10 **Macros of the xintkernel package** 150

11 **Macros of the xintcore package** 154

12 **Macros of the xint package** 159

13 **Macros of the xintfrac package** 171

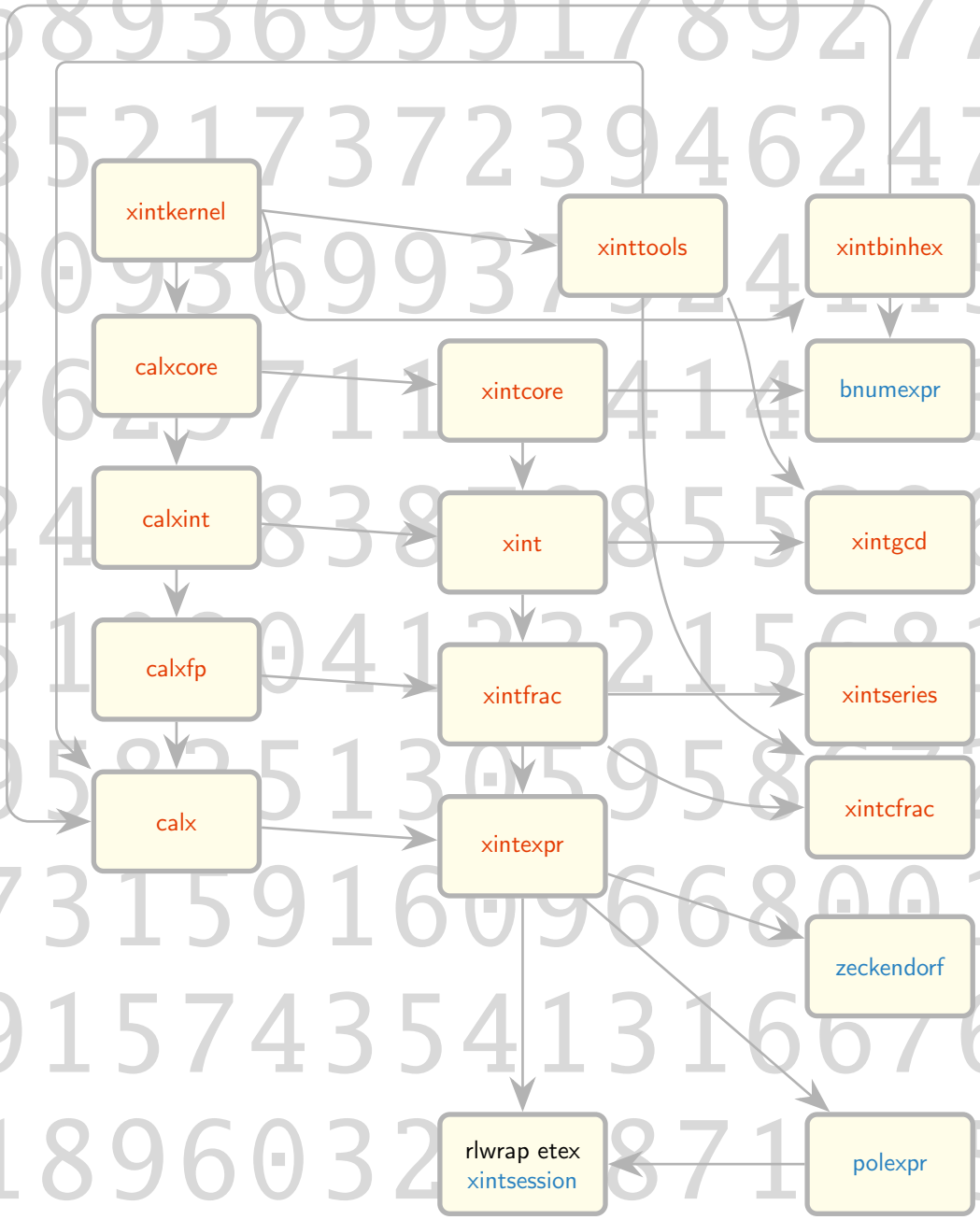
14 **Macros of the xintbinhex package** 202

15 **Macros of the xintgcd package** 208

16 **Macros of the xintseries package** 211

17 **Macros of the xintcfrac package** 227

Part III. The xintexpr and allied packages source code



Dependency graph for the xint bundle components. Modules pointed to by arrows **automatically** import the module from which the arrow originates. calxcore, calxint and calxfp provide no user interface at all. DO NOT USE THEM DIRECTLY.

bnumexpr is a LaTeX package by the author which uses (by default) xintcore as its mathematical engine. To use it under Plain eTeX issue first `\input miniltx.tex` then `\input bnumexpr.sty`.

zeckendorf can be used with Plain ε-TeX or with LaTeX and also directly on the command line. It enriches the \xinteval interface with tools related to Fibonacci numbers and the Bergman φ-representation.

polexpr handles definitions and algebraic operatione on one-variable polynomials, as well as root localization to arbitrary precision. It works both with Plain TeX and with LaTeX.

xintsession is invoked on the command line as `etex xintsession` (or, much better if available: `rlwrap etex xintsession`).

The 1250th Fibonacci number has 261 digits.

Part I.
The calx and xintexpr packages

1 The calx package 3
2 Introduction to the rest of this manual (which is the legacy 1.4 documentation) 8
3 Syntax reference and user guide 13
4 The macros of xintexpr (ancient documentation, mostly) 71
5 The calxtrig.tex library 89
6 The calxlog.tex library 93
7 Macros of the xinttools package 95
8 Additional (old) examples with xinttools or xintexpr or both 116

1. The calx package

.1 Usage 3
.2 What is calx? 4
.3 Timing comparisons versus \fpeval for sin(), ln(), exp(), ... 5
.4 User interface 5
.5 Arithmetic signals, exceptional results and error handling 6

Jürgen Gilg's interest into what he called "XINT" was instrumental in keeping the author motivated over the years. We exchanged on many topics extending beyond TEX and often reacted similarly to private and public events. I knew he was a very kind and devoted person, who took care of the needs of others prior to his own, although he never mentioned it. Jürgen suffered a sudden, unexpected, and deadly stroke in May 2022. I will miss his friendship profoundly.

The advent of xintexpr 2.0, also known as calx, is almost entirely backward compatible. There are a few breaking changes and probably not all of them are properly documented for the 2.0a1 release.

1.1. Usage

Loading:

```
\usepackage{calx}% with LaTeX
\input calx.sty\relax with other e-TeX formats.
```

Example:

```
$2^{200}=\calxinteval{2^200}$%  \calxinteval and \xintiieval are the same thing.

\noindent % \calxeval and \xintfloateval are the same thing.
$(\ln(3), \pi)\approx (\calxeval{ln(3), Pi})$
```

```
\xintDigits*:= 32\relax % or \xintSetDigits*{32}
\noindent % with 32 digits (hence, here, 31 decimal places).
$(\ln(3), \pi) \approx (\calxeval{ln(3), Pi})$

\xintDigits*:= 16\relax % back to default setting
```

2^{200} = 1606938044258990275541962092341162602522202993782792835301376
$(\ln(3), \pi) \approx (1.098612288668110, 3.141592653589793)$
$(\ln(3), \pi) \approx (1.09861228866810969139524523692225, 3.14159265358979323846264338327795)$

The scientific libraries support precisions up to 100 digits. The arithmetic operations, square root and ordinary integer powers can be used at higher precisions (use the non-starred precision setters). For tips on how to let TEX print such long numeric output without overflowing into the right margin, please search for \printnumber in the legacy manual (which is actually the bulk of this PDF).

Note: loading xintexpr automatically loads calx. So legacy documents automatically benefit from the speed-up and higher accuracy.

1.2. What is calx?

It is the new arithmetic unit (floating point and big integers) of xintexpr at release 2.0. It is a complete rewrite of thousands of lines of code, and addition of thousands more[1], which was done in October 2026 (and took the whole month to get completed) by (mainly) Claude Opus 5.5, and ChatGPT 6. This was under the training, guidance and general supervision by an anonymous human assistant to Professor Burnol.

NOTA BENE: *the anonymous assistant checked only very little of the produced code and was each time horrified by what he saw.* He tried to fix a few obvious sub-optimalities and to get the AI assistants to learn from him (which he thought they had, but no), with little success. Much remains to be done at time of the 2.0a1 release for cleaning up the code. But it works.

The whole process was much more difficult and lengthy than originally envisioned. It reached the limits of what the anonymous assistant was willing to devote to an open source project with no financial gratification (especially when on a few occasions the human did buy some additional credits on top of the basic monthly subscription). However it was seen as an escape from the Apocalypse of July-August-September 2026 which completely changed the field of interest of the anonymous assistant: mathematics. Its sociology is going to change for the worse very quickly and irremediably now.

Contrarily to xintexpr, calx loads *none* of xintcore, xint, xintfrac. **It does not handle fractions**. For this you must load xintexpr package for its \xinteval evaluator.

For mathematical functions the target accuracy was set at a maximal error of 0.005ulp. This means roughly that correct rounding is achieved in 99.75% of the cases[2]. It seems that trigonometry actually achieved better percentage of correct rounding: Among 20300 trigonometrical evaluations 19 were not correctly rounded, and 18 among them were for the especially handled $P \leq 8$ case. For example for 4872 cases involvoing log(), exp(), log1p(), expm1(), only 21 were not correctly rounded and the error was then 1ulp. The $P = 8$ mode has an about ten times degraded target accuracy, speed being the priority.

> The core arithmetic operations have not yet been specifically optimized for $P = 8$ and still incur some overhead from supporting higher precisions.

[1] Roughly the whole codebase has increased by 50%. If one loads only calx, the "size" is increased only about 25% compared with what applied with 1.4 series xintexpr, due to calx not providing the support for fractions. [2] Assuming a uniform distribution of the signed error in the allowed range, the probability of incorrect rounding decreases linearly from 50% near a tie to zero at distance 0.005ulp from it. Assuming also a uniform distribution of the position relative to ties, we obtain the indicated estimate. If the errors are more concentrated around zero, as is likely, the actual percentage of correct rounding will be higher.

For example mutliplication at P = 8 uses the two-block mantissa kernel designe for P = 16. This is clearly sub-optimal.

1.3. Timing comparisons versus \fpeval for sin(), ln(), exp(), ...

The Table 1 compares \fpeval with \calxeval used at precision settings P=8, 16 and 32 (P is set by \xintDigits*, before the measurement). Library reconstruction is not included in the timings. The average evaluation time of \fpeval{sin(1)} is used as a time unit for the entries in the \fpeval column. The \calxeval columns show the ratios of calx timings with those of \fpeval. Thus, a value below one indicates that calx is faster, and a value above one that it is slower. All measurements include expansion of the expression and assignment of its result to a macro via \edef. In particular they include the overhead of the respective parsers gathering the input and printing the output.

	\fpeval sin(1) units	\calxeval (P=16) timing ratio to \fpeval	\calxeval (P=8) timing ratio to \fpeval	\calxeval (P=32) timing ratio to \fpeval
sin(1.2345678)	1.04	0.84	0.37	1.80
cos(1.2345678)	1.00	1.18	0.37	2.22
tan(1.2345678)	1.40	1.52	0.31	3.63
asin(0.56784321)	3.21	1.11	0.11	2.79
acos(0.56784321)	3.17	1.30	0.11	3.23
atan(1.2345678)	2.44	1.96	0.15	4.68
ln(2.3456789)	1.29	0.40	0.26	0.70
exp(1.2345678)	1.06	0.55	0.41	0.83
12.3456^78.9012	2.30	0.85	0.57	1.69
1.23456^789.012	2.28	0.79	0.56	1.71

Table 1: Timings

The given values are indicative, variations of a few percent are typically observed whenever launching again the tests. The P = 8 precision is often enough for plots, perhaps a future release will include some interface to use the calx floating point engine in TikZ/pgf or PSTricks, but for now there is only \xintthecoords and \xintthespaceseparated.

The gap for atan() between P=8 and P=16 may be a sign of some serious sub-optimatlity in the current implementation. This may be addressed, if ever, at a next release.

1.4. User interface

The user interface is reduced to:

- \calxeval or its legacy alias \xintfloateval: a parser of floating point expressions at a precision P set by \xintSetDigits. For details of the available syntax elements see the legacy xintexpr manual. The scientific libraries support precisions from 1 to 100 (the legacy maximal precision was 62); use \xintSetDigits* to set their precision too.
- \calxinteval or its legacy alias \xintiieval: handles arbitrarily large integers. Refer to the legacy xintexpr manual.
- The interface to set the floating point precision via \xintSetDigits{⟨P⟩} or equivalently \xintDigits=⟨P⟩\relax. The non-starred forms set the arithmetic precision only. Their starred variants also set the precision of the scientific libraries and re-assign the trigonometrical constants such as Pi. The scientific libraries construct only the tables needed at

their working precision, which is the next multiple of four. The legacy libraries xintlog.sty and xinttrig.sty are gone, replaced by the calx ones.

- \xintifboolfloatexpr.
- The legacy xintexpr interface to define functions and variables: \xintNewFunction, \xintdeffloatfunc, \xintdeffloatvar, \xintdefiifunc, \xintdefiivar.
- Things such as \xintglobaldefstrue, \xintverbosetrue, \xintthecoords, and a few others.
- \calxsetup which is described in the next section devoted to the handling of condition signals.

The packages xintkernel, xintbinhex, and xinttools are loaded automatically.

Loading xintexpr does not modify anything to calx interface and functionalities. It adds to it the \xinteval parser (together with \xintdeffunc, \xintdefvar) and loads all of xintcore, xint, and xintfrac.

The 2.0 macro interface of xintfrac is unchanged but arithmetic internals are remapped to the calx substratum. Some undocumented at user level macros of xintfrac having to do with floating point numbers are gone.

The 2.0a1 release has a few breaking changes (behavior and availability of qfloat() and qfrac() come to mind) which may not have been documented in time. The code comments are very likely either outdated or barely legible.

1.5. Arithmetic signals, exceptional results and error handling

.5.1 Default: division by zero without a TEX error 6
.5.2 Exceptional values and their printed forms 7
.5.3 Testing an expression for a non-finite result 7
.5.4 Testing signals after a calculation 8
.5.5 Signals, flags, traps and handlers 8

Summary: *an arithmetic condition is not necessarily a TEX error. In fact, by default calx 2.0a1 in case of division by zero returns a non-finite result (**NaN**, **+Infinity**, or **-Infinity**) while the compilation continues:* more generally, all arithmetic traps are disabled initially.

A *trap* raises an error message and causes TEX to consider that a build error has arisen; applications which prefer active traps may enable them individually one-by-one or globally.

The codebase was already done by Professor Burnol in 2017 but was left then in a state where all error conditions are trapped, i.e. do trigger TEX errors.

> The whole signal-handler system, described next, is available only for arithmetic with \calxeval (aka \xintfloateval, i.e. floating point calculations) and \calxinteval (aka \xintiieval, i.e. big integers calculations). The legacy evaluator \xinteval (which handles fractions but also authorizes the mathematical functions) does not support the special values NaN, +Infinity and -Infinity as arithmetic operands. Note that the fact that \xinteval authorizes mathematical functions such as sin() and exp() should not be abused: any multiplication of two such creates mantissas twice as big, obviously.

1.5.1. Default: division by zero without a TEX error

The following code does not interrupt the compilation:

```
\calxresetflags
\edef\result{\calxeval{1/0}}
% \result expands to +Infinity
```

```
  \calxifflagraised{DivisionByZero}
    {Division by zero occurred.}
    {No division by zero occurred.} The result is \result.
```
```
Division by zero occurred. The result is +Infinity.
```

Similarly `\calxeval{-1/0}` produces `-Infinity`.

The calculation raises the DivisionByZero flag and returns +Infinity. Since calx has no signed zero, division of a nonzero numerator by zero uses the numerator's sign to determine the infinity. For `\calxeval{0/0}`, the result is instead NaN; this is the DivisionUndefined condition, which raises the InvalidOperation flag rather than DivisionByZero.

A flag is raised when the expression is actually expanded, as in the `\edef` above. Merely storing an unevaluated expression via `\def` or, in LaTeX, `\newcommand`, does not raise a flag.

To make division by zero a TeX error, enable its trap:

```
  \calxsetup{DivisionByZero.trap=true}
```

A trapped condition emits an expandable TeX error and returns the operation's trap value. To restore the default behavior, use:

```
  \calxsetup{DivisionByZero.trap=false}
```

1.5.2. Exceptional values and their printed forms

The expression parsers `\calxeval` and `\calxinteval` recognize NaN and Infinity as reserved constants. A negative infinity can be written -Infinity. Thus, the following expressions all yield a non-finite result, without necessarily raising a new signal:

```
  \calxeval{NaN, Infinity, -Infinity}
```
```
NaN, +Infinity, -Infinity
```

In particular, using a special constant is not the same as signalling an arithmetic condition.

An application which prefers NaN for division by zero such as 1/0 can replace the default handler. The handler receives the numerator's sign as its argument; it may ignore it and emit the internal marker for NaN, which is I?.

```
  \def\calxDivisionByZeroAsNaN#1{I?}
  \calxsetup{DivisionByZero.handler=\calxDivisionByZeroAsNaN}
  \edef\result{\calxeval{1/0}}
  % \result expands to NaN (which is the printed form of I?)
```
```
Overflow independently retains its default results, +Infinity and -Infinity.
```

Note that the internal markers for the non-finite values differ from the user interface ones NaN, +Infinity and -Infinity: they are respectively I?, I+, I-. These internal notations are only *to be used in defining handlers as above.*

1.5.3. Testing an expression for a non-finite result

The expandable predicate `\xintifNonFinitefloatexpr` takes a floating-point expression followed by two branches:

```
  \xintifNonFinitefloatexpr{<expression>}{<non-finite>}{<finite>}
```

It evaluates the expression once and selects the first branch when its *internal result* is I?, I+ or I-. The second branch is selected for a finite result. For example:

```
  \xintifNonFinitefloatexpr{1/0}{YES}{NO}% YES
  \xintifNonFinitefloatexpr{NaN}{YES}{NO}% YES
  \xintifNonFinitefloatexpr{Infinity}{YES}{NO}% YES
  \xintifNonFinitefloatexpr{sin(1)}{YES}{NO}% NO
```
```
YESYESYESNO
```

This predicate examines the computed value, not the signal flags or its printed spelling. Evaluating an expression inside this test can itself raise signal flags.

1.5.4. Testing signals after a calculation

If the calculation has already been performed, use the expandable predicate shown below to check the flags for invalid operations, division by zero and overflow:

```
\xintifNonFiniteSignalled{<yes>}{<no>}
```

For example:

```
\calxresetflags
\edef\result{\calxeval{1/0}}
\xintifNonFiniteSignalled
  {A non-finite condition was signalled}
  {No non-finite condition was signalled}
```

A non-finite condition was signalled

This test does not re-evaluate an expression and does not inspect \result. A handler may return a finite value while leaving a signal flag raised; conversely, a literal NaN or Infinity need not raise a flag. Flags persist until cleared within their scope (e.g. LaTeX environments), so use \calxresetflags before a calculation when only its newly raised conditions are of interest.

1.5.5. Signals, flags, traps and handlers

There are eight signals:

```
Clamped          DivisionByZero  Inexact       InvalidOperation
Overflow         Rounded         Subnormal     Underflow
```

Each has a flag and an independently configurable trap. No arithmetic trap is enabled by default.

Five additional conditions use the InvalidOperation flag and trap:

```
ConversionSyntax    DivisionImpossible  DivisionUndefined
InvalidContext      ControversialOperation
```

They each have individually configurable handlers.

Flags are inspected and cleared with the following commands:

```
\calxifflagraised{InvalidOperation}{<yes>}{<no>}
\calxresetflag{InvalidOperation}
\calxresetflags
```

The flag test is expandable; the reset commands are assignments and are not expandable. Flags are local to TeX groups. Examine flags raised inside a group before leaving it. Reset commands obey the prevailing \xintglobaldefs setting, as do assignments by \calxsetup.

The default handlers for InvalidOperation and its subordinate conditions return internal I?; the DivisionByZero and Overflow handlers return I+ or I-, according to the sign. The default handlers for Clamped, Inexact, Rounded, Subnormal and Underflow leave the computed result unchanged.

Traps and handlers are configured with \calxsetup:

```
\calxsetup{InvalidOperation.trap=true}
\calxsetup{Overflow.trap=false}
\calxsetup{ControversialOperation.handler=\myHandler}
```

The value of a .handler key is a single control sequence denoting an expandable one-argument handler. An unknown setup key or invalid trap value is a configuration error and is reported by TeX, regardless of the arithmetic-trap settings.

2. Introduction to the rest of this manual (which is the legacy 1.4 documentation)

.1 This documentation 9
.2 Compatible engines and formats 9
.3 Usage 9
.4 xintsession 10
.5 zeckendorf 11
.6 polexpr 11
.7 bnumexpr 11
.8 Printing big numbers on the page 11
.9 Repository 12
.10 License and installation instructions 12

2.1. This documentation

Whenever you see ***code*** above some control sequence **\foo**, it is an hyperlink to its coding in the second part of this document. You should find there, near your landing point, the control sequence \foo in orange: this is a hyperlink to go back to its documentation in the user manual part. Within the source code, other colored control sequences should link back to the location of first definition.

And throughout the user manual part, blue color such as \xinteval signals an hyperlink to the documentation of the macro. You can try it now.

2.2. Compatible engines and formats

The components of the xint bundle can be used indifferently with Plain TEX (and other formats, as mentioned next) or with LATEX. The sole difference being that with the latter the loading must be done by \usepackage whereas with any non-LATEX format it has to be via \input (using .sty filename extension, not .tex).

The engine can be PDFTEX, XeTEX, or LuaTEX.

With release 1.4n you can also use the packages with ConTEXt (only latest one, with LuaMetaTEX engine), and OpTEX.

You can't use the xint bundle with Knuth's original tex binary, because its functionalities require \numexpr and other e-TEX extensions as well as the more recent \expanded engine primitive (and \pdfstrcomp or \strcmp).

xintexpr will be probably the main entry point, and it actually automatically loads most other components. The aim of xintexpr is to provide expandable parsers of numerical expressions, either floating point numbers or fractions.

2.3. Usage

Here is an example:

```
  \xintfloateval{cos(3Pi/17)*sin(1)^0.123 + log(3.42e5)}
13.57492307809003
```

You can get the result with more digits:

```
  \xintSetDigits*{32}
  \xintfloateval{cos(3Pi/17)*sin(1)^0.123 + log(3.42e5)}
13.574923078090031147995554039818
```

And with still more digits (the [-2] rounds away the two least significant ones):

```
  \xintSetDigits*{62}
  \xintfloateval[-2]{cos(3Pi/17)*sin(1)^0.123 + log(3.42e5)}
13.574923078090031147995554039817996515169465699140806395197 5
```

Here is now an example with fractions:

```
  \xinteval{reduce(add(1/i^3, i=1..25))}
256097615265221153640811111018 9/213185813136131994295737600000 0
```

And two examples with large integers:

```
  $2^{1000}=\printnumber{\xinteval{2^1000}}$.\newline
  $100!^3=\printnumber{\xintiieval{100!^3}}$.
```

2^{1000} = 107150860718626732094842504906000181056140481170553360744375038837035105112493612249 3
↪ 198378815695858127594672917553146825187145285692314043598457757469857480393456777482423098541 5
↪ 210746050623711418779541821530464749835819412673987675591655439460770629145711964776865421675 5
↪ 66042983165262438683720566806937 6.

$100!^3$ = 81285103704665697929058034741394527800954175275203119077085794747670888482337305968 5
↪ 67201883750504781387762207126471259231411592064116091993540375458364906984360126190005190897 5
↪ 02481351072344988957966094631503344938807996687425862917630302052505909887462286075836527716 5
↪ 23341365916290092476956859429554672135618951275111007717373291473301054034842043089511584699 5

```
↪ 57099274146970547638354741532999364798054400000000000000000000000000000000000000000000000000000000000000000,
↪ 000000000000000000000.
```

The \printnumber utility to wrap very long output is not part of the package. It is documented in subsection 2.8 next.

The table of built-in functions and the one of built-in operators will give a quick overview of the available syntax.

2.4. xintsession

The simplest way[3] to test the syntax is to work interactively on the command line (this feature is available since April 2021, the version of xintsession used here is 1.3a). Beware though that ill-formed inputs will trigger TEX famously antiquated error handling, from which it is hard to recover, altough hitting S may sometimes miraculously bring you back to the xintsession prompt.

```
rlwrap etex xintsession
[...welcome banner...]
  Magic words: `&pause' (or `;'), `&help', `&bye',
               `&exact', `&fp', `&int', `&pol'.

  \jobname is xintsession
  Transcript will go to log and to xintsession-210609_12h00.tex
Starting in exact mode (floating point evaluations use 16 digits)
>>> 2^100;
@_1     1267650600228229401496703205376
>>> cos(1);
@_2     0.5403023058681397
>>> &fp=32
(/usr/local/texlive/2021/texmf-dist/tex/generic/xint/xintlog.sty)
(/usr/local/texlive/2021/texmf-dist/tex/generic/xint/xinttrig.sty)
fp mode (log and trig reloaded at Digits=32)
>>> cos(1);
@_3     0.54030230586813971740093660744298
>>> 3^1000;
@_4     1.3220708194808066368904552597521e477
>>> &exact
exact mode (floating point evaluations use 32 digits)
>>> 3^1000;

@_5     1322070819480806636890455259752144365965422032752148167664920368226828
97346704899540778313850608061963909777696872582355950954582100618911865342725
79536740276202251983208038780147742289648412743904001175886180411289478156230
43806156617305408667449050617812548034440554705439703889581746536825491613622
30268563778582290228416398307887896918556404084898937609373242171846359938695
67650189405881090604260896714388641028143503856487471658320106143661321731027
902855220001
>>> &bye
Did I say something wrong?
Session transcript written on xintsession-210609_12h00.tex
 )
No pages of output.
Transcript written on xintsession.log.
```

[3] I am assuming here in the displayed example a Unixen system, i.e. Mac OS or Linux, adapt to your environment.

2.5. zeckendorf

This package enriches the \xinteval interface with tools related to Fibonacci numbers and Bergman φ-representation. In particular, it implements the Knuth Fibonacci multiplication. It can be used either as a LaTeX package, or with Plain ε-TeX, or also on the command line via rlwrap etex zeckendorf.

2.6. polexpr

The package polexpr enriches the \xinteval syntax (but not the one of \xintfloateval) with a polynomial type with associated constructor pol([c0,c1,...]), and polynomial specific functions such as polgcd(pol1, pol2, ...).

Full usage of polynomials (and algebraic notations c_0 + c_1 x + c_2 x^2 + ... for input and also output) goes through a dedicated \poldef parser which is based upon \xintdefvar/\xintdeffunc and is a necessary step to then access via a dedicated macro interface operations such as identifying all rational roots and isolating all real roots to arbitrary precision.

The simplest manner to experiment with polexpr is via the &pol mode of xintsession.

2.7. bnumexpr

This LaTeX package loads xintcore and xintbinhex and provides \bnumeval which is a scaled-down \xintiieval (omitting support for nested structures, functions, variables, boolean branching, etc...). It can be used with Plain eTeX, thanks to miniltx. For this, use \input miniltx.tex followed by \input bnumexpr.sty (remark: miniltx is not needed for xintexpr).

\bnumeval is thus a boosted \inteval which addition of support for arbitrarily large integers, powers with ** and ^, rounded division with /, floored division with // and associated modulo /:, factorials via ! postfix operator, comma separated multi-item expressions.

It also supports as \xintiieval does the ', ", 0b, 0o and 0x input prefixes and the optional arguments [b], [o], or [h].

Further, it provides an interface (which does not exist with xintexpr) to let all operations be done by macros of one's own choosing, as replacement for some or all of the operations by default implemented via the help of xintcore and xintbinhex. It even makes it possible to add to the syntax extra infix or postfix operators and to modify the precedence levels of those already defined.

2.8. Printing big numbers on the page

When producing very long numbers there is the question of printing them on the page, without going beyond the page limits. In this document, I have most of the time made use of a ``\printnumber'' macro, which is not provided by the package. A primitive form would be:

```
\def\allowsplits #1{\ifx #1\relax \else #1\hskip 0pt plus 1pt\relax
                    \allowbreak
                    \expandafter\allowsplits\fi}%
\def\printnumber #1{\expandafter\allowsplits \romannumeral-`0#1\relax }%
```

This macro triggers ``*f*-expansion'' of its argument (and indeed \xinteval and friends expand completely under such trigger), then it goes through the computation result character by character inserting some stretchable glue in-between them. The \allowbreak's (which is known both to Plain TeX and to LaTeX and perhaps other formats, else use \penalty0\relax) are here only for the case of usage in inline math mode (they are ineffective in display math mode). The actual \printnumber used for building this documentation is a more sophisticated variant (never used in math mode), which handles comma separated lists without gobbling spaces and prints continuation symbols at line endings.

2.9. Repository

It is at https://github.com/jfbu/xint. At this stage, it does not serve for real-time development but only proposes all actual successive CTAN releases since 2013. Use it to report issues. Don't forget to include @jfbu in the ticket else I will not be pinged.

A front page at https://jfbu.github.io/xint provides, in addition to the present xint.pdf and to README.md a file CHANGES.html, which contains the complete list of changes relevant to user level since the initial release of the package:

https://jfbu.github.io/xint/CHANGES.html

Its version xintchanges.md in Markdown format is included in the CTAN upload,

texdoc xintchanges.md

Warning: I don't have the time to maintain perfectly such large documentation. It combines old documentation which never really got updated and may be locally obsolete with more recent stuff mostly written on occasion of the 1.4 release of January 2020 and the 1.4e one of May 2021, and the intervening changes might also have made some of it not completely accurate, despite my best efforts.

2.10. License and installation instructions

The xint bundle components are made available under the LaTeX Project Public License 1.3c.

They are available at CTAN. They are included in all major TEX distributions, thus there is probably no need for a custom install: just use the package manager to update from time to time to latest version.

3. Syntax reference and user guide

.1 Introduction: the three-parsers way 13
.2 Output customization 16
.3 Built-in operators and their precedences .. 20
Table of precedence levels of operators 21
.4 Built-in functions 25
Table of functions in expressions 26
.5 Generators of arithmetic progressions 43
.6 Python slicing and indexing of one-dimensional sequences 44
.7 NumPy like nested slicing and indexing for arbitrary oples and nutples 45
.8 Tacit multiplication 46
.9 User defined variables 47
.10 User defined functions 51
.11 Examples of user defined functions 58
.12 Links to some (old) examples within this document 62
.13 Oples and nutples: the 1.4 terminology ... 62
.14 Expansion (for geeks only) 67
.15 Known bugs/features (last updated at 1.4n) 68

3.1. Introduction: the three-parsers way

> **WARNING:** this documentation goes sometimes into too much details, and does need some improvements. But there is no time for that at 1.4n. Although people do not believe me when I say that, there is ample intellectual reward in actually reading the documentation, and it would be nice if at least, at last someone on Earth did, once (as Jürgen Gilg some years back).

xintexpr provides three numerical expression parsers corresponding to these three respective tasks:

\xintfloateval: evaluations with floating point numbers; the default precision is with 16 digits, it can be set via \xintSetDigits*,
\xinteval: exact evaluations with fractions, decimal fixed point numbers, numbers in scientific notation, with no size limitation,
\xintiieval: evaluations allowing only integers with no size limitation.

The interface is:

- \xintfloateval[⟨*Q*⟩]{⟨*expression*⟩} does floating point computations with a given precision, which defaults to 16. The precision P can be set using \xintDigits*:=P\relax or \xintSetDigits*{P} syntaxes.

  Its optional argument [Q], if present, means to do a *final* float rounding to a mantissa of Q digits (this thus makes sense only if Q<P).

  A negative Q is allowed and means to round to P+Q digits only.

  Prior to 1.4k the optional argument [⟨*Q*⟩] had to be located *within* the braces at the start of the expression. The legacy syntax is and will keep being allowed.

  The infix operator / will compute the correct rounding of the exact fraction. The operator // is floored division and /: is its associated modulo (see also divmod()).

```
\begingroup
\xintDigits:=64\relax
\xintfloateval{sqrt(3)}
\endgroup
```

```
1.7320508075688772935274463415058723669428052538103806280558069 79
```

  The four basic operations and the square root achieve *correct rounding*.[4]

  On output, \xintfloateval uses \xintPFloat for each numeric leaf. This can be modified (cf. \xintfloatexprPrintOne).

[4] when the inputs are already floating point numbers with at most P-digits mantissas.

- `\xinteval{⟨expression⟩}` handles integers, decimal numbers, numbers in scientific notation and fractions. The algebraic computations are done *exactly*, and in particular `/` simply constructs fractions. Use `//` for floored division.

```
  \xinteval{add(x/(x+1), x = 1000..1014)}\par
```
```
464848270976783588640014901759941534 3/3102065976122748153921551507331573 60
```

The output in this specific example came out irreducible. In general one needs a `reduce()` wrapper for an irreducible output:

```
  \[\xinteval{subs((y,reduce(y)), y = mul(x/1000, x = 10..[10]..90))}\]
```
$$362880000000000/1000000000000000000000000000,\ 567/1562500000000000$$

Arbitrarily long numbers are allowed in the input. The space character (contrarily to the situation inside `\numexpr`) and also the underscore character (as allowed in Python too) can serve to separate groups of digits for better readability. But the package currently provides no macros to let the output be formatted with such separators.

```
  \xinteval{123_456_789_012^5}
```
```
2867971861693552444294278300558210585854333156276376883 2
```

NEW WITH 1.4n

Hexadecimal, octal and binary (with fractional part allowed) can be input using suitable prefixes: respectively `"` or `0x`, `'` or `0o`, and `0b`:

```
  \xinteval{0b111111111111.111111111111}
```
```
4095.999755859375
```

- `\xintieval[⟨D⟩]{⟨expression⟩}` is the same parser as `\xinteval`, i.e. accepts the same inputs and does all computations exactly in the same manner, but it then rounds its final result to the nearest integer, or, in case there is an optional argument `[D]`, to:
  - if `D>0`: the nearest fixed point number with `D` digits after the decimal mark,
  - if `D=0`: the nearest integer (as for `\xintieval` with no optional argument),
  - if `D<0`: the rounded quotient by `10^(-D)`.

  The optional argument `[⟨D⟩]` can also be located *within* the braces at the start of the expression (this was actually the legacy syntax until `1.4k`).

- `\xintiieval{⟨expression⟩}` executes computations *on (big) integers only*. It is (only slightly) faster than `\xinteval` for the same expression.

  Attention: the forward slash `/` does the *rounded* integer division to match behaviour of `\numexpr`. The `//` operator does floored division as in `\xinteval`. The `/:` is the associated modulo operator (we could easily let the catcode 12 `%` character be an alias, but using such an unusual percent character would be a bit cumbersome in a TEX workflow, if only for matters of syntax highlighting in TEX-aware text editors).

```
  % add the i^5 only if i is a multiple of 7
  \xintiieval{add((i/:7)?{omit}{i^5}, i=1000..1020)}
```
```
3122939154402144
```

NEW WITH 1.4n

An optional argument `[h]`, `[o]`, or `[b]` says to convert the output to hexadecimal, octal or binary:

```
  \xintiieval[h]{3^100}
```
```
5A4653CA673768565B41F775D6947D55CF3813D1
```

Please note:

- If you find that \xintfloateval is too much of a mouthful, you can create an alias named, for example, \fpeval. Oops, no, that is LaTeX3 very efficient floating point engine (for 16 decimal digits of precision). It is faster than \xintfloateval due to the excellent coding and algorithms it employs, but also for reasons having to do in part with the fact that xint initially was created to handle big integers only, and support for arbitrary precision floating point was added on top of that, via some expedients which have never been refactored, yet.
- Although \xinteval manipulates arbitrarily long integers or fractions it also accepts scientific notation on input, as well as all the mathematical functions (evaluated using the prevailing digits precision), and (depending on customization) can thus produce also scientific notation on output. But be careful that for example a multiplication will then automatically roughly double the number of digits in general (some operations avoid extending mantissas, so short mantissas may remain short).
- So far, individual operations and the printing routine of \xinteval do not automatically reduce fractions to their lowest terms.

There is a core syntax:

- \xintexpr⟨*expression*⟩\relax,
- \xintiexpr⟨*expression*⟩\relax,
- \xintiiexpr⟨*expression*⟩\relax,
- \xintfloatexpr⟨*expression*⟩\relax,
- \xintboolexpr⟨*expression*⟩\relax.

\xintboolexpr⟨*expression*⟩\relax does all computations like \xintexpr then converts all (non-empty) leaves[5] to true or false (cf. \xintboolexprPrintOne). There is no \xintbooleval.[6]

CHANGED AT 1.4m!

Formerly the \xintexpr...\relax legacy syntax had to be prefixed by \xintthe if in typesetting context, else an error was raised (deliberately). The \xintthe prefix was made optional at 1.4.

In an \edef these constructs expand to some braced nested data, all computations having been completely done, which is prefixed with some \protected "typesetter" macros.

In an \edef, \xinteval (in contrast to \xintexpr), or \xintfloateval (in contrast to \xintfloatexpr) expand the "typesetting macros" and the final complete expansion consists of explicit digits and other characters such as those of scientific notation or square brackets.[7]

In LaTeX it is possible to use the core syntax \xintexpr⟨*expression*⟩\relax also in so-called moving arguments, because when written out to a file the final expansion outcome uses only standard catcodes and thus will get retokenized and expand as expected if it has been written to an external file which is then reloaded.

One needs \xinteval et al. only if one really wants the final digits (and other characters), for example in a context where TeX expects a number or a dimension.

As alternative to \xinteval{⟨*expression*⟩}, an equivalent is \xintthe\xintexpr⟨*expression*⟩\relax. Similarly \xintthe can prefix all other core parsers. And one can also use \xintttheexpr as shortcut for \xintthe\xintexpr.

Doing exact computations with fractions leads very quickly to very big results (and furthermore one needs to use explicitly the reduce() function to convert the fractions into smallest terms). Thus most probably what you want is \xintfloateval and \xintfloatexpr.

[5] Currently, empty leaves are output using \xintexprEmptyItem, i.e. default to []. This may change. [6] This was True and False prior to 1.4m. [7] \xinteval and \xintexpr both expand completely in exactly two steps. And \xintexpr expands fully under *f*-expansion (of the \romannumeral 0 or -`0 type). As per \xinteval attention that it may expand to nothing, then naturally *f*-expansion propagates to tokens following up in the input stream.

3.2. Output customization



Warning. As a consequence of the internal changes in 2.0, much of the material in this section is currently outdated. In particular, customizing \xintfloatexprPrintOne and \xintiiexprPrintOne at user level is no longer as straightforward as it was in earlier releases. This section will be revised in a future release.

3.2.1. \xintfloatexprPrintOne et al. for numerical values

code

The package provides only minimal facilities for formatting the output from \xinteval or \xintfloateval or. . . . And this output may well consist of comma separated values, even nested ones with, by default, square brackets. First we explain how to influence the handling of individual "leaves".

Here are the default definitions to this effect:

```
\def\xintexprEmptyItem{[]}                                % (all parsers)
\def\xintexprPrintOne  #1{\xintFracToSci{#1}}             % \xinteval
\def\xintiexprPrintOne #1{\xintDecToString{#1}}           % \xintieval
\def\xintiiexprPrintOne#1{#1}                             % \xintiieval
\def\xintfloatexprPrintOne [#1]#2{\xintPFloat[#1]{#2}}    % \xintfloateval
\def\xintboolexprPrintOne#1{\xintiiifNotZero{#1}{true}{false}}
```

They can be re-defined to one's wishes. If configured to do anything non expandable they must be \protected. LaTeX users will want to use \RenewDocumentCommand for this.

TeX-hackers note:

- Actually, the defaults are more done in the style

  ```
  \let\xintexprPrintOne\xintFracToSci
  ```

  thus sparing grabbing the argument #1. And one can do

  ```
  \def\xintexprPrintOne{\xintFracToSci}
  ```

  too.
- \xintexprPrintOne defaults in truth to some private variant of \xintFracToSci with exactly the same output but able to understand only certain limited types of inputs as used internally.
- NEW WITH 1.4n — \xintiiexprPrintOne is used with \xintiieval. But it gets replaced with \xintiiexprPrintOneHex, \xintiiexprPrintOneOct, or \xintiiexprPrintOneOct if the optional argument [h], [o], or [b] is used. These macros default to respectively \xintDecToHex, \xintDecToOct and \xintDecToBin.
- \xintfloatexprPrintOne defaults in fact to a private variant of \xintPFloat which assumes the optional argument [P] is present as it will be the case always in this context. This optional argument [P] is the optional argument [Q] of \xintfloateval (or Digits+Q if Q<0).
- The typesetter for \xintiiexpr simply prints ``as is'', but this may change in future, if some internal format is used requiring a conversion step.

Here is a possibly not up-to-date list of macros of interest, whose documentations you might consider reading (the first two require math mode):

- \xintTeXFromSci,
- \xintTeXFrac,
- \xintDecToString,
- \xintPRaw,
- \xintFracToSci,
- \xintFracToDecimal,
- \xintPFloat,
- and \xintFloatToDecimal.

Naming scheme, as one can see, has been pretty much incoherent, apologies.

Among packages providing macros formatting numeric values, there are numprint and its macro \np (or \numprint without the option np), and siunitx and its \num, and possibly more packages not known to the author.[8] These macros are suitable in combination with \xintFloat as in the example below to customize the \xintfloateval output. Numerical output from \xinteval is more challenging as individual values may naturally contain the / character for fractions which the above mentioned packages will not know how to handle, as far as I know.

Here an example, with LaTeX and \num from siunitx:

```
\RenewDocumentCommand\xintfloatexprPrintOne{o m}{\num{\xintFloat[#1]{#2}}}
```

We could have used here simply \def with delimited parameters [#1]#2 because:

- the optional argument will always be present at time of use,
- \num is a \protected macro.

Note that when using only \def for the definition, the argument of \num is getting to be expanded first, but \num would have done that anyhow.

With numprint, one can similarly do:

```
\RenewDocumentCommand\xintfloatexprPrintOne{o m}{\numprint{\xintPFloat[#1]{#2}}}
```

This used \xintPFloat rather than \xintFloat as with \num. This is because (in my limited testing) \numprint with not silently remove a zero scientific exponent but it will typeset it, for example as $1.5\cdot 10^{0}$. So we use our own \xintPFloat poor man "prettifier".

Maybe you want to use a macro which is unable to have \xintPFloat[#1]{#2} as argument because it needs to see only a number in scientific notation and nothing else. If that macro is \protected, do the definition with \def. It it is not \protected and not purely expandable either, one can do this:

```
\protected\def\myfoo{\foo}%<<<--- with options perhaps
\def\xintfloatexprPrintOne[#1]#2{\myfoo{\xintPFloat[#1]{#2}}}
```

Then when \myfoo finally expands, its argument has been expanded already.

The current behaviour of \xintfloateval corresponds to this set-up:

```
\def\xintfloatexprPrintOne [#1]#2{\xintPFloat[#1]{#2}}
```

and to this default configuration of \xintPFloat:

```
\def\xintPFloatE{e}
\def\xintPFloatZero{0}
\def\xintPFloatIntSuffix{}
\def\xintPFloatLengthOneSuffix{}
\def\xintPFloatNoSciEmax{5}
\def\xintPFloatNoSciEmin{-4}
\def\xintPFloatMinTrimmed{4}
```

With the custom replacement

```
\def\xintfloatexprPrintOne{\xintFloatToDecimal}
```

the \xintfloateval output will use decimal fixed point notation, i.e. no scientific exponents, and as many zeros as are needed (but no more, as trailing zeros will be removed from the significant digits). Here is an example comparing outputs from the default configuration and custom ones:

```
\xintfloateval{exp(-32.456)/2000} (default, i.e. PFloat)\newline
\def\xintfloatexprPrintOne{\xintFloatToDecimal}%
\xintfloateval{exp(-32.456)/2000} (FloatToDecimal)\newline
\def\xintfloatexprPrintOne[#1]#2{\xintTeXFromSci{\xintFloat[#1]{#2}}}%
$\xintfloateval{exp(-32.456)/2000}$ (TeXFromSci on Float)\par % math mode required
```

$4.013361680161317e-18$ (default, i.e. PFloat)
0.0000000000000000004013361680161317 (FloatToDecimal)
$4.013361680161317\cdot 10^{-18}$ (TeXFromSci on Float)

Some examples showing now the effect of sensible customizations on \xinteval:

```
\xinteval{exp(-32.456)/2000} (default, i.e. FracToSci)\newline
```

[8] There does not seems to be yet a LaTeX user level interface to the l3str-format package, part of l3experimental, which provides an implementation of the Python format function.

```
  \def\xintexprPrintOne{\xintFracToDecimal}%
  \xinteval{exp(-32.456)/2000} (FracToDecimal)\newline
  \def\xintexprPrintOne#1{\xintTeXFromSci{\xintFracToSci{#1}}}%
  $\xinteval{exp(-32.456)/2000}$ (TexFromSci on FracToSci)\par  % math mode required
```

8.026723360322633e-15/2000 (default, i.e. FracToSci)
0.00000000000000802672336032263 3/2000 (FracToDecimal)
$8.026723360322633 \cdot 10^{-15} \cdot 2000^{-1}$ (TexFromSci on FracToSci)

Notice that the /2000 denominator remains ``as is'' in the output, in conformity with the documented behaviour of \xintFracToSci in the first example and of \xintFracToDecimal for the second example. This has not changed since 1.4 (the handling of the numerator part has changed at 1.4e and again slightly at 1.4k, the zero value being now always printed as 0 and not 0 or 0.0 depending on the input) but is to be considered unstable and undecided so far.

A slightly more costly typesetter could be for example:

```
  \def\xintexprPrintOne#1{\xintDecToStringREZ{\xintIrr{#1}}}
```

Then

- the fraction (inclusive of its power of ten part) will be reduced to lowest terms (see \xintIrr),
- next the trailing zeros will be moved as an exponent (positive or negative) to the numerator,
- this numerator with a power of ten part will be printed in decimal fixed point notation, with as few zeros as are needed,
- and finally the denominator B, which has been trimmed of trailing zeros, will be printed as /B or not at all if B=1.

With the use case above:

```
  \def\xintexprPrintOne#1{\xintDecToStringREZ{\xintIrr{#1}}}
  \xinteval{exp(-32.456)/2000}\par
```

0.00000000000000000802672336032263 3/2

This trailing /2 is somewhat of a pain, but as documented and mentioned already \xintDecToStringREZ currently has not been educated to identify its presence and handle it. Slightly faster (see \xintPIrr) is

```
  \def\xintexprPrintOne#1{\xintDecToStringREZ{\xintPIrr{#1}}}
```

which with the used example produces the same output.

One can also consider this for math mode:

```
  \def\xintexprPrintOne#1{\xintTeXFromSci{\xintDecToStringREZ{\xintIrr{#1}}}}
  $\xinteval{exp(-32.456)/2000}$\par
```

$0.00000000000000000802672336032263^{2}$

See our hesitations about what \xintTeXFromSci should do with denominators.

TEX-hackers note: One can hope that in future \xintDecToString will identify denominators being products of only two's and five's, but even then of course \xintTeXFromSci will have to decide how to handle other denominators.

TEX-hackers note: The macro used as customization of \xintexprPrintOne (whose default is a private variant of \xintFracToSci with exactly same output) must understand the internal xintfrac format A/B[N], but with the /B and [N] parts being only optional. This is not a problem when using for this task (nested) macros of xintfrac, as they of course accept such inputs as argument and in fact much more general ones.

In particular one can benefit from \xintRaw, or \xintRawBraced, to convert the argument into a well defined shape (A/B[N] for the former and {N}{A}{B} for the latter) and then work from there.

The macro used by \xintfloatexprPrintOne has the guarantee that the [P] will be always present at expansion time.

The customization should be compatible with being exposed to \expanded (which is like expansion in an \edef), either from being completely expandable or at the opposite from being \protected. LaTeX2e commands defined via \newcommand as macros with one optional parameter are not compatible with this requirement.

Attention! The interface requirements described above for the macros customizing the behaviours of \xintexprPrintOne and \xintfloatexprPrintOne may change at any release... as they depend on some internal structures and it is not certain backwards compatiblity will be maintained systematically in case of evolution.

3.2.2. \xintthealign for output of general oples

code

With \xintthealign one can get nested data use a TEX alignment in the output. Here is an example :

```
  \xintthealign\xintexpr ndseq(1/(i+j), i = 1..10; j=1..10)\relax
[[  1/2,   1/3,   1/4,   1/5,   1/6,   1/7,   1/8,   1/9,   1/10,  1/11  ],
 [  1/3,   1/4,   1/5,   1/6,   1/7,   1/8,   1/9,   1/10,  1/11,  1/12  ],
 [  1/4,   1/5,   1/6,   1/7,   1/8,   1/9,   1/10,  1/11,  1/12,  1/13  ],
 [  1/5,   1/6,   1/7,   1/8,   1/9,   1/10,  1/11,  1/12,  1/13,  1/14  ],
 [  1/6,   1/7,   1/8,   1/9,   1/10,  1/11,  1/12,  1/13,  1/14,  1/15  ],
 [  1/7,   1/8,   1/9,   1/10,  1/11,  1/12,  1/13,  1/14,  1/15,  1/16  ],
 [  1/8,   1/9,   1/10,  1/11,  1/12,  1/13,  1/14,  1/15,  1/16,  1/17  ],
 [  1/9,   1/10,  1/11,  1/12,  1/13,  1/14,  1/15,  1/16,  1/17,  1/18  ],
 [  1/10,  1/11,  1/12,  1/13,  1/14,  1/15,  1/16,  1/17,  1/18,  1/19  ],
 [  1/11,  1/12,  1/13,  1/14,  1/15,  1/16,  1/17,  1/18,  1/19,  1/20  ]]
```

Attention, this \xintthealign must be a prefix to \xintexpr, or \xintfloatexpr etc..., but there will be low-level TEX errors if it is used to prefix \xinteval et al. or \xinttheexpr et al.

It is possible to customize the behaviour of \xintthealign. For example:

```
\protected\def\xintexpralignbegin       {\halign\bgroup\tabskip2ex\hfil##&&##\hfil\cr}%
\def\xintexpralignend                   {\crcr\egroup}% removed \protected at 1.4c
\protected\def\xintexpralignlinesep     {,\cr}% separates "lines"
\protected\def\xintexpralignleftsep     {&}% at left of first item in a "line"
                                           % (after "left bracket")
\protected\def\xintexpraligninnersep    {,&}% at the left of non-first items
\protected\def\xintexpralignrightsep    {&}% at right of last item in a "line"
                                           % (before "right bracket")
\protected\def\xintexpralignleftbracket {[}%
\protected\def\xintexpralignrightbracket{]}%
```

The above definitions use \protected with no strong reason, as the replacement tokens are not expanding anyhow, but the idea is that this allows to execute a computation via an \edef and later one can change the meaning of the auxiliary macros depending on what one wants to do with the expansion result.

TEX-hackers note: \xintexpralignend is expanded once, after the body has been submitted to exhaustive expansion (\expanded induced), and prior to the expansion of \xintexpralignbegin.

UNSTABLE!

Although we will try to keep stable the way "regular arrays" as in the above example are rendered by default, the \xintthealign macro (and its associated customizability) is to be considered work-in-progress and may experience breaking changes.

Use for example this for outputting to a file or a terminal:[9]

```
% Better here without \protected.
% We assume here \newlinechar has the LaTeX setting.
\def\xintexpralignbegin       {}%
\def\xintexpralignend         {}%
\def\xintexpralignlinesep     {,^^J}% separates "lines"
\def\xintexpralignleftsep     { }% at left of first item in a "line" (after brackets)
\def\xintexpraligninnersep    {, }% at the left of non-first items
\def\xintexpralignrightsep    { }% at right of last item in a "line" (before brackets)
\def\xintexpralignleftbracket {[}%
\def\xintexpralignrightbracket{]}%
```

In the LATEX example next using a pmatrix environment, \noexpand rather than \protected is used. This environment will not break across pages, contrarily to the display produced by the default

[9] With the xetex engine this will need its -8bit option else the ^^J in \xintexpralignlinesep will be printed literally instead of being converted into a line separator in the file or terminal output.

\xintthealign configuration which uses TEX's \halign.

```
\[
\def\xintexpralignbegin       {\begin{pmatrix}}%
\def\xintexpralignend         {\end{pmatrix}}%
\def\xintexpralignlinesep     {\noexpand\\}% needed to counteract an internal \expanded
\def\xintexpralignninnersep   {&}%
\let\xintexpralignleftbracket\empty  \let\xintexpralignleftsep\empty
\let\xintexpralignrightbracket\empty \let\xintexpralignrightsep\empty
% by default amsmath matrices can have 10 columns at most
% (cf amsmath documentation for what to do to allow more)
l.c.m.=\xintthealign\xintiiexpr  ndmap(lcm, 1..12; 1..10)\relax
\]
```

$$\text{l.c.m.} = \begin{pmatrix}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
2 & 2 & 6 & 4 & 10 & 6 & 14 & 8 & 18 & 10 \\
3 & 6 & 3 & 12 & 15 & 6 & 21 & 24 & 9 & 30 \\
4 & 4 & 12 & 4 & 20 & 12 & 28 & 8 & 36 & 20 \\
5 & 10 & 15 & 20 & 5 & 30 & 35 & 40 & 45 & 10 \\
6 & 6 & 6 & 12 & 30 & 6 & 42 & 24 & 18 & 30 \\
7 & 14 & 21 & 28 & 35 & 42 & 7 & 56 & 63 & 70 \\
8 & 8 & 24 & 8 & 40 & 24 & 56 & 8 & 72 & 40 \\
9 & 18 & 9 & 36 & 45 & 18 & 63 & 72 & 9 & 90 \\
10 & 10 & 30 & 20 & 10 & 30 & 70 & 40 & 90 & 10 \\
11 & 22 & 33 & 44 & 55 & 66 & 77 & 88 & 99 & 110 \\
12 & 12 & 12 & 12 & 60 & 12 & 84 & 24 & 36 & 60
\end{pmatrix}$$

3.3. Built-in operators and their precedences

The parser implements precedence rules based on concepts which are summarized below (only for binary infix operators):

- an infix operator has two associated precedence levels, say L for left and R for right,
- the parser proceeds from left to right, pausing each time it has found a new number and an operator following it,
- the parser compares the left-precedence L of the new found operator to the right-precedence R_last of the last delayed operation (which already has one argument and would like to know if it can use the new found one): if L is at most equal to it, the delayed operation is now executed, else the new-found operation is kept around to be executed first, once it will have gathered its arguments, of which only one is known at this stage.

This means for example in the case of the multiplication * and the division operators /, //, /: that they are parsed in a left-associative way because they all share the same (left and right) precedence level. This is the case with the analogous operators from the Python language, as well.

At 1.4g the power operators were changed to act in a right associative way. Again, this matches the behaviour of e.g. Python:

```
\xinteval{2^-3^4}
```

1/241785163922925834941235 2

The entries of Table 2 are hyperlinked to the more detailed discussion at each level. In these entries the number within parentheses indicates the right-precedence, if it differs from the left.

∞ At this highest level of precedence, one finds:

functions and variables Functions (even the logic functions !() and ?() whose names consist of a single non-letter character) must be used with parentheses. These parentheses may arise

∞: at this top level the syntax elements whose execution is done prior to operators preceding them:

- built-in or user-defined functions,
- variables,
- the * unpacking operator,
- intrinsic constituents of numbers: decimal mark ., e and E of scientific notation, hexadecimal prefix ".

Precedence	``Operators'' at this level
20	postfix ! and branching ?, ?? operators
-	minus sign as unary operator inherits the right-precedence of the infix operator it follows, if that precedence is higher than the one of binary + and -, else it inherits the latter
18 (17)	^ and ** are synonymous; they act in a right-associative way (**Changed at 1.4g!**)
16 (14)	Tacit multiplication has an elevated (left) precedence
14	*, /, // (floored division), and /: (associated modulo, alias 'mod')
12	+, -
10	<, >, ==, <=, >=, != (they can be chained)
8	Boolean conjunction && and its alias 'and'
6	Boolean disjunction || and its alias 'or'. Also 'xor' and .., ..[,].., and : have this precedence
4	the brackets for slicers and extractors [,]
3	the comma ,
2	the bracketers [,] construct nestable "arrays"
1	the parentheses (,), and the semi-colon ; in iter(), rseq(), and further structures

- Binary operators have a left and a right precedence, which for most coincide. The right precedence is indicated within parentheses.
- Tacit multiplication has an elevated left precedence level: (1+2)/(3+4)5 is computed as (1+2)/((3+4)*5) and x/2y is interpreted as x/(2*y) when using variables.

Table 2: Precedence levels

from expansion after the function name is parsed (there are exceptions which are documented at the relevant locations.)

`*` Python-like "unpacking" prefix operator. Sometimes one needs to use it as function `*()` (but I can't find an example right now) but most of the time parentheses are unneeded.

`.` is decimal mark; the number scanner treats it as an inherent, optional and unique component of a being formed number. `\xintexpr 0.^2+2^.0\relax` is interpreted as `0^2+2^0` and thus produces `1`.

Since release `1.2` an isolated decimal mark is illegal input in the xintexpr parsers (it remains legal as argument to the macros of xintfrac).

`e` scientific notation.

`E` scientific notation. For output, see `\xintPFloatE`.

`"` prefix for hexadecimal input. Only uppercase letters, and one optional `.` separating integer and fractional hexadecimal parts.

```
  \xintexpr "FEDCBA9876543210\relax\newline
  \xintexpr ".FEDCBA9876543210\relax\newline
  \xintexpr 16^5-("F75DE.0A8B9+"8A21.F5746+16^-5)\relax
```

```
18364758544493064720
0.995555555555555555559410496613281793543137609958648681640625
0
```

It is possible that in future the `"` prefix could be dropped in favour of `0x` prefix. This would free `"` to be used for input of "string"-like entities.

20 The postfix operators `!` and the branching conditionals `?`, `??`.

`!` computes the factorial of an integer.

`?` is used as `(stuff)?{yes}{no}`. It evaluates `stuff` and chooses the `yes` branch if the result is non-zero, else it executes `no`. After evaluation of `stuff` it acts as a macro with two mandatory arguments within braces, chooses the correct branch *without evaluating the wrong one*. Once the braces are removed, the parser scans and expands the uncovered material.

`??` is used as `(stuff)??{<0}{=0}{>0}`, where `stuff` is anything, its sign is evaluated and depending on the sign the correct branch is un-braced, the two others are discarded with no evaluation of their contents.

- As unary operator, the minus sign inherits as precedence the minimum of `12` (which is the precedence for addition and subtraction) and of the (right-) precedence of the operators preceding it (if any).

```
  \xintexpr -3-4*-5^-7, (-3)-(4*(-(5^(-7))))\relax\newline
  \xintexpr -3^-4*-5-7, (-((3^(-4))*(-5)))-7\relax\newline
  |2^-10| gives \xintexpr 2^-10\relax\space
```

```
-234371/78125, -234371/78125
-562/81, -562/81
2^-10 gives 1/1024 and is thus perfectly legal, no need for parentheses.
```

The `+` character as prefix unary operator is simply ignored during input parsing.

18

`^`

`**` Both compute powers. They act in a right associative way.

```
  \xintiiexpr 2^3^4\relax
```

```
2417851639229258349412352
```

16 see Tacit multiplication.

14

`*` multiplication

`/` division:

- in \xinteval: exact division in the field of rational numbers (not automatically reduced to lowest terms),
- in \xintfloateval: correct rounding of the exact division; the two operands are, if necessary, float-rounded before the fraction is evaluated and rounded (to obtain the correcty rounded A/B without prior rounding of A and B see qfloat()),
- in \xintiieval: for compatibility with the legacy behaviour of / in \numexpr, it rounds the exact fraction *with half-integers going towards the infinity of the same sign*.

The division is left-associative. Example:

```
  \xintexpr reduce(100/50/2)\relax
1
```

`//` floored division (and thus produces an integer, see divmod() for details)

`/:` the associated modulo (see divmod() and mod())

Left-associativity applies to the division operators:

```
  \xintexpr 100000/:13, 100000 'mod' 13\relax, \xintexpr 100000/:13/13\relax
4, 4, 4/13
```

Nothing special needs to be done in contexts such as LaTeX3 \ExplSyntaxOn where : is of catcode letter, but if : is an active character one needs to use input such as /\string : (or replace it with usage of the function mod()).

NEW WITH 1.4n

Bulky workarounds such as /\string : are *unneeded* if activation is due to Babel. See also \xintexprSafeCatcodes and a framed note found in subsection 4.2.

'mod' is same as /:.



Attention: with polexpr loaded, which allows ' in variable and function names, 'mod' can not follow a variable name. Add parentheses around the variable, or use /:.

12

`+` addition

`-` subtraction. According to the general left-associativity rule in case of equal precedence, it is left associative:

```
  \xintiiexpr 100-50-2\relax
48
```

10 Comparison operators are (as in Python) all at the same level of precedence, use parentheses for disambiguation.

`<` a<b evaluates to 1 if the strict inequality holds to 0 if not.

`>` a>b evaluates to 1 if the strict inequality holds to 0 if not.

`==` a==b evaluates to 1 if equality holds to 0 if not.

`<=` a<=b evaluates to 1 if left hand side is at most equal to right hand side, to 0 if not.

`>=` a>=b evaluates to 1 if left hand side is at least equal to right hand side, to 0 if not.

`!=` a!=b evaluates to 1 if they differ, to 0 if not.

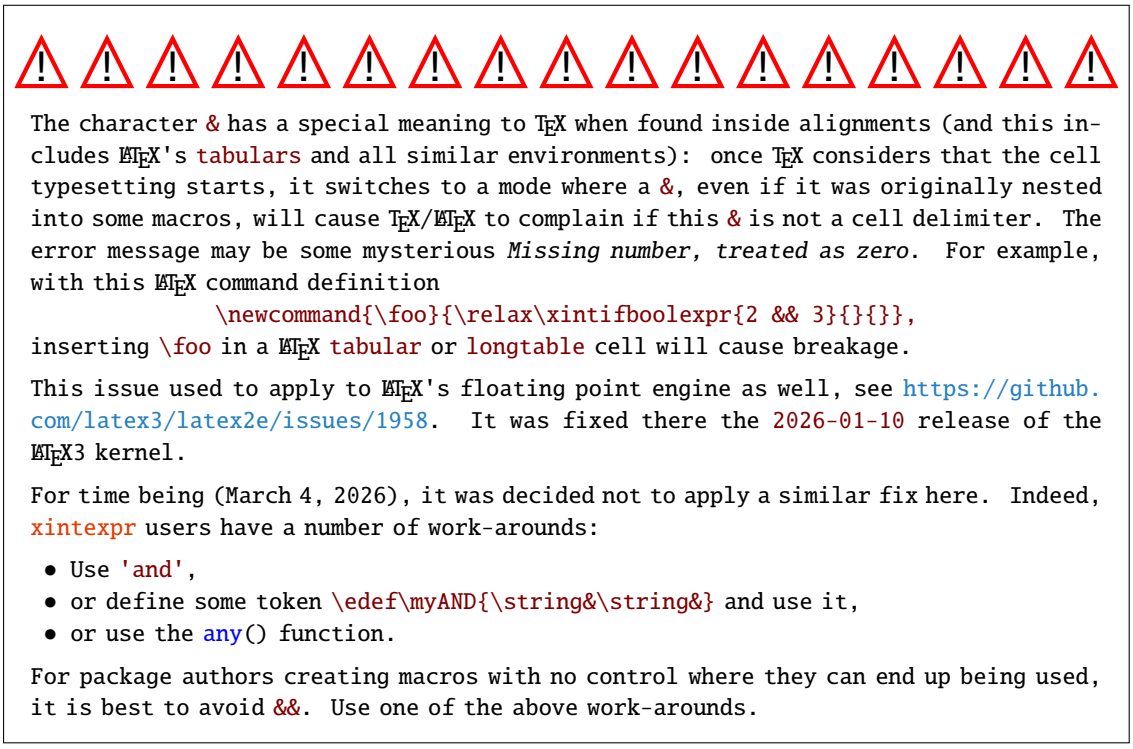
Comparisons can be chained arbitrarily, e.g., `x < y <= z != t` is equivalent to `x < y 'and' y <= z 'and' z != t` (and also to `all(x<y, y<=z, z!=t)`), except that if `y` and `z` involve computations, they are evaluated only once. Currently there is no short-circuit here, i.e. even if some intermediate comparison turns out false (in fact `0`), all the remaining conditionals will still be evaluated.

```
  \xintifboolexpr{1<=2!=3<4>1}{true}{\error},
  \xintifboolexpr{1<=2>=3<4>1}{\error}{false},
  \xintifboolexpr{3 != 3! == 6 != 4! == 24}{true}{\error}
true, false, true
```

8

&& logical conjunction. Evaluates to `1` if both sides are non-zero, to `0` if not.

'and' same as `&&`. See also the `all()` multi-arguments function.



The character & has a special meaning to TEX when found inside alignments (and this includes LATEX's tabulars and all similar environments): once TEX considers that the cell typesetting starts, it switches to a mode where a &, even if it was originally nested into some macros, will cause TEX/LATEX to complain if this & is not a cell delimiter. The error message may be some mysterious *Missing number, treated as zero*. For example, with this LATEX command definition

```
\newcommand{\foo}{\relax\xintifboolexpr{2 && 3}{}{}},
```

inserting `\foo` in a LATEX tabular or longtable cell will cause breakage.

This issue used to apply to LATEX's floating point engine as well, see https://github.com/latex3/latex2e/issues/1958. It was fixed there the 2026-01-10 release of the LATEX3 kernel.

For time being (March 4, 2026), it was decided not to apply a similar fix here. Indeed, xintexpr users have a number of work-arounds:

- Use `'and'`,
- or define some token `\edef\myAND{\string&\string&}` and use it,
- or use the `any()` function.

For package authors creating macros with no control where they can end up being used, it is best to avoid `&&`. Use one of the above work-arounds.

Nota bene: with polexpr loaded, which allows ' in variable and function names, something such as `foo 'and' bar` will break because xintexpr will try to find a variable named `foo'and'bar` and fail. The simplest workaround is to use `(foo) 'and' (bar)`. An alternative syntax is `all(foo,bar)` or of course `foo && bar` but see the above explanations relative to alignments.

6

|| logical (inclusive) disjunction. Evaluates to `1` if one or both sides are non-zero, to `0` if not.

'or' same as as `||`. See also the `any()` multi-arguments function.



Attention: with polexpr loaded, which allows ' in variable and function names, `'or'` can not follow a variable name. Add parentheses around the variable, or use `||`.

'xor' logical (exclusive) disjunction.

☞ { Attention: with polexpr loaded, which allows ' in variable and function names, 'xor' can not follow a variable name. Add parentheses around the variable, or use the xor() function syntax.

..

..[

].. Syntax for arithmetic progressions. See subsection 3.5.

: This is a separator involved in [a:b] Python-like slicing syntax.

4

[

] Involved in Python-like slicing [a:b] and extracting [N] syntax. And its extension à la NumPy [a:b,N,c:d,...,:]. Ellipsis ... is not yet implemented. The "step" parameter as in [a:b:step] is not yet implemented.

3

, The comma separates expressions (or function arguments).[10]

```
  \xintiiexpr 2^3,3^4,5^6\relax
8, 81, 15625
```

2

[

] The bracketers construct nestable "array-like" structures. Arbitrary (heterogeneous) nesting is allowed. For output related matters see \xintthealign (its usage is optional, without it rendering is "one-dimensional"). Output shape of non-homogeneous arrays is to be considered unstable at this time.

1

(

) The parentheses serve as mandatory part of the syntax for functions, and to disambiguate precedences.[11] They do not construct any nested structure.

; The semi-colon as involved as part of the syntax of iter(), rseq(), ndseq(), ndmap() has the same precedence as a closing parenthesis.

\relax This is the expression terminator for \xintexpr et al. It may arise from expansion during the parsing itself. As alternative to \xintexpr (et al.) use \xinteval (et al.) which have the usual macro interface (with one mandatory argument).

The ; also serves as syntax terminator for \xintdefvar and \xintdeffunc. It can in this rôle not arise from expansion as the expression body up to it is fetched by a delimited macro. But this is done in a way which does not require any specific hiding for inner semi-colons as involved in the syntax of iter(), etc...

3.4. Built-in functions

See Table 3 whose elements are hyperlinked to the corresponding definitions.

Functions are at the same top level of priority. All functions even ?() and !() require parentheses around their arguments.

[10] The comma is really like a binary operator, which may be called "join". It has lowest precedence of all (apart the parentheses) because when it is encountered all postponed operations are executed in order to finalize its *first* operand; only a new comma or a closing parenthesis or the end of the expression will finalize its *second* operand. [11] It is not apt to describe the opening parenthesis as an operator, but the closing parenthesis is analogous to a postfix unary operator. It has lowest precedence which means that when it is encountered all postponed operations are executed to finalize its operand. The start of this operand was decided by the opening parenthesis.

!()	arctg()	ch()	floor()	log10()	qfrac()	sind()
?()	argch()	cos()	frac()	log1p()	qint()	sinh()
`*`()	argcoth()	cosd()	gcd()	max()	qrand()	sqr()
`+`()	argcsch()	cosh()	if()	min()	qraw()	sqrt()
Arg()	argsech()	cot()	ifint()	mod()	random()	sqrtr()
Argd()	argsh()	cotd()	ifone()	mul()	randrange()	subs()
abs()	argth()	cotg()	ifsgn()	ndfillraw()	rbit()	subsm()
acos()	asec()	coth()	ilog10()	ndmap()	reduce()	subsn()
acosd()	asech()	csc()	inv()	ndseq()	reversed()	tan()
acosh()	asin()	cscd()	iquo()	not()	round()	tand()
acoth()	asind()	csch()	irem()	num()	rrseq()	tanh()
acsc()	asinh()	divmod()	isint()	nuple()	rseq()	tg()
acsch()	atan()	even()	isone()	odd()	sec()	th()
add()	atan2()	exp()	iter()	pArg()	secd()	togl()
all()	atan2d()	expm1()	iterr()	pArgd()	sech()	trunc()
any()	atand()	factorial()	last()	pfactorial()	seq()	unpack()
arccos()	atanh()	first()	lcm()	pow()	sgn()	xor()
arccsc()	binomial()	flat()	len()	pow10()	sh()	zip()
arcsec()	bool()	float()	ln()	preduce()	sin()	
arcsin()	ceil()	float_dgt()	log()	qfloat()	sinc()	

Table 3: Functions (click on names)

.4.1 Functions with no argument . . . 27
.4.2 Functions with one argument . . . 28
.4.3 Functions with an alphanumeric argument . . . 31
.4.4 Functions with one mandatory and a second but optional argument . . . 32
.4.5 Functions with two arguments . . . 33
.4.6 Functions with 3 or 4 arguments . . . 35
.4.7 Functions with an arbitrary number of arguments . . . 36
.4.8 Functions requiring dummy variables . . . 38

Miscellaneous notes:

- since release 1.3d gcd() and lcm() are extended to apply to fractions too, and do NOT require the loading of xintgcd,
- The randomness related functions random(), qrand() and randrange() require that the TEX engine provides the \uniformdeviate or \pdfuniformdeviate primitive. This primitive is abstracted into \xinttexuniformdeviate, but the functions use a wrapper \xintUniformDeviate which provides more randomness.
- togl() is provided for the case etoolbox package is loaded,
- bool(), togl() use delimited macros to fetch their argument and the closing parenthesis must be explicit, it can not arise from on the spot expansion. The same holds for qint(), qfrac(), qfloat(), qraw(), random() and qrand().
- Also functions with dummy variables use delimited macros for some tasks. See the relevant explanations there.
- Functions may be called with *ople*s as arguments as long as the total length is the number of arguments the function expects.

3.4.1. Functions with no argument

random() returns a random float x verifying 0 <= x < 1. It obeys the prevailing precision as set by \xintDigits: i.e. with P being the precision the random float multiplied by 10^P is an integer, uniformly distributed in the 0..10^P-1 range.

This description implies that if x turns out to be <0.1 then its (normalized) mantissa has P-1 digits and a trailing zero, if x<0.01 it has P-2 digits and two trailing zeros, etc... This is what is observed also with Python's random(), of course with 10 replaced there by radix 2.

```
    \xintsetrandomseed 12345
    \xintDigits:=37\relax
    \xintthefloatexpr random()\relax\newline
    \xintthefloatexpr random()\relax\par
0.24155448175962074555479298502095000 42
0.25848635299939966272854615542030213 52
```

qrand() returns a random float 0 <= x < 1 using 16 digits of precision (i.e. 10^{16}x is an integer). This is provided when speed is a at premium as it is optimized for precision being precisely 16.

```
    % still with 37 digits as prevailing float precision
    \xintthefloatexpr qrand(), random()\relax\newline
    \xintDigits:=16\relax
    \xintthefloatexpr qrand(), random()\relax\par
0.4883568991327765, 0.09165461826072383107532471669335645230
0.9069127435402274, 0.9106687541716861
```

One can use both qrand() and random() inside the \xintexpr parser too. But inside the integer only \xintiiexpr parser they will cause some low-level error as soon as they get involved in any kind of computation as they use an internal format not recognized by the integer-only parser.

See further randrange(), which generates random integers.

Currently there is no uniform() function[12] but it can be created by user:

```
\xintdefloatfunc uniform(a, b):= a + (b-a)*random();
\romannumeral\xintreplicate{10}%
{%
```

[12] Because I am not sure how to handle rounding issues: should the computation proceed exactly and a rounding be done only at very end?

```
    \xintthefloatexpr uniform(123.45678, 123.45679)\relax\newline
 }%
123.4567849497100
123.4567812033226
123.4567863308250
123.4567896366777
123.4567849656655
123.4567849908270
123.4567889123433
123.4567896262979
123.4567846543719
123.4567832664043
```

rbit() returns a random `0` or `1`.

3.4.2. Functions with one argument

> The trigonometric and exponential functions are not included here. See the calxtrig and calxlog chapters.

num(x) truncates to the nearest integer (truncation towards zero). It has the same sign as `x`, except of course with `-1<x<1` as then `num(x)` is zero.

```
  \xinttheexpr num(3.1415^20), num(1e20)\relax
```

8764785276, 100000000000000000000 The output is an explicit integer with as many zeros are as necessary. Even in float expressions, there will be an intermediate stage where all needed digits are there, but then the integer is immediately reparsed as a float to the target precision, either because some operation applies to it, or from the output routine of \xintfloatexpr if it stood there alone. Hence, inserting something like `num(1e10000)` is costly as it really creates ten thousand zeros, even though later the whole thing becomes a float again. On the other hand naturally `1e10000` without `num()` would be simply parsed as a floating point number and would cause no specific overhead.

frac(x) fractional part. For all numbers `x=num(x)+frac(x)`, and `frac(x)` has the same sign as `x` except when `x` is an integer, as then `frac(x)` vanishes.

```
  \xintthefloatexpr frac(-355/113), frac(-1129.218921791279)\relax
-0.1415929203539820, -0.218921791279
```

reduce(x) reduces a fraction to smallest terms

```
  \xinttheexpr reduce(50!/20!/20!/10!)\relax
```

141599788880796185940 0

Recall that this is NOT done automatically, for example when adding fractions.

preduce(x) internally, fractions may have some power of ten part (for example when they got input in scientific notation). This function ignores the decimal part when doing the reduction. See \xintPIrr.

```
  \xinttheexpr preduce(10e7/2), reduce(10e7/2)\relax
5e7, 50000000
```

abs(x) absolute value

sgn(x) sign. See also \xintifsgnexpr.

inv(x) inverse.

floor(x) floor function.

ceil(x) ceil function.

sqr(x) square.

ilog10(x) in \xintiiexpr the integer exponent a such that $10^a \le \mathrm{abs}(x) < 10^{a+1}$; returns (this may evolve in future) -2147450880 if x vanishes (i.e. 0x7fff8000).

```
  \xintiieval{ilog10(1), ilog10(-1234567), ilog10(-123456789123456789), ilog10(2**31)}\par
0, 6, 17, 9
```

See ilog10() for the behaviour in \xintexpr-essions.

sqrt(x) in \xintiiexpr, truncated square root; in \xintexpr or \xintfloatexpr this is the floating point square root, and there is an optional second argument for the precision. See sqrt().

sqrtr(x) available *only* in \xintiiexpr, rounded square root.

factorial(x) factorial function (like the post-fix ! operator.) When used in \xintexpr or \xintfloatexpr there is an optional second argument. See factorial().

?(x) is the truth value, 1 if non zero, 0 if zero. Must use parentheses.

!(x) is logical not, 0 if non zero, 1 if zero. Must use parentheses.

not(x) logical not.

even(x) is the evenness of the truncation num(x).

```
  \xintthefloatexpr [3] seq((x,even(x)), x=-5/2..[1/3]..+5/2)\relax
-2.50, 1, -2.17, 1, -1.83, 0, -1.50, 0, -1.17, 0, -0.833, 1, -0.500, 1, -0.167, 1, 0.167, 1,
0.500, 1, 0.833, 1, 1.17, 0, 1.50, 0, 1.83, 0, 2.17, 1, 2.50, 1
```

odd(x) is the oddness of the truncation num(x).

```
  \xintthefloatexpr [3] seq((x,odd(x)), x=-5/2..[1/3]..+5/2)\relax
-2.50, 0, -2.17, 0, -1.83, 1, -1.50, 1, -1.17, 1, -0.833, 0, -0.500, 0, -0.167, 0, 0.167, 0,
0.500, 0, 0.833, 0, 1.17, 1, 1.50, 1, 1.83, 1, 2.17, 0, 2.50, 0
```

isint(x) evaluates to 1 if x is an integer, to 0 if not. See ifint().

```
  $\xinttheexpr -5/3..[1/3]..+5/3\relax
  \rightarrow \xinttheexpr seq(isint(x), x=-5/3..[1/3]..+5/3)\relax$
-5/3, -4/3, -3/3, -2/3, -1/3, 0, 1/3, 2/3, 3/3, 4/3, 5/3 → 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0
```

isone(x) evaluates to 1 if x is 1, to 0 if not. See ifone().

```
  $\xintthefloatexpr subs(((x-1)/x, x/x, (x+1)/x), x=2**30)\relax
  \rightarrow
  \xintthefloatexpr seq(isone(y), y=subs(((x-1)/x, x/x, (x+1)/x), x=2**30))\relax$
0.9999999990686774, 1, 1.000000000931323 → 0, 1, 0
```

qint(x) belongs with qfrac(), qfloat(), qraw() to a special category and it may have been an error to start documenting them. In particular qraw() is basically a "DO NOT USE" interface.

1. They require the closing parenthesis of their argument to be immediately visible, it can not arise from expansion.

2. They grab the argument and store it directly; the format must be compatible with what is expected at macro level.
3. And in particular the argument can not be a variable, it has to be numerical.

qint() achieves the same result as num, but the argument is grabbed as a whole without expansion and handed over to the \xintiNum macro. The q stands for ``quick'', and qint is thought out for use in \xintiiexpr...\relax with integers having dozens of digits.

Testing (from eons ago) showed that using qint() starts getting advantageous for inputs having more (or *f*-expanding to more) than circa 20 explicit digits. But for hundreds of digits the input gain becomes a negligible proportion of (for example) the cost of a multiplication.

Leading signs and then zeroes will be handled appropriately but spaces will not be systematically stripped. They should cause no harm and will be removed as soon as the number is used with one of the basic operators. This input mode *does not accept decimal part or scientific part*.

```
\def\x{....many many many ... digits}\def\y{....also many many many digits...}
\xinttheiiexpr qint(\x)*qint(\y)+qint(\y)^2\relax\par
```

CHANGED AT 2.0a1!

qfrac(x) accepts fractions. **BREAKING CHANGE:** if used in \xintfloateval or \xintiieval the argument MUST be of the type A/B with *strict* integers A and B with no leading zeros or signs, only A may have a minus sign (in short A/B[0] must be xintfrac "raw format"). **ONLY** if used within \xinteval will it be parsed as done in general by the xintfrac macros.

As per qfloat(), if qfrac() is used within an \xintiiexpr-ession, it must be hidden inside functions such as round() or trunc() which then produce integers acceptable to the integer-only parser.

We mention that qfrac() has nothing to do with the similarly named function frac() (sigh...).

qfloat(x) main aim is to be able to input a fraction A/B to \xintfloateval and have it correctly rounded without prior separate rounding of numerator and denominator. This can also be achieved by nesting a sub \xintexpr A/B\relax, which though has a bit of overhead and is not available if only calx is loaded.

CHANGED AT 2.0a1!

BREAKING CHANGE: it is **not** recognized anymore in \xintiieval. And within \xintfloateval the only recognized input formats are A, A[N], A/B, and A/B[N]. As per \xinteval, there its argument can be anything parsed on input generally speaking by the xintfrac macros, as for qfloat().

Let us give two examples. The next computations done with 16 digits of precision do not give the same result:

```
\calxeval{qfloat(1234567812345678 5001/1234567812345678 4999)}${}\neq{}$%
\calxeval{1234567812345678 5001/1234567812345678 4999}\newline
\calxeval{1234567812345679/1234567812345678}${}\neq{}$%
\calxeval{\xintexpr 1234567812345678 5001/1234567812345678 4999\relax}\newline
```

$1 \neq 1.000000000000001$
$1.000000000000001 \neq 1$

because the second is equivalent to the third, whereas the first one is equivalent to the fourth one.

float_dgt(x) is like float() and avoids float()'s check whether it used with its second optional argument. This is useful in the context of converting function definitions done via \xintdeffunc (see explanations there) to functions usable in \xintfloateval.

DO NOT USE! (1.4)

nuple(x) is currently same as [...]. Reserved for possible alternative meaning in future.

```
\xinteval{nuple(1,2,3)}
```

[1, 2, 3]

unpack(x) is alternative for * unpacking operator.

```
\xinteval{unpack([1,2,3])}
```
```
1, 2, 3
```

flat(ople) removes all nesting to produce a (non-bracketed) ople having the same leaves (some possibly empty) but located at depth 1.

```
\xinteval{flat([[[[1,[],3],[4,[[[5,6,[]],[8,9],[[],11]],12],[13,14]]], [[],16]]], [])}
```
```
1, [], 3, 4, 5, 6, [], 8, 9, [], 11, 12, 13, 14, [], 16, []
```

Unstable?

I almost delayed indefinitely release because I was hesitating on the name: perhaps better with flattened(), but long names add (negligible, but still) overhead compared to short names. For this reason, consider that name may change.

3.4.3. Functions with an alphanumeric argument

bool(name) returns 1 if the TEX conditional \ifname would act as \iftrue and 0 otherwise. This works with conditionals defined by \newif (in TEX or LATEX) or with primitive conditionals such as \ifmmode. For example:

```
\xintifboolexpr{25*4-if(bool(mmode),100,75)}{YES}{NO}
```

will return NO if executed in math mode (the computation is then 100-100 = 0) and YES if not (the if() conditional is described below; the \xintifboolexpr test automatically encapsulates its first argument in an \xintexpr and follows the first branch if the result is non-zero (see subsection 4.15)).

The alternative syntax 25*4-\ifmmode100\else75\fi could have been used here, the usefulness of bool(name) lies in the availability in the \xintexpr syntax of the logic operators of conjunction &&, inclusive disjunction ||, negation ! (or not), of the multi-operands functions all, any, xor, of the two branching operators if and ifsgn (see also ? and ??), which allow arbitrarily complicated combinations of various bool(name).

togl(name) returns 1 if the LATEX package etoolbox[13] has been used to define a toggle named name, and this toggle is currently set to true. Using togl in an \xintexpr..\relax without having loaded etoolbox will result in an error from \iftoggle being a non-defined macro. If etoolbox is loaded but togl is used on a name not recognized by etoolbox the error message will be of the type ``ERROR: Missing \endcsname inserted.'', with further information saying that \protect should have not been encountered (this \protect comes from the expansion of the non-expandable etoolbox error message).

When bool or togl is encountered by the \xintexpr parser, the argument enclosed in a parenthesis pair is expanded as usual from left to right, token by token, until the closing parenthesis is found, but everything is taken literally, no computations are performed. For example tog
↪ l(2+3) will test the value of a toggle declared to etoolbox with name 2+3, and not 5. Spaces are gobbled in this process. It is impossible to use togl on such names containing spaces, but \iftoggle{name with spaces}{1}{0} will work, naturally, as its expansion will pre-empt the \xintexpr scanner.

There isn't in \xintexpr... a test function available analogous to the test{\ifsometest} construct from the etoolbox package; but any *expandable* \ifsometest can be inserted directly in an \xintexpr-ession as \ifsometest10 (or \ifsometest{1}{0}), for example if(\ifsometest{1}{
↪ 0},YES,NO) (see the if operator below) works.

A straight \ifsometest{YES}{NO} would do the same more efficiently, the point of \ifsomete
↪ st10 is to allow arbitrary boolean combinations using the (described later) && and || logic operators: \ifsometest10 && \ifsomeothertest10 || \ifsomethirdtest10, etc... YES or NO above stand for material compatible with the \xintexpr parser syntax.

See also \xintifboolexpr, in this context.

[13] https://ctan.org/pkg/etoolbox

3.4.4. Functions with one mandatory and a second but optional argument

round(x[, n]) Rounds its first argument to an integer multiple of 10^(-n) (i.e. it *quantizes*). The case of negative n is new with 1.4a. Positive n corresponds to conversion to a fixed point number with n digits after decimal mark.

```
  \xinteval{round(-2^30/3^5,12), round(-2^30/3^5,-3)}
-4.418690633744855967e6, -4.419e6
```

trunc(x[, n]) Truncates its first argument to an integer multiple of 10^(-n). The case of negative n is new with 1.4a.

```
  \xinteval{trunc(-2^30/3^5,12), trunc(-2^30/3^5,-3)}
-4.418690633744855967e6, -4.418e6
```

float(x[, n]) Rounds its first argument to a floating point number, with a precision given by the second argument, which must be positive.

```
  \xinteval{float(-2^30/3^5,12), float(-2^30/3^5, 1)}
-4.41869063374e6, -4e6
```

For this example and earlier ones if the parser had been \xintfloateval, not \xinteval, the first argument (here 2^30/3^5) would already have been computed as floating point number with numerator and denominator rounded separately first to the prevailing precision. To avoid that, use \xintexpr...\relax wrapper. Then the rounding or truncation will be applied to an exact fraction.

sfloat(x[, n]) It is the same as float(), but in case of a short (non-fractional) input it gets stored internally without adding zeros to make the mantissa have the \xinttheDigits length. One may wonder then what is the utility of sfloat()? See for an example of use the documentation of \xintdeffunc. Notice however that this is a bit experimental and may evolve in future when xint gets a proper internal data structure for floating point numbers. The non-normalized format is useful for multiplication or division, but float additions and subtractions usually convert their arguments to a normalized mantissa.

ilog10(x[, n]) If there is an optional argument n, returns the (relative) integer a such that $10^a \leq$ abs(float(x,n)) $< 10^{a+1}$. In absence of the optional argument:

- in \xintexpr, it returns the exponent a such that $10^a \leq$ abs(x) $< 10^{a+1}$.
- in \xintfloatexpr, the input is first rounded to \xinttheDigits float precision, then the exponent a is evaluated.

```
  \xintfloateval{ilog10(99999999/10000000, 8), ilog10(-999999995/100000000, 8),
                                               ilog10(-999999995/100000000, 9)}\newline
  \xinteval{ilog10(-999999995/100000000), ilog10(-999999995/100000000, 8)}
0, 1, 0
0, 1
```

If the input vanishes the function outputs -2147450880 (i.e. -0x7fff8000 which is near the minimal TeX number -0x7fffffff). This is also subject to change.

The integer-only variant for \xintiiexpr admits no optional argument.

sqrt(x[, n]) in \xintexpr...\relax and \xintfloatexpr...\relax it achieves the precision given by the optional second argument. For legacy reasons the sqrt function in \xintiiexpr *truncates* (to an integer), whereas sqrt in \xintfloatexpr...\relax (and in \xintexpr...\relax which borrows it) *rounds* (in the sense of floating numbers). There is sqrtr in \xintiiexpr for *rounding* to nearest integer.

```
  \xinttheexpr sqrt(2,31)\relax\ and \xinttheiiexpr sqrt(num(2e60))\relax
```

1.41421356237309504880168872421 and 141421356237309504880168724209

There is an integer only variant for \xintiiexpr.

factorial(x[, n]) when the second optional argument is made use of inside \xintexpr...\relax, this switches to the use of the float version, rather than the exact one.

```
\xinttheexpr factorial (100,32)\relax, {\xintDigits:=32\relax \xintthefloatexpr
                                          factorial (100)\relax}\newline
\xinttheexpr factorial (50)\relax\newline
\xinttheexpr factorial (50, 32)\relax
```

```
9.33262154439441526816992388562 67e157, 9.33262154439441526816992388562 67e157
30414093201713378043612608166064768844377641568960512000000000000
3.041409320171337804361260816606 5e64
```

The integer only variant of course has no optional second argument.

randrange(A[, B]) when used with a single argument A returns a random integer 0 <= x < A, and when used with two arguments A and B returns a random integer A <= x < B. As in Python it is an "empty range" error in first case if A is zero or negative and in second case if B <= A.

Attention that the arguments are first converted to integers using \xintNum (i.e. truncated towards zero).

The function can be used in all three parsers. Of course the size is not limited (but in the float parser, the integer will be rounded if involved in any operation).

```
    \xintsetrandomseed 12345
    \xinttheiiexpr randrange(10**20)\relax\newline
    \xinttheiiexpr randrange(1234*10**16, 1235*10**16)\relax\newline
    \printnumber{\xinttheiiexpr randrange(10**199,10**200)\relax}\par
```

```
12545314555479298502
12341249468233524155
38724271496566552250944896366777081662436330824968873373120332258200044549497099786643316
↪ 91066875417168619069127435402274480091654618260723831075324716693356452348835689913277656
↪ 39525848635299939966272 8
```

For the support macros see \xintRandomDigits, \xintiiRandRange, \xintiiRandRangeAtoB. For some details regarding how xint uses the engine provided generator of pseudo-random numbers, see \xintUniformDeviate.

3.4.5. Functions with two arguments

The atan2() and related trigonometric functions are documented in the calxtrig chapter.

iquo(m, n) Only available in \xintiiexpr/\xintiieval context. Computes the Euclidean quotient. Matches with the remainder defined in next item. See \xintiiQuo.

irem(m, n) Only available in \xintiiexpr/\xintiieval context. Computes the Euclidean remainder. Attention that, following mathematical definition, it is always non-negative. See \xintiiRem.

mod(f, g) computes f - g*floor(f/g). Hence its output is a general fraction or floating point number or integer depending on the used parser. If non-zero, it has the same sign as g.

Prior to 1.2p it computed f - g*trunc(f/g).

The /: and 'mod' infix operators are both mapped to the same underlying macro as this mod(f, g) function. At 1.3 this macro produces smaller denominators when handling fractions than formerly.

```
\xinttheexpr mod(11/7,1/13), reduce(((11/7)//(1/13))*1/13+mod(11/7,1/13)),
mod(11/7,1/13)- (11/7)/:(1/13), (11/7)//(1/13)\relax\newline
\xintthefloatexpr mod(11/7,1/13)\relax\par
```

```
3/91, 11/7, 0, 20
0.03296703296703260
```

Attention: the precedence rules mean that `29/5 /: 3/5` is handled like `((29/5)/:3)/5`. This is coherent with behaviour of Python language for example:

```
>>> 29/5 % 3/5, 11/3 % 17/19, 11/57
(0.5599999999999999, 0.19298245614035087, 0.19298245614035087)
>>> (29/5) % (3/5), (11/3) % (17/19), 5/57
(0.4, 0.08771929824561386, 0.08771929824561403)
```

For comparison (observe on the last lines how `\xintfloatexpr` is more accurate than Python!):

```
\noindent\xinttheexpr 29/5 /: 3/5, 11/3 /: 17/19\relax\newline
         \xinttheexpr (29/5) /: (3/5), (11/3) /: (17/19)\relax\newline
         \xintthefloatexpr 29/5 /: 3/5, 11/3 /: 17/19, 11/57\relax\newline
         \xintthefloatexpr (29/5) /: (3/5), (11/3) /: (17/19), 5/57\relax\newline
         5/57 = \xinttheexpr trunc(5/57, 20)\relax\dots\newline
```

```
14/25, 11/57
2/5, 5/57
0.56, 0.1929824561403509, 0.1929824561403509
0.4, 0.08771929824561420, 0.08771929824561404
5/57 = 0.08771929824561403508...
```

Regarding some details of behaviour in `\xintfloatexpr`, see discussion of `divmod` function next.

divmod(f, g) computes the two mathematical values `floor(f/g)` and `mod(f,g)=f - g*floor(f/g)` and produces them as a bracketed pair in other terms it is analogous to the Python `divmod` function. Its output is equivalent to using `f//g, f/:g` but its implementation avoids doing twice the needed division.

In `\xintfloatexpr...\relax` the modulo is rounded to the prevailing precision. The quotient is like in the other parsers an exact integer. It will be rounded as soon as it is used in further operations, or via the global output routine of `\xintfloatexpr`. *Those examples behave as in 1.3f because assignments to multiple variables tacitly unpack if this is necessary.*

```
\xintdefvar Q, R := divmod(3.7, 1.2);%
\xinttheexpr Q, R, 1.2Q + R\relax\newline
\xintdefiivar Q, R := divmod(100, 17);%
\xinttheiiexpr Q, R, 17Q + R\relax\newline
\xintdeffloatvar Q, R := divmod(100, 17e-20);%
\xintthefloatexpr Q, R, 17e-20 * Q + R\relax\newline
% show Q exactly, although defined as float it can be used in iiexpr:
\xinttheiiexpr Q\relax\ (we see it has more than 16 digits)\par
\xintunassignvar{Q}\xintunassignvar{R}%
```

```
3, 0.1, 3.7
5, 15, 100
5.882352941176471e20, 9e-20, 100
F+21.0.147100000!152941176!100058823! (we see it has more than 16 digits)
```

Again: `f//g` or the first item output by `divmod(f, g)` is an integer `q` which when computed inside `\xintfloatexpr..\relax` is not yet rounded to the prevailing float precision; the second item `f-q*g` is the rounding to float precision of the exact mathematical value evaluated with this

exact q. *This behaviour may change in future major release; perhaps q will be rounded and f-q*g will correspond to usage of this rounded q.*

As \xintfloatexpr rounds its global result, or rounds operands at each arithmetic operation, it requires special circumstances to show that the q is produced unrounded. Either as in the above example or this one with comparison operators:

```
  \xintDigits := 4\relax
  \xintthefloatexpr if(12345678//23==537000, 1, 0), 12345678//23\relax\newline
  \xintthefloatexpr if(float(12345678//23)==537000, 1, 0)\relax\par
  \xintDigits := 16\relax
0, 537000
1
```

In the first line, the comparison is done with floor(12350000/23)=536957 (notice in passing that 12345678//23 was evaluated as 12350000//23 because the operands are first rounded to 4 digits of floating point precision), hence the conditional takes the "False" branch. In the second line the float forces rounding of the output to 4 digits, and the conditional takes the "True" branch.

This example shows also that comparison operators in \xintfloatexpr..\relax act on unrounded operands.

binomial(x, y) computes binomial coefficients. It returns zero if y<0 or x<y and raises an error if x<0 (or if x>99999999.)

```
  \xinttheexpr seq(binomial(20, i), i=0..20)\relax
1, 20, 190, 1140, 4845, 15504, 38760, 77520, 125970, 167960, 184756, 167960, 125970, 77520,
38760, 15504, 4845, 1140, 190, 20, 1

  \printnumber{\xintthefloatexpr seq(binomial(100, 50+i), i=-5..+5)\relax}%
6.144847121413618e28, 7.347099819081500e28, 8.441348728306404e28, 9.320655887504988e28, 9
↪ .891308288780803e28, 1.008913445455642e29, 9.891308288780803e28, 9.320655887504988e28, 8.
↪ 441348728306404e28, 7.347099819081500e28, 6.144847121413618e28
```

The arguments must be (expand to) short integers.

pfactorial(a, b) computes partial factorials i.e. pfactorial(a,b) evaluates the product (a+1)...
↪ b.

```
  \xinttheexpr seq(pfactorial(20, i), i=20..30)\relax
1, 21, 462, 10626, 255024, 6375600, 165765600, 4475671200, 125318793600, 3634245014400,
109027350432000
```

The arguments must (expand to) short integers. See subsection 12.37 for the behaviour if the arguments are negative.

ndfillraw(TEX-macro, n-uple) The second argument is [N1, N2, ..., Nk]. The construct fills an N1x
↪ N2x...xNk hyperrectangular nested list by evaluating the given macro as many times as needed. The expansion result goes directly into internal data and must thus comply with what is expected internally for an individual numeric leaf (at 1.4, xintfrac raw format worked for \xintexpr or \xintfloatexpr, but not \xintiiexpr, and this may have changed since). This is an experimental function serving to generate either constant or random arrays. Attention that **TEX-macro** stands here for any expandable TEX *macro*, and an \xintexpr-ession at this location thus requires an explicit \xinteval wrapping.

DO NOT USE!

3.4.6. Functions with 3 or 4 arguments

if(cond,yes,no) (twofold-way conditional)

checks if cond is true or false and takes the corresponding branch. Any non zero number or fraction is logical true. The zero value is logical false. Both ``branches'' are evaluated (they are not really branches but just numbers). See also the ? operator.

ifint(x,yes,no) (twofold-way conditional)

checks if x is an integer and in that case chooses the ``yes'' branch.

See also isint().

ifone(x,yes,no) (twofold-way conditional)

checks if x is equal to one and in that case chooses the ``yes'' branch.

Slightly more efficient than if(x==1,..,..). See also isone().

ifsgn(cond,<0,=0,>0) (threefold-way conditional)

checks the sign of cond and proceeds correspondingly. All three are evaluated. See also the ?? operator.

3.4.7. Functions with an arbitrary number of arguments

The functions all(), any(), xor(), `+`(), `*`(), max(), min(), gcd(), lcm(), first(), last(), reversed() and len() work both with "open" and "packed" lists (aka nutples).

Since 1.4, when used with a single argument which is a nutple, it is automatically unpacked. But from 1.4 to 1.4h these functions could not be used with a single numeric argument: either they had at least two arguments, or only one and it had to be a nutple. At 1.4i it is again possible to use them with a lone numeric argument.

In the specific case of reversed() with a nutple argument the output is then repacked so that the output is a nutple if and only if the input was one (the reversal does not propagate to deeper nested nutple's, it applies only at depth one).

DO NOT USE!

qraw(stuff) It injects directly tokens to represent internally numerical data. Will break at any release modifying the internal data format specifications (which are not always documented).

all(x, y, ...) inserts a logical AND in-between its arguments and evaluates the resulting logical assertion (as with all functions, all arguments are evaluated).

```
  \xinteval{all(1,1,1), all([1,0,1]), all([1,1,1])}
1, 0, 1
```

any(x, y, ...) inserts a logical OR in-between its arguments and evaluates the resulting logical assertion,

```
  \xinteval{any(0,0,0), any([1,0,1]), any([0,0,0])}
0, 1, 0
```

xor(x, y, ...) inserts a logical XOR in-between its arguments and evaluates the resulting logical assertion,

```
  \xinteval{xor(1,1,1), xor([1,0,1]), xor([1,1,1])}
1, 0, 1
```

`+`(x, y, ...) adds (left ticks mandatory):

```
  \xinttheexpr `+`(1,3,19), `+`(1**2,3**2,sqr(19)), `+`([1**2,3**2,sqr(19)])\relax
23, 371, 371
```

`*`(x, y, ...) multiplies (left ticks mandatory):

```
  \xinttheexpr `*`(1,3,19), `*`(1^2,3^2,19^2), `*`([1^2,3^2,19^2])\relax
57, 3249, 3249
```

max(x, y, ...) maximum of the (arbitrarily many) arguments,

```
  \xinttheexpr max(1,3,19), min([1,3,19])\relax
19, 1
```

min(x, y, ...) minimum of the (arbitrarily many) arguments,

```
  \xinttheexpr min(1,3,19), min([1,3,19])\relax
1, 1
```

gcd(x, y, ...) computes the positive generator of the fractional ideal of rational numbers $x\mathbb{Z}+y\mathbb{Z}+\ldots\subset\mathbb{Q}$. Since 1.4d the output is always in lowest terms.

This example shows how to reduce an n-uple to its primitive part:

```
  \xinteval{gcd(7/300, 11/150, 13/60)}\newline
  $(7/300, 11/150, 13/60)\to
  (\xinteval{subsn(seq(reduce(x/D), x = L), D=gcd(L); L=7/300, 11/150, 13/60)})$\newline
  \xintexpr gcd([7/300, 11/150, 13/60])\relax\par
1/300
(7/300, 11/150, 13/60) → (7, 22, 65)
1/300
```

MEMO Perhaps a future release will provide a primpart() function as built-in functionality.

In case of strict integers, using a \xintiiexpr...\relax wrapper is advantageous as the integer-only gcd() is more efficient. As \xintiiexpr accepts only strict integers, doing this may require wrapping the argument in num().

lcm(x, y, ...) computes the positive generator of the fractional ideal of rational numbers $x\mathbb{Z}\cap y\mathbb{Z}\cap\ldots\subset\mathbb{Q}$.

```
  \xinttheexpr lcm([7/300, 11/150, 13/60])\relax
1001/30
```

As for gcd(), since 1.4d the output is always in lowest terms. For strict integers it is slightly advantageous to use a sub \xintiiexpr-ession.

first(x, y, ...) first item of the list or nutple argument:

```
  \xintiiexpr first([last(-7..3), [58, 97..105]])\relax
3
```

last(x, y, ...) last item of the list or nutple argument:

```
  \xintiiexpr last([-7..3, 58, first(97..105)])\relax
97
```

reversed(x, y, ...) reverses the order of the comma separated list or inside a nutple:

```
  \xintiieval{reversed(reversed(1..5), reversed([1..5]))}
[5, 4, 3, 2, 1], 1, 2, 3, 4, 5
```

The above is correct as xintexpr functions may produce oples and this is the case here.

len(x, y, ...) computes the number of items in a comma separated list or inside a nutple (at first level only: it is not a counter of leaves).

```
  \xinttheiiexpr len(37.5), len(1..50, [101..150], 1001..1050), len([1..10])\relax
1, 101, 10
```

zip(*nutples) behaves similarly to the Python function of the same name: i.e. it produces *an ople of nutples, where the i-th nutple contains the i-th element from each of the argument nutples. The ople ends when the shortest input nutple is exhausted. With a single nutple argument, it returns an ople of 1-nutples. With no arguments, it returns the empty ople.*

As there is no exact match in xintexpr of the concept of "iterator" object,[14] there is a significant difference here that (for example) the zip(x,x,x) Python idiom to cluster the iterator x into successive chunks of length 3 does not apply. Consider for this reason even the name of the function as work-in-progress, susceptible to change.

Unstable?

```
\xintiieval{zip([1..9], [0, 1, 2], [11..29], [111..139])}
[1, 0, 11, 111], [2, 1, 12, 112], [3, 2, 13, 113]
```

See also \xintthespaceseparated for some possible usage in combination with flat().

3.4.8. Functions requiring dummy variables

The pseudo-functions subs(), seq(), subsm(), subsn(), iter(), add(), mul(), rseq(), iterr(), rrseq(), iterr(), ndseq(), ndmap(), ndfillraw() use delimited macros for some tasks:

- for all of them, whenever a <varname>= chunk must be parsed into a (non-assigned) variable name, then the equal sign must be visible,
- and if the syntax is with ,<varname>= the initial comma also must be visible (spaces do not matter),
- for all of them but ndmap() and ndfillraw() the final closing parenthesis must be visible.

Although delimited macros involving commas are used to locate ,<varname= this is done in a way silently ignoring commas located inside correctly balanced parentheses. Thus, as the examples will show, nesting works as expected.

The semi-colons involved in the syntax may arise from expansion alone. For rseq(), iter(), rrseq() and iterr() the ,<varname>= part may also be created from the expansion which will generate the initial comma separated values delimited by a semi-colon.

Prior to 1.4, semi-colons needed to be braced or otherwise hidden when located in an expression parsed by \xintdefvar or \xintdeffunc, to not be confused with the expression terminator.

seq(), rseq(), iter(), rrseq(), iterr() and also add(), mul(), but not subs() admit the omit, abort, and break() keywords. This is a new feature at 1.4 for add() and mul().

In the case of a potentially infinite list generated by the <integer>++ syntax, use of abort or of break() is mandatory, naturally.

All lowercase and uppercase Latin letters are pre-configured for usage as dummy variables. In Unicode engines one can use \xintnewdummy to turn any letter into a usable dummy variable.

And since 1.4, \xintnewdummy works (in all engines) to turn a multi-letter word into a dummy variable. In the descriptions, varname stands for such a dummy variable, either single-letter or word.

subs(expr, varname=comma separated values) for variable substitution.

```
\xinttheexpr subs(subs(seq(x*z,x=1..10),z=y^2),y=10)\relax\newline
100, 200, 300, 400, 500, 600, 700, 800, 900, 1000
```

Attention that xz generates an error, one must use explicitely x*z, else the parser expects a variable with name xz.

subs() is useful when defining macros for which some argument will be used more than once but may itself be a complicated expression or macro, and should be evaluated only once, for matters

[14] Speaking of iterators, I have some ideas about this: as \xintexpr does not have the global expression in its hands it is difficult to organize globally expandably the idea of iterator, but locally via syntax like the one for seq() this is feasible. When one thinks about it, seq() is closely related to the iterator idea.

of efficiency. But subs() is helpless in function definitions: all places where a variable is substituted will receive the complete recipe to compute the variable, rather than evaluate only once.

One should rather define auxiliary functions to compute intermediate results. Or one can use seq(). See the documentation of \xintdeffunc.

add(expr, varname=comma separated values) addition

```
  \xintiiexpr add(x^3,x=1..20), add(x(x+1), x=1,3,19)\relax\newline
  \xintiiexpr add(x^3, x = 1..[2]..20)\relax\newline         % add only odd cubes
  \xintiiexpr add((odd(x))?{x^3}{omit}, x = 1..20)\relax\par % add only odd cubes
44100, 394
19900
19900
```

At 1.4 (fixed at 1.4a), the keywords omit (as in example above), abort and break() are allowed. The meaning of break() is specific: its argument serves as last operand for the addition, not as ultimate value.

```
  \xintiiexpr add((x>10)?{break(1000)}{x}, x = 1..15)\relax
1055
```

The @ special variable holds the so-far accumulated value. Initially its value is zero.

```
  \xintiiexpr add(1 + @, i=1..10)\relax % iterates x <- 2x+1
1023
```

See `+`() for syntax simply adding items of a list without usage of a dummy variable.

mul(expr, varname=comma separated values) multiplication

```
  \xintiiexpr mul(x^2, x = 1, 3, 19, 37..50)\relax
217184665384874110852122798021721110872064000000000
```

The @ special variable holds the so-far accumulated value. Initially its value is one.

At 1.4 (fixed at 1.4a), the keywords omit, abort and break() are allowed. The meaning of break() is specific: its argument serves as last operand for the multiplication, not as ultimate value.

```
  \xintiieval{mul((i==100)?{break(i^4)}{i}, i = 98, 99, 100)}
970200000000
```

See `*`() for syntax without a dummy variable.

seq(expr, varname=comma separated valuesvalues) comma separated values generated according to a formula

```
  \xintiiexpr seq(x(x+1)(x+2)(x+3),x=1..10), `*`(seq(3x+2,x=1..10))\relax
24, 120, 360, 840, 1680, 3024, 5040, 7920, 11880, 17160, 1162274713600
```

```
  \smallskip
  \leavevmode\vbox{\xintthealign\xintiiexpr [seq([seq(i^2+j^2, i=0..j)], j=0..10)]\relax}
```

```
[[  0     ],
 [  1,    2     ],
 [  4,    5,    8     ],
 [  9,    10,   13,   18    ],
 [  16,   17,   20,   25,   32    ],
 [  25,   26,   29,   34,   41,   50    ],
 [  36,   37,   40,   45,   52,   61,   72    ],
 [  49,   50,   53,   58,   65,   74,   85,   98    ],
 [  64,   65,   68,   73,   80,   89,   100,  113,  128   ],
 [  81,   82,   85,   90,   97,   106,  117,  130,  145,  162   ],
 [  100,  101,  104,  109,  116,  125,  136,  149,  164,  181,  200  ]]
```

rseq(initial value; expr, varname=comma separated values) recursive sequence, `@` for the previous value.

```
  \printnumber {\xintthefloatexpr subs(rseq (1; @/2+y/2@, i=1..10),y=1000)\relax }\newline
1, 500.5, 251.2490009990010, 127.6145581634591, 67.72532736082604, 41.24542607499115, 32.7
↪ 4526934448864, 31.64201586865079, 31.62278245070105, 31.62277660168434, 31.62277660168379
```



Attention: in the example above `y/2@` is interpreted as `y/(2*@)`. With versions 1.2c or earlier it would have been interpreted as `(y/2)*@`.

In case the initial stretch is a comma separated list, `@` refers at the first iteration to the whole list. Use parentheses at each iteration to maintain this ``nuple''. For example:

```
  \printnumber{\xintthefloatexpr rseq(1,10^6;
               (sqrt(@[0]*@[1]),(@[0]+@[1])/2), i=1..7)\relax }
1, 1e6, 1000, 500000.5, 22360.69095533499, 250500.25, 74842.22521066670, 136430.4704776675,
101048.3052657827, 105636.3478441671, 103316.8617608946, 103342.3265549749, 103329.593373
↪ 4841, 103329.5941579348, 103329.5937657094, 103329.5937657095
```

Prior to 1.4 the above example had to be written with `[@]`. This is still possible (`@` stands for an ople with two items, bracketing then extracting is like extracting directly), but it is leaner to drop the extra "packing".

iter(initial value; expr, varname=comma separated values) is exactly like `rseq`, except that it only prints the last iteration.

`iter()` is convenient to handle compactly higher order iterations. We can illustrate its use with an expandable (!) implementation of the Brent-Salamin algorithm for the computation of π:

```
\xintDigits:= 87\relax % we target 84 digits, and use 3 guard digits
\xintdeffloatfunc BS(a, b, t, p):= 0.5*(a+b), sqrt(a*b), t-p*sqr(a-b),
                                                  \xintiiexpr 2p\relax;
\xinteval
{trunc(% I feel truncation is better than rounding to display decimals of π
  \xintfloatexpr
    iter(1, sqrt(0.5), 1, 1; % initial values
% this 43 is 84/2 + 1
      (@[0]-@[1]<2e-43)?% stopping criteria; takes into account that the
                        % exit computation (break() argument) doubles
                        % number of exact digits (roughly)
      {break(sqr(@[0]+@[1])/@[2])}      % ... do final computation,
      {BS(@)}, % else do iteration
       i=1++)  % This generates infinite iteration. The i is not used.
  \relax
% this 83 is 84 - 1 (there is a digit known to be 3 actually, before decimal mark)
```

```
  , 83)% closing parenthesis of trunc()
 }...% some dots following end of \xinteval argument
 \xintDigits:=16\relax
3.14159265358979323846264338327950288419716939937510582097494459230781640628620899862...
```

You can try with `\xintDigits:=1004\relax` and `2e-501` in place of `\xintDigits:=87\relax` and `2e-43`, but be patient for some seconds for the result. Of course don't truncate the final result to only `83` fractional decimal digits but `1000...` and better to wrap the whole thing in `\message` or `\immediate\write128` or `\edef` because it will then run in the right margin.

Prior to `1.4` the above example had to use notation such as `[@][0]`; this would still work but `@[0]` is leaner.

rrseq(initial values; expr, varname=comma separated values) recursive sequence with multiple initial terms. Say, there are `K` of them. Then `@1`, ..., `@4` and then `@@(n)` up to `n=K` refer to the last `K` values. Notice the difference with `rseq()` for which `@` refers to a list of items in case the initial value is a list and not a single item.[15] Using `rrseq()` with `@1` etc... accessors may be perhaps a bit more efficient than using `rseq()` with a list as staring value and constructs such as `@[0]`, `@[1]` (or rather `@[-1]`, `@[-2]` to mimick what `@1`, `@2`, `@3`, `@4` and `@@(integer)` do in `rrseq()`.

```
 \xinttheiiexpr rrseq(0,1; @1+@2, i=2..30)\relax
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765, 10946,
17711, 28657, 46368, 75025, 121393, 196418, 317811, 514229, 832040
```

```
 \xinttheiiexpr rseq(1; 2@, i=1..10)\relax
1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024
```

```
 \xinttheiiexpr rseq(1; 2@+1, i=1..10)\relax
1, 3, 7, 15, 31, 63, 127, 255, 511, 1023, 2047
```

```
 \xinttheiiexpr rseq(2; @(@+1)/2, i=1..5)\relax
2, 3, 6, 21, 231, 26796
```

```
 \xinttheiiexpr rrseq(0,1,2,3,4,5; @1+@2+@3+@4+@@(5)+@@(6), i=1..20)\relax
0, 1, 2, 3, 4, 5, 15, 30, 59, 116, 229, 454, 903, 1791, 3552, 7045, 13974, 27719, 54984, 109065,
216339, 429126, 851207, 1688440, 3349161, 6643338
```

I implemented an `Rseq` which at all times keeps the memory of *all* previous items, but decided to drop it as the package was becoming big.

iterr(initial values; expr, varname=comma separated values) same as `rrseq` but does not print any value until the last `K`.

```
 \xinttheiiexpr iterr(0,1; @1+@2, i=2..5, 6..10)\relax
 % the iterated over list is allowed to have disjoint defining parts.
55
```

subsm(expr, var1=value1; var2=value2; ....; varN=valueN[;]) Simultaneous substitutions. The assigned values must not involve the variables. An optional final semi-colon is allowed.

```
 \xintiieval{subsm(x+2y+3z+4t, x=1; y=10; z=100; t=1000;)}
4321
```

subsn(expr, var1=value1; var2=value2; ....; varN=valueN[;]) Simultaneous substitutions. The assigned values may involve all variables located further to its right. An optional final semi-colon is allowed.

[15] Prior to `1.4`, one could use `@` in `rrseq()` and `iterr()` as an alias to `@1`. This undocumented feature is dropped and `@` will break `rrseq()` and `iterr()`.

```
\xintiieval{subsn(x+y+z+t, x=20y; y=20z; z=20t; t=1)}
```
```
8421
```

ndmap(function, values1; values2; ....; valuesN[;]) Construction of a nested list (a priori having N dimensions) from function values. The function must be an N-variable function (or a function accepting arbitrarily many arguments), but it is not constrained to produce only scalar values. Only in the latter case is the output really an N-dimensional "ndlist" type object. An optional final semi-colon in the input before the closing parenthesis is allowed.

```
\xintdeffunc foo(a,b,c,d) = a+b+c+d;
\begin{multicols}{2}
\xintthealign\xintexpr ndmap(foo, 1000,2000,3000; 100,200,300; 10,20,30; 1,2,3)\relax
\end{multicols}
```

```
[[[[  1111,  1112,  1113  ],                    [  2231,  2232,  2233  ]],
   [  1121,  1122,  1123  ],                   [[  2311,  2312,  2313  ],
   [  1131,  1132,  1133  ]],                   [  2321,  2322,  2323  ],
  [[  1211,  1212,  1213  ],                    [  2331,  2332,  2333  ]]],
   [  1221,  1222,  1223  ],                  [[[  3111,  3112,  3113  ],
   [  1231,  1232,  1233  ]],                   [  3121,  3122,  3123  ],
  [[  1311,  1312,  1313  ],                    [  3131,  3132,  3133  ]],
   [  1321,  1322,  1323  ],                   [[  3211,  3212,  3213  ],
   [  1331,  1332,  1333  ]]],                  [  3221,  3222,  3223  ],
 [[[  2111,  2112,  2113  ],                    [  3231,  3232,  3233  ]],
   [  2121,  2122,  2123  ],                   [[  3311,  3312,  3313  ],
   [  2131,  2132,  2133  ]],                   [  3321,  3322,  3323  ],
  [[  2211,  2212,  2213  ],                    [  3331,  3332,  3333  ]]]]
   [  2221,  2222,  2223  ],
```

ndseq(expr, var1=values1; var2=values2; ....; varN = valuesN[;]) Constructs a nested list (a priori having N dimensions) from substitutions in an expression involving N (dummy) variables. The expression is not constrained to produce only scalar values. Only in the latter case is the output really an N-dimensional "ndlist" type object. An optional final semi-colon in the input before the closing parenthesis is allowed.

```
\begin{multicols}{2}
\xintthealign\xintexpr ndseq(a+b+c+d, a=1000,2000,3000; b=100,200,300;
                                                    c=10,20,30; d=1,2,3;)\relax
\end{multicols}% in case of page break, this makes amusing zigzag rendering
```

```
[[[[  1111,  1112,  1113  ],                    [  2231,  2232,  2233  ]],
   [  1121,  1122,  1123  ],                   [[  2311,  2312,  2313  ],
   [  1131,  1132,  1133  ]],                   [  2321,  2322,  2323  ],
  [[  1211,  1212,  1213  ],                    [  2331,  2332,  2333  ]]],
   [  1221,  1222,  1223  ],                  [[[  3111,  3112,  3113  ],
   [  1231,  1232,  1233  ]],                   [  3121,  3122,  3123  ],
  [[  1311,  1312,  1313  ],                    [  3131,  3132,  3133  ]],
   [  1321,  1322,  1323  ],                   [[  3211,  3212,  3213  ],
   [  1331,  1332,  1333  ]]],                  [  3221,  3222,  3223  ],
 [[[  2111,  2112,  2113  ],                    [  3231,  3232,  3233  ]],
   [  2121,  2122,  2123  ],                   [[  3311,  3312,  3313  ],
   [  2131,  2132,  2133  ]],                   [  3321,  3322,  3323  ],
  [[  2211,  2212,  2213  ],                    [  3331,  3332,  3333  ]]]]
   [  2221,  2222,  2223  ],
```

Recursions may be nested, with `@@@(n)` giving access to the values of the outer recursion... and there is even `@@@@(n)` to access the outer outer recursion but I never tried it!

The following keywords are recognized:

abort it is a pseudo-variable which indicates to stop here and now.

omit it is a pseudo-variable which says to omit this value and go to next one.

break(stuff) says to abort and insert `stuff` as last value.

<integer>++ serves to generate a potentially infinite list. In conjunction with an `abort` or `break()` this is often more efficient than iterating over a pre-established list of values.

```
\xinttheiiexpr iter(1;(@>10^40)?{break(@)}{2@},i=1++)\relax
```

10889035741470030830827987437816582766592 is the smallest power of 2 with at least fourty one digits.

The `i=<integer>++` syntax (any letter is allowed in place of `i`) works only in the form `<letter>=<integer>++`, something like `x=10,17,30++` is not legal. The `<integer>` must be a TEX-allowable integer.

```
First Fibonacci number at least |2^31| and its index
% we use iterr to refer via @1 and @2 to the previous and previous to previous.
\xinttheiiexpr iterr(0,1; (@1>=2^31)?{break(@1, i)}{@2+@1}, i=1++)\relax
```

First Fibonacci number at least 2^31 and its index 2971215073, 47. If one also wants the previous Fibonacci number one only has to use `break(@2, @1, i)` in the above example.

3.5. Generators of arithmetic progressions

- `a..b` constructs the **small** integers from the ceil ⌈a⌉ to the floor ⌊b⌋ (possibly a decreasing sequence): one has to be careful if using this for algorithms that `1..0` for example is not empty or `1` but expands to `1, 0`. Again, `a..b` *can not* be used with `a` and `b` greater than $2^{31}-1$. Also, only about at most `5000` integers can be generated (this depends upon some TEX memory settings).

  The `..` has lower precedence than the arithmetic operations.

  ```
  \xintexpr 1.5+0.4..2.3+1.1\relax; \xintexpr 1.9..3.4\relax; \xintexpr 2..3\relax
  ```

  2, 3; 2, 3; 2, 3

  The step of replacing a by its ceil and b by its floor is a kind of silly overhead, but a and b are allowed to be themselves the result of computations and there is no notion of "int" type in `\xinteval`. The solution is, when a and b are given explicit integers to temporarily switch to the `\xintiiexpr` parser:

  ```
  \xintexpr \xintiiexpr 1..10\relax\relax
  ```

  1, 2, 3, 4, 5, 6, 7, 8, 9, 10

  On the other hand integers from `\xintexpr 1..10\relax` are already in raw `xintfrac` format for example `3/1[0]` which speeds up their usage in the macros internally involved in computations... thus perhaps what one gains on one side is lost on the other side.

  Since `2.0`, the values generated by `a..b` in the floating point parser are in the internal floating point format, even though they represent integers. If a dummy variable is to be used inside `\xintiieval`, for example for constructing indices, one must now use `i=\xintiiexpr 1..10\relax` rather than `i=1..10`. This applies also in the context of `\xintdefloatfunc` and `\xintdefloatvar`.

- `a..[d]..b` generates "real" numbers along arithmetic progression of reason `d`. It does *not* replace `a` by its ceil, nor `b` by its floor. The generated list is empty if `b-a` and `d` are of opposite signs; if `d=0` or if `a=b` the list expands to single element `a`.

```
\xintexpr 1.5..[1.01]..11.23\relax
```
```
1.5, 2.51, 3.52, 4.53, 5.54, 6.55, 7.56, 8.57, 9.58, 10.59
```

At 1.4, this generator behaves in \xintfloatexpr exactly as in \xintexpr, i.e. *exactly*. This is breaking change.

```
\xintDigits:=6;
\xintexpr\xintfloatexpr 100..[1.23456]..110\relax\relax
\xintDigits:=16;
```
```
100, 101.23456, 102.46912, 103.70368, 104.93824, 106.1728, 107.40736, 108.64192, 109.87648
```

This demonstration embedded the float expression in the exact parser only to avoid the rounding to the prevailing precision on output, thus we can see that internally additions are done exactly and not with 6 digits mantissas (in this example).

3.6. Python slicing and indexing of one-dimensional sequences

We denote here by *list* or *sequence* a general *ople*, either given as a variable or explicitly. *In the former case the parentheses are optional.*[16]

- (list)[n] returns the n+1th item if n>=0. If n<0 it enumerates items from the tail. Items are numbered as in Python, the first element corresponding to n=0.

  ```
  \xintexpr (0..10)[6], (0..10)[-1], (0..10)[23*18-22*19]\relax
  ```
  ```
  6, 10, 7
  ```

  This also works for singleton *oples* which are in fact a *number*:

  ```
  \xintexpr (7)[0], (7)[-1], 9, (7)[-2], 9\relax
  ```
  ```
  7, 7, 9, 9
  ```

  In the example above the parentheses serve to disambiguate from the raw xintfrac format such as 7[-1] which, although discouraged, is accepted on input. And we used a trick to show that (7)[-2] returns nil.

  The behaviour changes for singleton *oples* which are not *numbers*. They are thus *nutples*, or equivalently they are the bracketing (bracing, packing) of another *ople*. In this case, the meaning of the syntax for item indexing is, as in Python, item *extraction*:

  ```
  \xintexpr [0,1,2,3,4,5][2], [0,1,2,3,4,5][-3]\relax\newline
  \xintexpr [0,[1,2,3,4,5],6][1][-1]\relax
  ```
  ```
  2, 3
  5
  ```

- (list)[:n] produces the first n elements if n>0, or suppresses the last |n| elements if n<0.

  ```
  \xintiiexpr (0..10)[:6]\relax\ and \xintiiexpr (0..10)[:-6]\relax
  ```
  ```
  0, 1, 2, 3, 4, 5 and 0, 1, 2, 3, 4
  ```

  As above, the meaning change for *nutples* and fits with expectations from Python regarding its sequence types:

  ```
  \xintiiexpr [0..10][:6]\relax\ and \xintiiexpr [0..10][:-6]\relax
  ```
  ```
  [0, 1, 2, 3, 4, 5] and [0, 1, 2, 3, 4]
  ```

- (list)[n:] suppresses the first n elements if n>0, or extracts the last |n| elements if n<0.

  ```
  \xintiiexpr (0..10)[6:]\relax\ and \xintiiexpr (0..10)[-6:]\relax
  ```

[16] Even for an "open list", if it is given as a *variable* then the indexing or slicing will not apply to its last item but to itself as an entity.

```
6, 7, 8, 9, 10 and 5, 6, 7, 8, 9, 10
```

As above, the meaning change for *nutples* and fit with expectations from Python with *tuple* or *list* types:

```
  \xintiiexpr [0..10][6:]\relax\ and \xintiiexpr [0..10][-6:]\relax
[6, 7, 8, 9, 10] and [5, 6, 7, 8, 9, 10]
```

- Finally, (list)[a:b] also works according to the Python ``slicing'' rules (inclusive of negative indices). Notice though that stepping is currently not supported.

```
  \xinttheiiexpr (1..20)[6:13]\relax\ = \xinttheiiexpr (1..20)[6-20:13-20]\relax
  \newline
  \xinttheiiexpr [1..20][6:13]\relax\ = \xinttheiiexpr [1..20][6-20:13-20]\relax
7, 8, 9, 10, 11, 12, 13 = 7, 8, 9, 10, 11, 12, 13
[7, 8, 9, 10, 11, 12, 13] = [7, 8, 9, 10, 11, 12, 13]
```

- It is naturally possible to execute such slicing operations one after the other (the syntax is simplified compared to before 1.4):

```
  \xintexpr (1..50)[13:37][10:-10]\relax\newline
  \xintexpr (1..50)[13:37][10:-10][-1]\relax
24, 25, 26, 27
27
```

3.7. NumPy like nested slicing and indexing for arbitrary oples and nutples

I will give one illustrative example and refer to the NumPy documentation for more.

Notice though that our interpretation of the syntax is more general than NumPy's concepts (of basic slicing/indexing):

- slicing and itemizing apply also to non-bracketed objects i.e. *oples*,
- the leaves do not have to be all at the same depth,
- there are never any out-of-range index errors: out-of-range indices are silently ignored.

```
\begin{multicols}{3}
\xintdefvar myArray = ndseq(a+b+c, a=100,200,300; b=40,50,60; c=7,8,9);
myArray = \xintthealign\xintexpr myArray\relax
\columnbreak
mySubArray = \xintthealign\xintexpr myArray[0:2,0:2,0:2]\relax
myExtractedSubArray = \xintthealign\xintexpr myArray[0:2,0:2,0:2][0]\relax
\columnbreak
myExtractedSubArray = \xintthealign\xintexpr myArray[0:2,0:2,0:2][0,1]\relax
\noindent
firstExtractedScalar = \xintexpr myArray[0:2,0:2,0:2][0,1,0]\relax\newline
secondExtractedScalar = \xintexpr myArray[0,1,0]\relax\par
\end{multicols}
```

```
  myArray =                    [  267,  268,  269  ]],          mySubArray =
[[[  147,  148,  149  ],         [[  347,  348,  349  ],         [[[  147,  148  ],
  [  157,  158,  159  ],          [  357,  358,  359  ],           [  157,  158  ]],
  [  167,  168,  169  ]],         [  367,  368,  369  ]]]         [[  247,  248  ],
 [[  247,  248,  249  ],                                           [  257,  258  ]]]
  [  257,  258,  259  ],                                          myExtractedSubArray =
```

```
[[  147,  148  ],                myExtractedSubArray =        firstExtractedScalar = 157
 [  157,  158  ]]               [  157,  158  ]              secondExtractedScalar = 157
```

As said before, *stepping* is not yet implemented. Also the NumPy extension to Python for item selection (i.e. via a tuple of comma separated indices) is not yet implemented.

3.8. Tacit multiplication

Tacit multiplication (insertion of a *) applies when the parser is currently either scanning the digits of a number (or its decimal part or scientific part, or hexadecimal input), or is looking for an infix operator, and:

(1.) *encounters a count or dimen or skip register or variable or an ε-TEX expression*, or
(2.) *encounters a sub-\xintexpression*, or
(3.) *encounters an opening parenthesis*, or
(4.) *encounters a letter (which is interpreted as signaling the start of either a variable or a function name)*, or
(5.) (of course, only when in state "looking for an operator") *encounters a digit*.

!!!!ATTENTION!!!!

Explicit digits prefixing a variable, or a function, whose name starts with an e or E will trap the parser into trying to build a number in scientific notation. So the * must be explicitly inserted.

```
\xintdefiivar e := (2a+4b+6d+N)/:7;%
\xintdefiivar f := (c+11d+22*e)//451;% 22e would raise errors
```

I don't think I will fix this anytime soon...

For example, if x, y, z are variables all three of (x+y)z, x(y+z), (x+y)(x+z) will create a tacit multiplication.

Furthermore starting with release 1.2e, whenever tacit multiplication is applied, in all cases it *always* ``ties'' more than normal multiplication or division, but still less than power. Thus x/2y is interpreted as x/(2y) and similarly for x/2max(3,5) but x^2y is still interpreted as (x^2)*y and 2n! as 2*n!.

```
\xintdefvar x:=30;\xintdefvar y:=5;%
\xinttheexpr (x+y)x, x/2y, x^2y, x!, 2x!, x/2max(x,y)\relax
1050, 30/10, 4500, 265252859812191058636308480000000, 530505719624382117272616960000000,
30/60
```

Since 1.2q tacit multiplication is triggered also in cases such as (1+2)5 or 10!20!30!.

```
\xinttheexpr (10+7)5, 4!4!, add(i, i=1..10)10, max(x, y)100\relax
85, 576, 550, 3000
```

The ``tie more'' rule applies to all cases of tacit multiplication. It impacts only situations with a division operator as the last seen operator, as multiplication is mathematically associative.

```
\xinttheexpr 1/(3)5, (1+2)/(3+4)(5+6), 2/x(10), 2/10x,
                                  3/y\xintiiexpr 5+6\relax, 1/x(y)\relax\
differ from\newline\xinttheexpr 1/3*5, (1+2)/(3+4)*(5+6), 2/x*(10), 2/10*x,
                                 3/y*\xintiiexpr 5+6\relax, 1/x*(y)\relax\par
1/15, 3/77, 2/300, 2/300, 3/55, 1/150 differ from
5/3, 33/7, 20/30, 60/10, 33/5, 5/30
```

Note that y\xinttheiiexpr 5+6\relax would have tried to use a variable with name y11 rather than doing y*11: tacit multiplication works only in front of sub-\xintexpressions, not in front

of \xinttheexpressions which are unlocked into explicit digits.

Here is an expression whose meaning is completely modified by the ``tie more'' property of tacit multiplication:

```
\xintdeffunc e(z):=1+z(1+z/2(1+z/3(1+z/4)));
```

will be parsed as

```
\xintdeffunc e(z):=1+z*(1+z/(2*(1+z/(3*(1+z/4)))));
```

which is not at all the presumably hoped for:

```
\xintdeffunc e(z):=1+z*(1+(z/2)*(1+(z/3)*(1+(z/4))));
```

3.9. User defined variables

Since release 1.1 it is possible to make an assignment to a variable name and let it be known to the parsers of xintexpr. Since 1.2p simultaneous assignments are possible. Since 1.4 simultaneous assignments are possible with a right-hand-side being a nutple which will be automatically unpacked.

```
\xintdefvar myPi:=3.14159265358979323846264 3;%
$myPi = \xinteval{myPi}$\newline % (there is already built-in Pi variable)
\xintdefvar x_1, x_2, x_3 := 10, 20, 30;%
$x_1 = \xinteval{x_1}, x_2 = \xinteval{x_2}, x_3 = \xinteval{x_3}$\newline
\xintdefvar x_1, x_2, x_3 := [100, 200, 300];%
$x_1 = \xinteval{x_1}, x_2 = \xinteval{x_2}, x_3 = \xinteval{x_3}$\par
```

myPi = 3.14159265358979323846264

$x_1 = 10, x_2 = 20, x_3 = 30$

$x_1 = 100, x_2 = 200, x_3 = 300$

Simultaneous assignments with more variables than values do not raise an error but simply set the extra variables to the nil value.

```
\xintdefiivar a, b, c := [1, 2];% will be automatically unpacked
The value of a is \xinteval{a}, the one of b is \xinteval{b} and
the one of c is \xinteval{c}.
```

The value of a is 1, the one of b is 2 and the one of c is .

```
\xintdefiivar a, b, c := 314;%
The value of a is \xinteval{a}, the one of b is \xinteval{b} and
the one of c is \xinteval{c}.
```

The value of a is 314, the one of b is and the one of c is .

Notice that nil variables must be used with caution as they break arithmetic operations if used as operands to them. And they are not the same as the None variables, which can also be input as [].

Simultaneous assignments with less variables than values do not raise an error but set the last variable to be the ople concatenating the remaining values.

```
\xintdefiivar seq := 1..10;%
\xintdefiivar a, seq := seq;%
\xintdefiivar b, seq := seq;%
\xintdefiivar c, d, seq := seq;%
The value of a is \xinteval{a}, the one of b is \xinteval{b}, the one of c is \xinteval{c},
the one of d is \xinteval{d}, the one of seq is \xinteval{seq}.
```

The value of a is 1, the one of b is 2, the one of c is 3, the one of d is 4, the one of seq is 5, 6, 7, 8, 9, 10.

In the above we define a variable seq but there is a built-in function seq(). It is indeed allowed to use the same name for both a variable and a function.[17] But for safety we will unassign seq now:

```
\xintunassignvar{a}\xintunassignvar{b}\xintunassignvar{c}\xintunassignvar{d}%
```

[17] But until a bugfix added at release 1.4i, some built-in function names (those implementing syntax with dummy variables, and the so-called "pseudo"-functions) were fragile under such overloading.

```
  \xintunassignvar{seq}%
```

Single letter names a..z and A..Z are pre-declared by the package for use as a special type of variables called ``dummy variables''. Unnassigning them restores this initial meaning. See further \xintunassignvar and \xintnewdummy. Since 1.4 even assigned variables can be used in the call signatures of function declarations.

Regarding the manipulation of an "open list" as above, there is no way to obtain with only one use of the variable both its last item and the reduction of the variable to its truncated self. One can do rather:

```
  \xintdefiivar mylist := 1..10;%
  \xintdefiivar z, mylist := last(mylist), mylist[:-1];%
  The value of z is \xinteval{z} and mylist is now \xinteval{mylist}.\par
```

The value of z is 10 and mylist is now 1, 2, 3, 4, 5, 6, 7, 8, 9.

This uses twice mylist and is about the same as doing it in two steps:

```
  \xintdefiivar w := last(mylist);%
  \xintdefiivar mylist := mylist[:-1];%
  The value of w is \xinteval{w} and mylist is now \xinteval{mylist}.%
  \xintunassignvar{z}\xintunassignvar{w}\xintunassignvar{mylist}\par
```

The value of w is 9 and mylist is now 1, 2, 3, 4, 5, 6, 7, 8.

It is recommended generally speaking to work with "closed (i.e. bracketed) lists" because only them and numbers can be arguments to functions (but see \xintdeffunc and the notion of variadic last argument). For more on the Python-like slicing used above see subsection 3.6 and subsubsection 3.13.4. For more information relative to variables versus arguments see subsubsection 3.13.6.

- For catcodes issues (particularly, for the semi-colon used to delimit the fetched expression), see the discussion of \xintexprSafeCatcodes and some comments in the section documenting \xintdeffunc.
- Both syntaxes \xintdefvar foo := <expr>; and \xintdefvar foo = <expr>; are accepted.
- Spaces in the variable name or around the equal sign are removed and are immaterial.
- The variable names are expanded in an \edef (and stripped of spaces). Example:

  ```
  \xintdefvar x\xintListWithSep{, x}{\xintSeq{0}{10}} := seq(2**i, i = 0..10);%
  ```

  This defines x0, x1, . . . , x10 for future usage.

Legal variable names are composed of letters, digits, _ and @ and characters. A variable name must start with a letter. Variable names starting with a @ or _ are reserved for internal usage.[18]

As x_1x_2 or even x_1x are licit variable names, and as the parser does not trace back its steps, input syntax must be x_1*x_2 if the aim is to multiply such variables.

Using \xintdefvar, \xintdefiivar, or \xintdeffloatvar means that the variable value will be computed using respectively \xintexpr, \xintiiexpr or \xintfloatexpr. It can then be used in all three parsers, as long as the parser understands the format. Currently this means that variables using \xintdefvar or \xintdeffloatvar can be used freely either with \xintexpr or \xintfloatexpr but not with \xintiiexpr, and variables defined via \xintdefiivar can be used in all parsers.

When defining a variable with \xintdeffloatvar it (or generally speaking its numerical leaves) is rounded to \xinttheDigits precision. So the variable holds the same value as would be printed via \xintfloateval for the same computation.

Prior to 1.4e, this was the case only if the variable definition actually involved some computation.

However the \xintfloatexpr..\relax wrapper by itself induces no rounding. If it is encountered in the typesetting flow, the print-out will be rounded to \xinttheDigits precision, but this is an effet of behaving like \xintfloateval in this context.

[18] The process of variable declaration does not check that these rules are met, and breakage will arise on use, if rules are not followed. For example, prior to 1.4g, using a variable (illegally) declared with a name starting with a (normal, catcode 8) _ triggered an infinite loop.

```
 % Since 1.4e, \xintdeffloatvar always rounds (to \xinttheDigits)
 \xintdeffloatvar e:=2.718281828459045235360287471352662497757247093699959574966967 6;%
 1) \xintexpr  e\relax\newline  % shows the recorded value: it is rounded
 2) \xintfloatexpr   % when used in typesetting flow, acts like \xintfloateval:
     2.718281828459045235360287471352662497757247093699959574966967 6
 \relax\newline   % the print-out is rounded.
 3) \xintexpr
   \xintfloatexpr
     2.718281828459045235360287471352662497757247093699959574966967 6
   \relax
 \relax\newline
 %
 % but we can see via the \xintexpr wrapper all the digits were there rounding
 % can be forced using an extra 0+, the float() function, or the [D] option.
 % tidbit: comparison operators do not pre-round, so 1.2345678 is not same as
 % (1.2345678+0) in low precision.
 %
 \begingroup\xintDigits:=4;%
 4) \xintifboolfloatexpr{1.2345 == 1.23456}
    {\error}{Different! Comparisons do not pre-round to Digits precision.}\newline
 5) \xintifboolfloatexpr{1.2345 == 1.2345 + 0}
    {\error}{Different! Right hand side rounded from operation,
                        left hand side not rounded.}\par
 \endgroup
1) 2.718281828459045
2) 2.718281828459045
3) 2.718281828459045235360287471352662497757247093699959574966967 6
4) Different!  Comparisons do not pre-round to Digits precision.
5) Different!  Right hand side rounded from operation, left hand side not rounded.
```

After issuing \xintverbosetrue the values of defined variables are written out to the log (and terminal). As in this example:

```
 Package xintexpr Info: (on line 1)
     Variable myPi defined with value {3141592653589793238462643[-24]}.
 Package xintexpr Info: (on line 2)
     Variable x_1 defined with value {10}.
 Package xintexpr Info: (on line 2)
     Variable x_2 defined with value {20}.
 Package xintexpr Info: (on line 2)
     Variable x_3 defined with value {30}.
 Package xintexpr Info: (on line 3)
     Variable List defined with value {0}{1}{3}{6}{10}{15}{21}{28}{36}{45}{55}
 .
 Package xintexpr Info: (on line 4)
     Variable Nuple defined with value {{0}{1}{9}{36}{100}{225}{441}{784}{1296
 }{2025}{3025}}.
 Package xintexpr Info: (on line 5)
     Variable FourthPowers defined with value {{0}{1}{81}{1296}{10000}{50625}{
 194481}{614656}{1679616}{4100625}{9150625}}.
```

3.9.1. \xintunassignvar

code

\xintunassignvar{⟨*variable*⟩} will make the ⟨*variable*⟩ un-assigned. For example in the previous

section we used

```
\xintdeffloatvar e := ...some value...;
```

To undo one either waits for the current scope (e.g. a LaTeX environment) to expire or the impatient does:

```
\xintunassignvar{e}
```



In this special case of using \xintunassignvar with a single ⟨*letter*⟩ the effect is actually to let the ⟨*letter*⟩ recover the meaning of a dummy variable (i.e. it is the same as using \xintnewdummy documented in the next section):

```
% overwriting a dummy letter
\xintdefvar i := 3;%
\xinteval{i^3} is as expected but |\xinteval{add(i,i=1..10)}| computes
\xinteval{add(i,i=1..10)} because "i" has the fixed value 3.\newline
\xintunassignvar{i}% back to normal
After |\xintunassignvar{i}|
|\xinteval{add(i,i=..10)}| is evluated with "i" acting as a dummy variable
and thus outputs \xinteval{add(i,i=1..10)}.\par
```

27 is as expected but \xinteval{add(i,i=1..10)} computes 30 because "i" has the fixed value 3. After \xintunassignvar{i} \xinteval{add(i,i=..10)} is evluated with "i" acting as a dummy variable and thus outputs 55.

Under \xintglobaldefstrue regime the effect of \xintunassignvar is of global scope.

3.9.2. \xintnewdummy

code

Any catcode 11 character can serve as a dummy variable, via this declaration:

```
\xintnewdummy{<letter>}% the <letter> must be of catcode letter!
```

For example with LuaTeX or XeTeX the following works:

```
% Requires a Unicode engine
\xintnewdummy{ξ}
\xinteval{add(ξ, ξ=1..10)}
```

Starting with 1.4, it is allowed to use \xintnewdummy to define ``dummy variables'' having names with more than one letter. They can then be used as expected:

```
% Requires a Unicode engine
\xintnewdummy{ατλν}
\xintdeffunc test(ατλν) = sqr(1 + ατλν);
\xinteval{seq(test(ατλν), ατλν = 0..10)}
```

Remark regarding OpTeX: it is a format using LuaTeX, but non ascii letters have catcode ``other''. So for the above example to compile with optex, the catcodes need to be set to ``letter'' beforehand:

```
% Requires a Unicode engine
% next line is required if with OpTeX (only):
\xintFor* #1 in {ατλν}\do{\catcode`#1=11 }
\xintnewdummy{ατλν}
\xintdeffunc test(ατλν) = sqr(1 + ατλν);
\xinteval{seq(test(ατλν), ατλν = 0..10)}
```

Under \xintglobaldefstrue regime the effect of \xintnewdummy is of global scope.

3.9.3. \xintensuredummy, \xintrestorevariable

code *code*

Use

```
\xintensuredummy{<character>}
...
... code using the (catcode 11) character as a dummy variable
```

```
...
\xintrestorevariable{<character>}
```

if other parts need the letter as an assigned variable name. For example calxtrig being written at high level needs a few genuine dummy variables, and it uses \xintensuredummy to be certain everything is ok.

3.10. User defined functions

3.10.1 \xintdeffunc 51
3.10.2 \xintdefiifunc 54
3.10.3 \xintdeffloatfunc 54
3.10.4 \xintdefufunc, \xintdefiiufunc, \xintdeffloatufunc 54
3.10.5 Using the same name for both a variable and a function 57
3.10.6 \xintunassignexprfunc, \xintunassigniiexprfunc, \xintunassignfloatexprfunc 57
3.10.7 \ifxintverbose conditional 57
3.10.8 \ifxintglobaldefs conditional 58
3.10.9 \xintNewFunction 58

3.10.1. \xintdeffunc

code

Here is an example:

```
\xintdeffunc
    Rump(x,y):=1335 y^6/4 + x^2 (11 x^2 y^2 - y^6 - 121 y^4 - 2) + 11 y^8/2 + x/2y;
```

(notice the numerous tacit multiplications in this expression; and that x/2y is interpreted as x/(2y).)

- The ending semi-colon is allowed to be of active catcode, as \xintdeffunc temporarily resets catcodes via \xintexprSafeCatcodes before parsing the expression.

  But this will fail if the whole thing is inside a macro definition. Then the used semi-colon must be the standard one.

  In the case of a LaTeX document using Babel, and a language such as French which makes the semi-colon active, it is still the standard one inside the preamble, so there is no problem there.

  For a macro definition done inside the document body, which, as I understand, is sin, almost evil, either turn off locally the activation (\string; will not work because \xintdeffunc uses delimited macros to fetch all the way to the semi-colon), or define in the preamble \MyDefFunc{#1} to do \xintdeffunc #1; and use that in the \newcommand\foo inside the document body.

- Semi-colons used inside the expression need not be hidden inside braces. (new with 1.4)
- The colon before the equal sign is optional and its (reasonable) catcode does not matter.

Here are a few important items (bookmark this for reading again later once you have gained experience in using this interface...):

- The function names are composed of letters, digits, underscores or @ signs. A function name must start with a letter. It may be a single letter (see subsubsection 3.10.5).
- The variable names used in the function signature may be multi-letter words. It is also allowed for them to already be in use for previously declared variables. Their meanings will get restored for usage after the function declaration.

- A function can be declared with at most nine arguments. It can be declared as a function with no arguments.
- If in the function declaration the last argument is prefixed by *, it stands for a nutple which will gather all arguments of the function call beyond the first positional ones. See subsubsection 3.13.6 for additional explanations on such "variadic" arguments.
- Recursive definitions are possible; for them to not generate error or fall in infinite loops, the use of the short-circuit conditionals ? and ?? is *mandatory*.
- If a function is used in another definition it will check if it is applied to numerical arguments and if this is the case will expand fully.
- The previous item has an exception for functions with no arguments; they never expand immediately in other function definitions (else they would be almost like variables). This provides a way to define functions with parameters: simply let their definition use some functions with no arguments.
- A function declared via \xintdeffunc remains unknown to \xintfloatexpr (or \xintfloateval). See \xintdeffloatfunc, \xintdefiifunc. One can use the same formula in a new definition, but if one wants the expansion to execute in a parser independent way, one can transfer a function with scalar values like this:

  ```
  \xintdeffloatfunc foo(x) := float_dgt(\xintexpr foo(x)\relax);
  ```

  The float_dgt() wrapper (which was renamed at 1.4e) is in order for the float variant to produce an already-rounded value, possibly speeding-up usage if used as input for other functions. Using float() here would work the same but the produced function would incorporate a routine to check (at time of use, because at time of definition it is impossible to tell what will be the case) if it is applied to one or two arguments.
- And in the reverse direction one can do:

  ```
  \xintdeffunc bar(x) := \xintfloatexpr bar(float(x))\relax;
  ```

  With this the transplanted float-function will expand in \xintexpr as it would have in \xintfloatexpr, i.e. using float operations; this is different from declaring the function again with the same expression as used for the original, as it would have then been parsed with a mapping of infix operators to the macros doing the exact operations, not the floating point ones.

  The inner float() above is not mandatory but recommended: the macro associated to the user float function bar(x) may use many times its argument x and does not worry about rounding it, because its expectation is that it is already rounded; but in \xintexpr that value could very well be a fraction 19/13 and its float rounding will be done again by each float macro receiving it as argument; with a float() used as above this will have already been done once and the ulterior roundings are faster: they have nothing to do apart from realizing that they have nothing to do.... One can also use sfloat(), this would serve to nothing for the 19/13 case but would possibly for a short integer input involved in multiplications.

  Here it is not needed to use float_dgt(), because it will be identified at time of definition that float() is used without optional argument.

  An external float_dgt() could be added but is not a priori necessary, except perhaps if the bar() function has been defined at a low level using support macros producing output with garbage extra digits, which usually would be rounded out in input to other float functions.

A function once declared is a first class citizen, its expression is entirely parsed and converted into a big nested *f*-expandable macro.

When used its action is via this defined macro. For example

```
\xintdeffunc
     e(z):=(((((((((z/10+1)z/9+1)z/8+1)z/7+1)z/6+1)z/5+1)z/4+1)z/3+1)z/2+1)z+1;
```

creates a macro whose meaning one can find in the log file, after \xintverbosetrue. Here it is (it has at 1.4 an extra external brace pair compared to what happened with earlier releases):

```
    Function e for \xintexpr parser associated to \XINT_expr_userfunc_e with me
aning macro:#1->{\xintAdd {\xintMul {\xintAdd {\xintDiv {\xintMul {\xintAdd {\x
intDiv {\xintMul {\xintAdd {\xintDiv {\xintMul {\xintAdd {\xintDiv {\xintMul {\
xintAdd {\xintDiv {\xintMul {\xintAdd {\xintDiv {\xintMul {\xintAdd {\xintDiv {
\xintMul {\xintAdd {\xintDiv {\xintMul {\xintAdd {\xintDiv {#1}{10}}{1}}{#1}}{9
}}{1}}{#1}}{8}}{1}}{#1}}{7}}{1}}{#1}}{6}}{1}}{#1}}{5}}{1}}{#1}}{4}}{1}}{#1}}{3}
}{1}}{#1}}{2}}{1}}{#1}}{1}}
```

The above is not entirely true. At 1.4, \xintdeffunc is more powerful and digests more of the syntax but it may have to store it in such a way that usage will be done via a sub-expression: hence it is not the case that the original expression has been *entirely* parsed. See \xintNewFunction for related discussion.

The main difficulty of \xintdeffunc is with the pseudo-functions seq(), iter(), etc..., which admit the keywords omit, abort, break(). We have no alternative for them, if the iterated over values are not entirely numerical than to postpone expansion, but this means simply storing for later a possibly big sub-expression.

At 1.4 we did some obstinate work to make this working but:

- this means that the stored function body has not been entirely parsed, parsing will happen on the fly at each execution for small or large bits,
- there remains a main stumbling-block. If the variables used in the function declaration are used only in the iterated over values or the initial values, then the mechanism may work. If however they are used not only in those values iterated over but directly in the expression which the generators map to the iterated over values, then it will break certainly. Indeed at this stage the variables are simply names, and it is impossible to transfer the mechanism which converts these names into numerical arguments for delayed usage by the declared function. Except if one is ready to basically freeze the entire thing; which then is not any different at all than using \xintNewFunction.

Conclusion: if some \xintdeffunc break, check if it does not fit the above criterion before reporting... and recall \xintNewFunction is your friend. It has the big advantage of declaring a function for all parsers simultaneously!

A special note on subs(): it is and has always been hopeless in \xintdeffunc context. All it does (if it works at all) after being malaxed by \xintdeffunc is to copy over at the indicated places the *recipe* to compute something. Thus at every location where that something is needed it will be evaluated from scratch again. Yes, this is disappointing. But... on the other hand the more general seq() does work, or pretends to work. Let me illustrate to make thinks clear. We start with this:

```
\xintverbosetrue
\xintdeffunc foo(x,y,z) = subs(S + S^2, S = x+y+z);
\xintdeffunc bar(x,y,z) = seq(S + S^2, S = x+y+z);
\xintexpr foo(100,10,1), bar(100,10,1)\relax
\xintverbosefalse
```

```
12432, 12432
```

It produces in the log:

```
Package xintexpr Info: (on line 2)
    Function foo for \xintexpr parser associated to \XINT_expr_userfunc_foo wit
h meaning macro:#1#2#3->{\xintAdd {\xintAdd {\xintAdd {#1}{#2}}{#3}}{\xintPow {
\xintAdd {\xintAdd {#1}{#2}}{#3}}{2}}}
Package xintexpr Info: (on line 3)
```

```
    Function bar for \xintexpr parser associated to \XINT_expr_userfunc_bar wit
  h meaning macro:#1#2#3->\expanded \bgroup \expanded {\unexpanded {\XINT_expr_se
  q:_b {\xintbareeval S + S^2\relax !S}}{\xintAdd {\xintAdd {#1}{#2}}{#3}}^}
```

Even without understanding all details one sees that in the first case the \xintAdd {\xintAdd {#1}{#2}}{#3} appears twice, and in the second case only once. But in the second case we have a yet to evaluate expression. So the second approach is not much different in its effect than using the more simple-minded \xintNewFunction. Besides one gets a feeling why the function arguments can not appear in the expression but only in the iterated over values, because there is no way to understand what x, y, z are supposed to mean without adding extra structure showing they map to #1, #2, #3.

The above remarks apply to subsm() and subsn(). Even if they do work in \xintdeffunc context (warning, testing at 1.4 release has remained minimal), they will not bring added efficiency if the substituted values are to be used multiple times. They may still be useful to visually simplify the input of a big expression by expressing it in terms of smaller constituents.

Another workaround if one wants genuine (not "macro"-) functions for some expression where the same thing is used multiple times is to define helper functions computing the intermediate data. One can see illustrations of this in the code source of calxtrig (or in the matrix multiplication example at the end of this chapter).

3.10.2. \xintdefiifunc

code

With \xintdeffunc the created function is known by the \xintexpr parser only. For usage in the \xintiiexpr parser, it is required to use \xintdefiifunc.

3.10.3. \xintdeffloatfunc

code

With \xintdeffunc the created function is known by the \xintexpr parser only. For usage in the \xintfloatexpr parser, it is required to use \xintdeffloatfunc.

Note: the optional argument [Q] accepted by \xintfloatexpr does not work with \xintdeffloatfunc. It is still possible to wrap the expression in float(expression,Q), if it evaluates to a scalar.

3.10.4. \xintdefufunc, \xintdefiiufunc, \xintdeffloatufunc

code *code* *code*

This allows to define so-called "Universal functions". This is terminology borrowed from NumPy.

Here is an example:

```
\xintdefiivar Array = ndmap(lcm, 1..5; 1..10; 1..10);
Array = \xintthealign\xintiiexpr Array\relax
\xintdefiiufunc foo(x) = x^3;
\begin{figure}[htbp]
\caption{Output of a universal function acting on an array}\label{fig:ufunc}
\centeredline{$\vcenter{\xintthealign\xintiiexpr foo(Array)\relax}$}
\end{figure}
See \autopageref{fig:ufunc} for the output.
```

```
Array =
[[[  1,   2,   3,    4,    5,    6,    7,    8,    9,    10   ],
  [  2,   2,   6,    4,    10,   6,    14,   8,    18,   10   ],
  [  3,   6,   3,    12,   15,   6,    21,   24,   9,    30   ],
  [  4,   4,   12,   4,    20,   12,   28,   8,    36,   20   ],
  [  5,   10,  15,   20,   5,    30,   35,   40,   45,   10   ],
  [  6,   6,   6,    12,   30,   6,    42,   24,   18,   30   ],
  [  7,   14,  21,   28,   35,   42,   7,    56,   63,   70   ],
  [  8,   8,   24,   8,    40,   24,   56,   8,    72,   40   ],
```

```
  [  9,   18,  9,    36,   45,   18,   63,   72,   9,    90   ],
  [  10,  10,  30,   20,   10,   30,   70,   40,   90,   10   ]],
 [[  2,   2,   6,    4,    10,   6,    14,   8,    18,   10   ],
  [  2,   2,   6,    4,    10,   6,    14,   8,    18,   10   ],
  [  6,   6,   6,    12,   30,   6,    42,   24,   18,   30   ],
  [  4,   4,   12,   4,    20,   12,   28,   8,    36,   20   ],
  [  10,  10,  30,   20,   10,   30,   70,   40,   90,   10   ],
  [  6,   6,   6,    12,   30,   6,    42,   24,   18,   30   ],
  [  14,  14,  42,   28,   70,   42,   14,   56,   126,  70   ],
  [  8,   8,   24,   8,    40,   24,   56,   8,    72,   40   ],
  [  18,  18,  18,   36,   90,   18,   126,  72,   18,   90   ],
  [  10,  10,  30,   20,   10,   30,   70,   40,   90,   10   ]],
 [[  3,   6,   3,    12,   15,   6,    21,   24,   9,    30   ],
  [  6,   6,   6,    12,   30,   6,    42,   24,   18,   30   ],
  [  3,   6,   3,    12,   15,   6,    21,   24,   9,    30   ],
  [  12,  12,  12,   12,   60,   12,   84,   24,   36,   60   ],
  [  15,  30,  15,   60,   15,   30,   105,  120,  45,   30   ],
  [  6,   6,   6,    12,   30,   6,    42,   24,   18,   30   ],
  [  21,  42,  21,   84,   105,  42,   21,   168,  63,   210  ],
  [  24,  24,  24,   24,   120,  24,   168,  24,   72,   120  ],
  [  9,   18,  9,    36,   45,   18,   63,   72,   9,    90   ],
  [  30,  30,  30,   60,   30,   30,   210,  120,  90,   30   ]],
 [[  4,   4,   12,   4,    20,   12,   28,   8,    36,   20   ],
  [  4,   4,   12,   4,    20,   12,   28,   8,    36,   20   ],
  [  12,  12,  12,   12,   60,   12,   84,   24,   36,   60   ],
  [  4,   4,   12,   4,    20,   12,   28,   8,    36,   20   ],
  [  20,  20,  60,   20,   20,   60,   140,  40,   180,  20   ],
  [  12,  12,  12,   12,   60,   12,   84,   24,   36,   60   ],
  [  28,  28,  84,   28,   140,  84,   28,   56,   252,  140  ],
  [  8,   8,   24,   8,    40,   24,   56,   8,    72,   40   ],
  [  36,  36,  36,   36,   180,  36,   252,  72,   36,   180  ],
  [  20,  20,  60,   20,   20,   60,   140,  40,   180,  20   ]],
 [[  5,   10,  15,   20,   5,    30,   35,   40,   45,   10   ],
  [  10,  10,  30,   20,   10,   30,   70,   40,   90,   10   ],
  [  15,  30,  15,   60,   15,   30,   105,  120,  45,   30   ],
  [  20,  20,  60,   20,   20,   60,   140,  40,   180,  20   ],
  [  5,   10,  15,   20,   5,    30,   35,   40,   45,   10   ],
  [  30,  30,  30,   60,   30,   30,   210,  120,  90,   30   ],
  [  35,  70,  105,  140,  35,   210,  35,   280,  315,  70   ],
  [  40,  40,  120,  40,   40,   120,  280,  40,   360,  40   ],
  [  45,  90,  45,   180,  45,   90,   315,  360,  45,   90   ],
  [  10,  10,  30,   20,   10,   30,   70,   40,   90,   10   ]]]
```

See page 56 for the output.

The function can be applied to any nested strucure:

```
\xintiiexpr foo([1, [2, [3, [4, [5, 6, 7, 8, 9, 10]]]]])\relax
[1, [8, [27, [64, [125, 216, 343, 512, 729, 1000]]]]]
```

It must be defined as function acting on scalars, but its value type is not constrained.

```
\xintdefiivar Array = [1..10];
\xintdefiiufunc foo(x) = [1..x];
\xintthealign\xintiiexpr foo(Array)\relax
[[  1   ],
 [  1,  2   ],
```

Figure 1: Output of a universal function acting on an array

```
[[[  1,      8,       27,      64,      125,     216,     343,      512,      729,      1000     ],
  [  8,      8,       216,     64,      1000,    216,     2744,     512,      5832,     1000     ],
  [  27,     216,     27,      1728,    3375,    216,     9261,     13824,    729,      27000    ],
  [  64,     64,      1728,    64,      8000,    1728,    21952,    512,      46656,    8000     ],
  [  125,    1000,    3375,    8000,    125,     27000,   42875,    64000,    91125,    1000     ],
  [  216,    216,     216,     1728,    27000,   216,     74088,    13824,    5832,     27000    ],
  [  343,    2744,    9261,    21952,   42875,   74088,   343,      175616,   250047,   343000   ],
  [  512,    512,     13824,   512,     64000,   13824,   175616,   512,      373248,   64000    ],
  [  729,    5832,    729,     46656,   91125,   5832,    250047,   373248,   729,      729000   ],
  [  1000,   1000,    27000,   8000,    1000,    27000,   343000,   64000,    729000,   1000     ]],
 [[  8,      8,       216,     64,      1000,    216,     2744,     512,      5832,     1000     ],
  [  8,      8,       216,     64,      1000,    216,     2744,     512,      5832,     1000     ],
  [  216,    216,     216,     1728,    27000,   216,     74088,    13824,    5832,     27000    ],
  [  64,     64,      1728,    64,      8000,    1728,    21952,    512,      46656,    8000     ],
  [  1000,   1000,    27000,   8000,    1000,    27000,   343000,   64000,    729000,   1000     ],
  [  216,    216,     216,     1728,    27000,   216,     74088,    13824,    5832,     27000    ],
  [  2744,   2744,    74088,   21952,   343000,  74088,   2744,     175616,   2000376,  343000   ],
  [  512,    512,     13824,   512,     64000,   13824,   175616,   512,      373248,   64000    ],
  [  5832,   5832,    5832,    46656,   729000,  5832,    2000376,  373248,   5832,     729000   ],
  [  1000,   1000,    27000,   8000,    1000,    27000,   343000,   64000,    729000,   1000     ]],
 [[  27,     216,     27,      1728,    3375,    216,     9261,     13824,    729,      27000    ],
  [  216,    216,     216,     1728,    27000,   216,     74088,    13824,    5832,     27000    ],
  [  27,     216,     27,      1728,    3375,    216,     9261,     13824,    729,      27000    ],
  [  1728,   1728,    1728,    1728,    216000,  1728,    592704,   13824,    46656,    216000   ],
  [  3375,   27000,   3375,    216000,  3375,    27000,   1157625,  1728000,  91125,    27000    ],
  [  216,    216,     216,     1728,    27000,   216,     74088,    13824,    5832,     27000    ],
  [  9261,   74088,   9261,    592704,  1157625, 74088,   9261,     4741632,  250047,   9261000  ],
  [  13824,  13824,   13824,   13824,   1728000, 13824,   4741632,  13824,    373248,   1728000  ],
  [  729,    5832,    729,     46656,   91125,   5832,    250047,   373248,   729,      729000   ],
  [  27000,  27000,   27000,   216000,  27000,   27000,   9261000,  1728000,  729000,   27000    ]],
 [[  64,     64,      1728,    64,      8000,    1728,    21952,    512,      46656,    8000     ],
  [  64,     64,      1728,    64,      8000,    1728,    21952,    512,      46656,    8000     ],
  [  1728,   1728,    1728,    1728,    216000,  1728,    592704,   13824,    46656,    216000   ],
  [  64,     64,      1728,    64,      8000,    1728,    21952,    512,      46656,    8000     ],
  [  8000,   8000,    216000,  8000,    8000,    216000,  2744000,  64000,    5832000,  8000     ],
  [  1728,   1728,    1728,    1728,    216000,  1728,    592704,   13824,    46656,    216000   ],
  [  21952,  21952,   592704,  21952,   2744000, 592704,  21952,    175616,   16003008, 2744000  ],
  [  512,    512,     13824,   512,     64000,   13824,   175616,   512,      373248,   64000    ],
  [  46656,  46656,   46656,   46656,   5832000, 46656,   16003008, 373248,   46656,    5832000  ],
  [  8000,   8000,    216000,  8000,    8000,    216000,  2744000,  64000,    5832000,  8000     ]],
 [[  125,    1000,    3375,    8000,    125,     27000,   42875,    64000,    91125,    1000     ],
  [  1000,   1000,    27000,   8000,    1000,    27000,   343000,   64000,    729000,   1000     ],
  [  3375,   27000,   3375,    216000,  3375,    27000,   1157625,  1728000,  91125,    27000    ],
  [  8000,   8000,    216000,  8000,    8000,    216000,  2744000,  64000,    5832000,  8000     ],
  [  125,    1000,    3375,    8000,    125,     27000,   42875,    64000,    91125,    1000     ],
  [  27000,  27000,   27000,   216000,  27000,   27000,   9261000,  1728000,  729000,   27000    ],
  [  42875,  343000,  1157625, 2744000, 42875,   9261000, 42875,    21952000, 31255875, 343000   ],
  [  64000,  64000,   1728000, 64000,   64000,   1728000, 21952000, 64000,    46656000, 64000    ],
  [  91125,  729000,  91125,   5832000, 91125,   729000,  31255875, 46656000, 91125,    729000   ],
  [  1000,   1000,    27000,   8000,    1000,    27000,   343000,   64000,    729000,   1000     ]]]
```

```
[  1,  2,  3   ],
[  1,  2,  3,  4   ],
[  1,  2,  3,  4,  5   ],
[  1,  2,  3,  4,  5,  6   ],
[  1,  2,  3,  4,  5,  6,  7   ],
[  1,  2,  3,  4,  5,  6,  7,  8   ],
[  1,  2,  3,  4,  5,  6,  7,  8,  9   ],
[  1,  2,  3,  4,  5,  6,  7,  8,  9,  10  ]]
```

It is even allowed to produce oples and act on oples:

```
\xintdefiivar Ople = 1..10;
\xintdefiiufunc bar(x) = x, x^2, x^3;
\xintiiexpr bar(Ople)\relax
```

```
1, 1, 1, 2, 4, 8, 3, 9, 27, 4, 16, 64, 5, 25, 125, 6, 36, 216, 7, 49, 343, 8, 64, 512, 9, 81, 729,
10, 100, 1000
```

3.10.5. Using the same name for both a variable and a function

It is licit to overload a variable name (all Latin letters are predefined as dummy variables) with a function name and vice versa. The parsers will decide from the context if the function or variable interpretation must be used (dropping various cases of tacit multiplication as normally applied).

```
\xintdefiifunc f(x):=x^3;
\xinttheiiexpr add(f(f),f=100..120)\relax\newline
\xintdeffunc f(x,y):=x^2+y^2;
\xinttheexpr mul(f(f(f,f),f(f,f)),f=1..10)\relax
\xintunassigniiexprfunc{f}\xintunassignexprfunc{f}%
```

```
28205100
1861881348675788854278488064000000000
```

3.10.6. \xintunassignexprfunc, \xintunassigniiexprfunc, \xintunassignfloatexprfunc

code *code* *code*

Function names can be unassigned via \xintunassignexprfunc{⟨*name*⟩}, \xintunassigniiexprfunc{⟨*name*⟩}, and \xintunassignfloatexprfunc{⟨*name*⟩}.

```
\xintunassignexprfunc{e}
\xintunassignexprfunc{f}
```

```
Warning: no check is done to avoid undefining built-in functions...
```

3.10.7. \ifxintverbose conditional

code

With \xintverbosetrue the meanings of the functions (or rather their associated macros) will be written to the log. For example the Rump declaration above generates this in the log file:

```
     Function Rump for \xintexpr parser associated to \XINT_expr_userfunc_Rump w
ith meaning macro:#1#2->{\xintAdd {\xintAdd {\xintAdd {\xintDiv {\xintMul {1335
}{\xintPow {#2}{6}}}{4}}{\xintMul {\xintPow {#1}{2}}{\xintSub {\xintSub {\xintS
ub {\xintMul {11}{\xintMul {\xintPow {#1}{2}}{\xintPow {#2}{2}}}}{\xintPow {#2}
{6}}}{\xintMul {121}{\xintPow {#2}{4}}}}{2}}}}{\xintDiv {\xintMul {11}{\xintPow
 {#2}{8}}}{2}}}{\xintDiv {#1}{\xintMul {2}{#2}}}}
```

☞ The meanings written out to the log for more complicated functions may sometimes use the same character at different locations but with different catcodes.

It may thus be impossible to retokenize it (even after having removed the extra spaces from the added line breaks).

This is in contrast with variable values which are always output in the log in the benign way, using digits, braces and some characters of catcode 12.

3.10.8. \ifxintglobaldefs conditional

code

If true user defined variables (\xintdefvar, ...) and functions (\xintdeffunc, ..., \xintNewFunction) for the expression parsers, as well as macros obtained via \xintNewExpr et al. have global scope. If false (default) they have local scope.

Since 2.0a2, this conditional also governs scientific-library reconstruction. With \xintglobaldefstrue, the new library precision, tables and associated definitions survive the end of the group. The arithmetic precision (\xinttheDigits) is still local. Only the currently installed tables are kept; successive reloads at the same precision and scope do not rebuild them.

3.10.9. \xintNewFunction

code

This is syntactic sugar which allows to use notation of functions for what is nothing more in disguise than a TEX macro. Here is an example:

```
\xintNewFunction {foo}[3]{add(mul(x+i, i=#1..#2),x=1..#3)}
```

We now have a genuine function foo() of three variables which can be used in *all three parsers*.

```
\xintexpr seq(foo(0, 3, j), j= 1..10)\relax
```

```
24, 144, 504, 1344, 3024, 6048, 11088, 19008, 30888, 48048
```

Each time the created "macro-function" foo() will be encountered the corresponding replacement text will get inserted as a sub-expression (of the same type as the surrounding one), the macro parameters having been replaced with the (already evaluated) function arguments, and the parser *will then have to parse the expression.* It is very much like a macro substitution, but with parentheses and comma separated arguments (which can be arbitrary expressions themselves).

It differs fundamentally from \xintdeffunc as it realizes no pre-parsing whatsoever of the associated sub-expression; using it shortens the input but not the parsing time (which however is most of the time negligible compared to actual numerical computations). Use it for syntax which \xintdeffunc does not parse successfully.

3.11. Examples of user defined functions

3.11.1. Example with vectors and matrices

Suppose we want to manipulate 3-dimensional vectors, which will be represented as nutples of length 3. And let's add a bit of matrix algebra.

```
\xintdeffunc dprod(V, W) := V[0]*W[0] + V[1]*W[1] + V[2]*W[2];
\xintdeffunc cprod(V, W) := [V[1]*W[2] - V[2]*W[1],
                             V[2]*W[0] - V[0]*W[2],
                             V[0]*W[1] - V[1]*W[0]];
\xintdeffunc Det3(U, V, W) := dprod(cprod(U, V), W);
\xintdeffunc DetMat(M) = Det3(*M);
\xintdeffunc RowMat(U, V, W) := [U, V, W];
\xintdeffunc ColMat(U, V, W) := [[U[0], V[0], W[0]],
                                 [U[1], V[1], W[1]],
                                 [U[2], V[2], W[2]]];
\xintdeffunc MatMul(A, B) :=
 [[A[0,0]*B[0,0]+A[0,1]*B[1,0]+A[0,2]*B[2,0],
```

```
    A[0,0]*B[0,1]+A[0,1]*B[1,1]+A[0,2]*B[2,1],
    A[0,0]*B[0,2]+A[0,1]*B[1,2]+A[0,2]*B[2,2]],
   [A[1,0]*B[0,0]+A[1,1]*B[1,0]+A[1,2]*B[2,0],
    A[1,0]*B[0,1]+A[1,1]*B[1,1]+A[1,2]*B[2,1],
    A[1,0]*B[0,2]+A[1,1]*B[1,2]+A[1,2]*B[2,2]],
   [A[2,0]*B[0,0]+A[2,1]*B[1,0]+A[2,2]*B[2,0],
    A[2,0]*B[0,1]+A[2,1]*B[1,1]+A[2,2]*B[2,1],
    A[2,0]*B[0,2]+A[2,1]*B[1,2]+A[2,2]*B[2,2]]];

\xintdefvar vec1, vec2, vec3 := [1, 1, 1], [1, 1/2, 1/4], [1, 1/3, 1/9];
\xintdefvar mat1 = RowMat(vec1, vec2, vec3);
\xintdefvar mat2 = ColMat(vec1, vec2, vec3);
\xintdefvar mat12 = MatMul(mat1,mat2);
\xintdefvar mat21 = MatMul(mat2,mat1);
Some computations (|align| executes multiple times hence we pre-computed!):
\begin{align*}
   M_1      &= \vcenter{\xintthealign \xintexpr mat1\relax}&&\qquad
   M_2 . M_1 = \vcenter{\xintthealign \xintexpr mat21\relax}\\[3\jot]
   M_2      &= \vcenter{\xintthealign \xintexpr mat2\relax}&&\qquad
   M_1 . M_2 = \vcenter{\xintthealign \xintexpr mat12\relax}
\end{align*}
$$
   \det(M_1)     = \xinteval{DetMat(mat1)},\quad
   \det(M_1.M_2) = \xinteval{reduce(DetMat(mat12))},\quad
   \det(M_2.M_1) = \xinteval{reduce(DetMat(mat21))}
$$
```

Some computations (align executes multiple times hence we pre-computed!):

$$M_1 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1/2 & 1/4 \\ 1 & 1/3 & 1/9 \end{bmatrix} \qquad M_2.M_1 = \begin{bmatrix} 3 & 11/6 & 49/36 \\ 11/6 & 49/36 & 251/216 \\ 49/36 & 251/216 & 1393/1296 \end{bmatrix}$$

$$M_2 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1/2 & 1/3 \\ 1 & 1/4 & 1/9 \end{bmatrix} \qquad M_1.M_2 = \begin{bmatrix} 3 & 7/4 & 13/9 \\ 7/4 & 21/16 & 43/36 \\ 13/9 & 43/36 & 91/81 \end{bmatrix}$$

$$\det(M_1) = -1/18, \quad \det(M_1.M_2) = 1/324, \quad \det(M_2.M_1) = 1/324$$

For some hair-raising experience check the \xintverbosetrue output in the log... here is an alternative with two (three, counting dprod()) helper functions:

```
% annoying that Tr also starts Trace, but Spur is available
% well Sp also starts Spectrum. Big problems.
\xintdeffunc Tr(M) :=
     [[M[0,0], M[1,0], M[2,0]],
      [M[0,1], M[1,1], M[2,1]],
      [M[0,2], M[1,2], M[2,2]]];

\xintdeffunc MatMul_a(r1, r2, r3, c1, c2, c3) :=
     [[dprod(r1, c1), dprod(r1, c2), dprod(r1, c3)],
      [dprod(r2, c1), dprod(r2, c2), dprod(r2, c3)],
      [dprod(r3, c1), dprod(r3, c2), dprod(r3, c3)]];

\xintdeffunc MatMul(A, B) := MatMul_a(*A, *Tr(B));
```

And once we have the transpose and the scalar product of vectors, we can simply use ndmap() for a lean syntax (this would extend to arbitrary dimension):

```
\xintdeffunc MatMul(A, B) = ndmap(dprod, *A; *Tr(B));
\xintdefvar mat1212 = MatMul(mat12, mat12);
\begingroup
\def\xintexprPrintOne       #1{\xintTeXFrac{#1}}%
\def\xintexpralignbegin       {\begin{pmatrix}}%
\def\xintexpralignend         {\end{pmatrix}}%
\def\xintexpralignlinesep     {\noexpand\\[2\jot]}% counteract an internal \expanded
\def\xintexpralignninnersep   {&}%
\let\xintexpralignleftbracket\empty  \let\xintexpralignleftsep\empty
\let\xintexpralignrightbracket\empty \let\xintexpralignrightsep\empty
$$ \xintthealign \xintexpr mat1\relax \cdot \xintthealign \xintexpr mat2\relax \cdot
   \xintthealign \xintexpr mat1\relax \cdot \xintthealign \xintexpr mat2\relax =
   \xintthealign \xintexpr mat12\relax ^2 = \xintthealign \xintexpr mat1212\relax$$
$$ \det(M_1\cdot M_2 \cdot M_1 \cdot M_2) = \xinteval{reduce(DetMat(mat1212))}$$
\endgroup
```

$$\begin{pmatrix}1 & 1 & 1\\ 1 & \frac{1}{2} & \frac{1}{4}\\ 1 & \frac{1}{3} & \frac{1}{9}\end{pmatrix}\cdot\begin{pmatrix}1 & 1 & 1\\ 1 & \frac{1}{2} & \frac{1}{3}\\ 1 & \frac{1}{4} & \frac{1}{9}\end{pmatrix}\cdot\begin{pmatrix}1 & 1 & 1\\ 1 & \frac{1}{2} & \frac{1}{4}\\ 1 & \frac{1}{3} & \frac{1}{9}\end{pmatrix}\cdot\begin{pmatrix}1 & 1 & 1\\ 1 & \frac{1}{2} & \frac{1}{3}\\ 1 & \frac{1}{4} & \frac{1}{9}\end{pmatrix}=\begin{pmatrix}3 & \frac{7}{4} & \frac{13}{9}\\ \frac{7}{4} & \frac{21}{16} & \frac{43}{36}\\ \frac{13}{9} & \frac{43}{36} & \frac{91}{81}\end{pmatrix}^2=\begin{pmatrix}\frac{18337}{1296} & \frac{48067}{5184} & \frac{93853}{11664}\\ \frac{48067}{5184} & \frac{128809}{20736} & \frac{253687}{46656}\\ \frac{93853}{11664} & \frac{253687}{46656} & \frac{501289}{104976}\end{pmatrix}$$

$$\det(M_1\cdot M_2\cdot M_1\cdot M_2)=\frac{1}{104976}$$

3.11.2. Example with the Rump test

Let's try out our Rump() function:

```
\xinttheexpr Rump(77617,33096)\relax.
```

-54767/66192. Nothing problematic for an *exact* evaluation, naturally!

Thus to test the Rump polynomial (it is not quite a polynomial with its x/2y final term) with floats, we *must* also declare Rump as a function to be used there:

```
\xintdeffloatfunc
    Rump(x,y):=333.75 y^6 + x^2 (11 x^2 y^2 - y^6 - 121 y^4 - 2) + 5.5 y^8 + x/2y;
```

The numbers are scanned with the current precision, hence as here it is 16, they are scanned exactly in this case. We can then vary the precision for the evaluation.

```
\def\CR{\cr}
\halign
{\tabskip1ex
\hfil\bfseries#&\xintDigits:=\xintiloopindex\relax
                \xintthefloatexpr Rump(77617,33096)#\cr
\xintiloop [8+1]
\xintiloopindex &\relax\CR
\ifnum\xintiloopindex<40 \repeat
}
```

```
 8 7e29
 9 -1e28
10 5e27
11 -3e26
12 4e25
13 3e24
14 3e23
```

```
15 -2e22
16 1e21
17 -5e20
18 1.17260394005317863
19 1.000000000000000001e18
20 -9.9999999999999998827e16
21 1.000000000000000011726e16
22 3.000000000000001172604e15
23 -9.99999999999988273960060e13
24 -1.99999999999988273960599e13
25 -1.99999999999988273960599947e12
26 1.17260394005317863185885883490
27 -5.99999999988273960599468214e10
28 -9.9999999882739605994682136814e8
29 2.000000011726039400531786318588e8
30 1.00000011726039400531786318588e7
31 -999998.82739605994682136814116509548
32 200001.17260394005317863185883490
33 -9998.827396059946821368141165095480
34 -1998.8273960599468213681411650954798162
35 -198.82739605994682136814116509547982
36 21.172603940053178631858834904520183700
37 -0.827396059946821368141165095479816292000
38 -0.8273960599468213681411650954798162919990
39 -0.82739605994682136814116509547981629200
40 -0.827396059946821368141165095479816292000
```

3.11.3. Examples of recursive definitions

Recursive definitions *require* using the short-circuit branching operators.

```
\xintdeffunc GCD(a,b):=(b)?{GCD(b,a/:b)}{a};
```

This of course is the Euclide algorithm: it will be here applied to variables which may be fractions. For example:

```
\xinttheexpr GCD(385/102, 605/238)\relax
55/714
```

There is already a built-in gcd() (which accepts arbitrarily many arguments):

```
\xinttheexpr gcd(385/102, 605/238)\relax
55/714
```

Our second example is modular exponentiation:

```
\xintdefiifunc powmod_a(x, m, n) :=
       isone(m)?
           % m=1, return x modulo n
           {   x /: n  }
           % m > 1 test if odd or even and do recursive call
           {   odd(m)? {  x*sqr(powmod_a(x, m//2, n)) /: n }
                       {    sqr(powmod_a(x, m//2, n)) /: n }
            }
         ;
\xintdefiifunc powmod(x, m, n) := (m)?{powmod_a(x, m, n)}{1};
```

I have made the definition here for the \xintiiexpr parser; we could do the same for the \xintexpr-parser (but its usage with big powers would quickly create big denominators, think powmod(1/2, 1000, 1) for example.)

```
\xinttheiiexpr seq(powmod(x, 1000, 128), x=9, 11, 13, 15, 17, 19, 21)\relax\par
```

```
65, 97, 33, 1, 1, 33, 97
```

The function assumes the exponent is non-negative (the Python pow behaved the same until 3.8 release), but zealous users will add the necessary code for negative exponents, after having defined another function for modular inverse!

If function A needs function B which needs function A start by giving to B some dummy definition, define A, then define B properly. TODO: add some example here...

3.12. Links to some (old) examples within this document

- The utilities provided by xinttools (section 7), some completely expandable, others not, are of independent interest. Their use is illustrated through various examples: among those, it is shown in subsection 8.9 how to implement in a completely expandable way the Quick Sort algorithm and also how to illustrate it graphically. Other examples include some dynamically constructed alignments with automatically computed prime number cells: one using a completely expandable prime test and \xintApplyUnbraced (subsection 8.3), another one with \xintFor* (subsection 8.7).
- One has also a computation of primes within an \edef (subsection 7.16), with the help of \xintiloop. Also with \xintiloop an automatically generated table of factorizations (subsection 8.6).
- The code for the title page fun with Fibonacci numbers is given in subsection 4.19 with \xintFor* joining the game.
- The computations of π and log 2 (subsection 16.12) using xint and the computation of the convergents of e with the further help of the xintcfrac package are among further examples.
- Also included, an expandable implementation of the Brent-Salamin algorithm for evaluating π.
- The subsection 8.5 implements expandably the Miller-Rabin pseudo-primality test.
- The functionalities of xintexpr are illustrated with various other examples, in subsubsection 3.10.1, Functions with dummy variables, subsection 8.2 or Recursive definitions.

3.13. Oples and nutples: the 1.4 terminology

Skip this on first reading, else you will never start using the package. SKIP THIS! (understood?)

In this section I will describe a mathematical terminology which models how the parser handles the input syntax with numbers, commas, and brackets, and how it maps internally to TEX specific concept, particularly braces and macro arguments.

.13.1 Base terminology . . . 62
.13.2 Items (and sub-items) versus elements . . . 63
.13.3 Oples as trees . . . 64
.13.4 Ople slicing and indexing . . . 65
.13.5 Nested slicing of oples . . . 65
.13.6 Function arguments versus variables . . . 66
.13.7 Final words on leaves . . . 66
.13.8 Farewell, thanks for your visit! . . . 67

3.13.1. Base terminology

We start with a set $\mathcal{A}$ of *atoms*, which represent numeric data. In TEX syntax such *atoms* are always braced, more precisely, currently they look like

{raw format within TEX braces}

The TEX braces are not set-theoretical braces here, they are simply used for TEXnical reasons (one

could imagine using rather some terminator token, but ultimately support macros for built-in and user defined functions rely on TEX macros with undelimited parameters, at least so far).

Our category $\mathcal{C}$ of "oples" is the smallest collection of *totally ordered finite sets* verifying these properties:

1. The empty set $\emptyset$ is an *ople*, i.e. it belongs to $\mathcal{C}$.
2. Each singleton set {O} whose element O is either an *atom* a $\in \mathcal{A}$ or an *ople* qualifies as an *ople*.
3. $\mathcal{C}$ is stable by concatenation.

Notes:

- We refer to the empty set $\emptyset$ via the variable *nil*.[19]
- It is convenient to accept the empty set as being also an *atom*. If this is done, then we may refer to the original *atoms* (elements of $\mathcal{A}$) as *non empty numerical data*.
- Concatenation is represented in the syntax by the comma. Thus repeated commas are like only one and nil is a neutral element.
- A singleton *ople* {a} whose single element is a (non-empty) *atom* is called a *number*.[20]
- The operation of constructing {O} from the *ople* O is called *bracing* (set theory, TEX), or *bracketing* (xintexpr input syntax, Python lists), or *packing* (as a reverse to Python's unpacking of sequence type objects). In the expression input syntax it corresponds to enclosing O within square brackets: [O].
- A braced *ople* is called a *nutple*. Among them {nil} (aka {$\emptyset$}) is a bit special. It is called the *none-ple*.[21] It is not nil.[22]

Each *ople* has a *length* which is its cardinality as set. The singleton oples are called *one-ples*. There are thus two types of *one-ples*:

- *numbers* {a}, a $\in \mathcal{A}$,
- *nutples* {O}, O $\in \mathcal{C}$.

If we consider the empty set nil on the same footing as atoms, the two types have only one common object which is the *none-ple*. As a rule arithmetic operations will either break or silently convert the *none-ple* to the zero value:

```
\xinteval{3+[], 5^[], 10*[]}
```

3, 1, 0 . But attention that \xintiieval in contrast to \xinteval is broken by such inputs.

3.13.2. Items (and sub-items) versus elements

In order to illustrate these concepts, let us consider how one should interpret notation such as 3,5,7,9 when it arises in an \xintexpression:

tempting vocabulary: Each of 3, 5, 7, and 9 is an *item*, or *element* of the (comma separated) *list*. In other terms we have here a list with 4 items.

rigorous vocabulary: each one of 3, 5, 7, 9 stands for an *ople* (of the *one-ple* type) and 3,5,7,9 stands for their *concatenation*.

[19] There is actually a built-in variable with this name. At 1.4, \xintexpr \relax is legal and also generates the *nil*. [20] This has to be taken in a general sense, for example with polexpr, polynomials are represented by such "numbers". [21] Prior to version 1.4j of this documentation it was called the *not-ple*. [22] There is (experimental) a pre-defined "None" variable which stands for the *none-ple*. It can also be input as [].

It is important to understand that in an \xintexpression, there is no difference between 3,5,7 and 3,,,,5,,,,,,,,,7. So the view of the comma as separator is misleading. In other terms, the comma is NOT a separator but the (associative) operator of concatenation of totally ordered sets, and the number 3 for example represents a (singleton) set.

If we want to refer to 3 or 5 or 7 or 9 as "the items of the (open) list 3,5,7,9" (and probably this documentation already has such utterances, due to legacy reasons from the pre-1.4 internal model), we *must* realize that this clashes with using the word *item* as synonymous to *element* in the set-theoretical sense.

To repeat, any ople O is a finite totally ordered set: if not the empty set, it has *elements* a_1, ..., a_k, and the above means that its *items* are the singleton oples (aka one-ples) $I_1 = \{a_1\}$, ..., $I_k = \{a_k\}$. Each a_j may be an atom, then I_j is a number, or a_j is an ople (possibly the empty set), then I_j is a nutple whose depth is one more than the one of the ople a_j.

Thus we can refer to "items" but must then understand they are not "elements": "items" are "singleton sub-sets". The cardinality (aka length) of an ople is also the number of its items. It would be tempting to use the terminology "sub-item" to keep in mind they are "sub-sets" but this would again create confusion: a nutple has only one item which is itself; and we need some terminology to refer to the individual numbers in the nutple given in input as [1,2,3] for example. It is natural to refer to 1, 2, 3 as "sub-items" of [1,2,3] as the latter may be an "item" (it is in particular an "item" of itself, the unique one at that).

We distinguish the oples of length zero (there is only one, the empty set) or at least two as those which can never be an "item". Those of length one, the one-ples, are exactly those which can be "items". Among them some may have "sub-items", they are the nutples with the exception of the none-ple. And the others do not have "sub-items", they are the numbers and the none-ple (whose input syntax is either [] or the variable None).[23]

3.13.3. Oples as trees

We say that the empty set nil and *atoms* are *leaves*.

We associate with any *ople* a tree. The root is the ople. In the case of the nil ople, there is nothing else than the root, which we then consider also a *leaf*. Else the children at top level are the successive *elements* (not "items"!) of the ople.[24] Among the elements some are *atoms* giving *leaves* of the tree, others are *nutples* which in turn have children. In the special case of the *none-ple* we consider it has a child, which is the empty set and this is why we consider the empty set nil to be also a potential *leaf*. We then proceed recursively. We thus obtain from the root *ople* a tree whose vertices are either *oples* or *leaves*. Only the empty set nil is both a *leaf* and an *ople*.

Considering the empty set nil as an *atom* fits with the xintexpr internal implementation based on TEX: nil is an empty pair of braces {}, whereas an *atom* is a braced representation of a numeric value using digits and other characters. We construct *oples* by putting one after the other such constituents and bracing them, and then repeating the process recursively.

It has also an impact on the definition of the *depth* (a.k.a as *maximal dimension*) of an *ople*. For example the *ople* $\{\emptyset A_1 A_2\}$ with three elements, among them the empty set and two atoms is said to have depth 1, or to have maximal dimension 1. And $\{\{\emptyset\}A_1A_2\}$ is of depth 2 because it has a leaf (the

[23] A note on the \xintverbosetrue regime: for a variable defined to be 3,5,7,9, it will say that its value is {3}{5}{7}{9}, because it does not keep the external set-theoretical braces. The braces here are only TEX braces, and {3} is an atom. The number would be {{3}} with the external braces being set-theoretical and also used internally as TEX braces. From the four numbers {{3}}, ..., {{9}} concatenation gives {{3}{5}{7}{9}}, which is the ople 3,5,7,9. But the log view drops deliberately the external braces. If the variable is defined to be the nutple [3,5,7,9], then the log view will be {{3}{5}{7}{9}} (up to details on how exactly the numeric quantities are coded) and the actual internal TEX entity will be {{{3}{5}{7}{9}}}, where the two external layers of braces are both set-theoretical and TEXnical braces. [24] We could also consider a tree for which the children of the root node would be its items and recursively; in that case the leaves would be numbers and possibly the None. The tree of the nil would be the empty tree, the tree of None would have a single node and no edges. Such a tree would match the input syntax (of course applying the rule that iterated commas are like only one). The tree which is described in this section matches more directly the internal syntax, hence is more useful to the author, who is also the sole reader who extracts some benefit from reading this documentation once in a while.

empty set) which is a child of a child of the *ople*. NumPy *ndarrays* have a more restricted structure for example {{$A_{00}A_{01}$}{$A_{10}A_{11}$}} is a 2-dimensional array, where all leaves are at the same depth. When slicing empties the array from its atoms, NumPy keeps the shape information but prints the array as []. This will not be the case with xintexpr, which has no other way to indicate the shape than display it.

```
  \xinteval{[[],[]]}
[[], []]
  \xinteval{[[0,1],[10,11]][:,2:]}
[[], []]
```

3.13.4. Ople slicing and indexing

"Set-theoretical" slicing of an *ople* means replacing it with one of its subsets. This applies also if it is a *number*. Then it can be sliced only to itself or to the empty set (indeed it has only one element, which is an atom). Similarly the *none-ple* can only be sliced to give itself or the empty set. And more generally a *nutple* is a singleton so also can only be set-sliced to either the empty set or itself.

xintexpr extends "Python-like" slicing to act on *oples*:

- if they are not *nutples* set-theoretical slicing applies,
- if they are *nutples* (only case having a one-to-one correspondence in Python) then the slicing happens *within brackets*: i.e. the *nutple* is unpacked then the set-theoretical slicing is applied, then the result is *repacked* to produce a new *nutple*.

With these conventions the *none-ple* for example is invariant under slicing: unpacking it gives the empty set, which has only the empty set as subset and repacking gives back the *none-ple*. Slicing a general *nutple* returns a *nutple* but now of course in general distinct from the first one.

The input syntax for Python slicing is to postfix a variable or a parenthesized ople with [a:b]. See subsection 3.6 for more. There are never any out-of-range errors when slicing or indexing. All operations are licit and resolved by the nil, a.k.a. empty set.

"Set-theoretical" item indexing of an *ople* means reducing it to a subset which is a singleton. It is thus a special case of set-theoretical slicing (which is the general process of selecting a subset as replacement of a set).

xintexpr extends "Python-like" indexing to act on *oples*:

- if they are not *nutples* set-theoretical item indexing applies,
- if they are *nutples* (only case having a one-to-one correspondence in Python) then the meaning becomes *extracting*: i.e. the *nutple* is unpacked then the set-theoretical indexing is applied, but the result is *not repacked*.

For example when applied to the *none-ple* we always obtain the nil. Whereas as we saw slicing the *none-ple* always gives back the *none-ple*. Indexing is denoted in the syntax by postfixing by [N]. Thus for *nutples* (which are analogous to Python objects), there is genuine difference between the [N] extractor and the [N:N+1] slicer. But for *oples* which are either nil, a *number*, or of length at least 2, there is no difference.

3.13.5. Nested slicing of oples

Nested slicing is a concept from NumPy, which is extended by xintexpr to trees of varying depths. We have a chain of slicers and extractors. I will describe only the case of slicers and letting them act on a nutple. The first slicer gives back a new nutple. The second slicer will be applied to each of one of its remaining elements. However some of them may be *atoms* or the empty set. In the NumPy context all leaves are at the same depth thus this can happen only when we have reached beyond the last dimension (axis). This is not permitted by NumPy and generates an error. xintexpr does not generate an error. But any attempt to slice an *atom* or the empty set (as element of its container) removes it. Recall we call them *leaves*. We can not slice leaves. We can only slice non-leaf elements: such items are necessarily nutples. The procedure then applies recursively.

If we handle an extractor rather than a slicer, the procedure is similar: we can not extract out of an *atom* or the empty set. They are thus removed. Else we have a nutple. It is thus unpacked and replaced by the selected element. This element may be an atom or the empty set and any further slicer or extractor will remove them, or it is a nutple and the procedure applies with the next slicer/extractor.

xintexpr allows to apply such a [a:b,c:d,N,e:f,...] chain of slicing/extracting also to an *ople*, which is not a *nutple*. We simply apply the first step as has been described previously and successive steps will only get applied to either *nutples* or *leaves*, the latter getting silently removed by any attempted operation.

3.13.6. Function arguments versus variables

In a function declaration with \xintdeffunc, the call signature is parsed as a comma separated list, so here it is not true that repeated commas are like only one: repeated commas are not allowed and will break the function declaration.

When xintexpr parses a function call, it first constructs the ople which is delimited by the opening and closing parentheses, then it applies the function body, after having mapped the successive items (not the elements) of the parsed ople to the variables appearing in the function call signature. Hence the arguments in the call signature stand for one-ples (i.e. either numbers or nutples).

Let me explain why we can not define a function foo(A,B) of two oples: the function call will evaluate as an ople what is enclosed within the parentheses. It is then impossible in general to split this uniquely into two oples A and B, except if for example we know a priori the length of A. We could imagine defining a declarative interface for a foo(A,B) with A preset to have 37 items or at least a pre-defined number of items but this is extraneous layer for a functionality no-one will use.

The alternative would be to consider that declaring foo(A,B) means A will pick-up always the first item and B all the remaining ones, and thus will be an ople; here, there are some TEXnical implementation reasons which have dissuaded the author to do this.

In its place, a special syntax foo(A,*B) for the declaration of the function is available. It means that B stands for the nutple which receives as items all arguments in the function call beyond the first one already assigned to A.

More generally, the last positional argument in a function declaration can have the form *⟨*argname*⟩. This then means that ⟨*argname*⟩ represents a nutple which will receive as items all arguments in the function call remaining after the earlier positional arguments have been assigned. The declared function body is free to again use the syntax *⟨*argname*⟩ which will unpack it and thus produce the ople concatenating all such optional arguments.

With \xintdefvar one can define a variable with value an ople of arbitrary cardinality. Such a variable can be used in a function call, it will then occupy the place of as many arguments as its cardinality (which is its number of elements, hence of its associated items). For example if function foo was declared as a function of 5 arguments f(a,b,c,d,e) it is legitimate to use it as f(A,B) if A is an ople-valued variable of length three and B of length two. The actual arguments a,b,c,d,e will be made to match the three items of A and the two items of B.

3.13.7. Final words on leaves

In case things were too clear, let's try to add a bit of confusion with an extra word on *leaves*. When we discuss informally (particularly to compare with NumPy) an input such as

```
[[1, 2], [3, 4]]
```

we may well refer to 1, 2, 3, and 4 as being "the leaves of the 2d array". But obviously we have here numbers and previously we explained that a number is not a *leaf*, its *atom* is. Well, the point here is that we must make a difference between the input form as above and the actual constructed *ople* the parser will obtain out of it. In the input we do have numbers. The comma is a *concatenator*,

it is not a separator for enumeration! The *ople* which corresponds to it has a TEX representation like this:

```
{{{1}{2}}{{3}{4}}}
```

where we don't have the *numbers* anymore (which would look like {{1}}, {{2}}, ...) but numeric *atoms* {1}, {2}, {3}, {4} where the braces are TEX braces and **not** set-theoretical braces (the other braces are both). Hence we should see the above as the ople {{$A_{00}A_{01}$}{$A_{10}A_{11}$}} with atoms A_{00} = {1}, ..., being the *leaves* of the tree associated to (or which is) the *ople*.

Numbers may be called the *leaves* of the **input**, but once parsed, the input becomes an *ople* which is (morally) a tree whose leaves are *atoms* (and the empty set). This discussion can also be revisited with footnote 24 in mind.

3.13.8. Farewell, thanks for your visit!

I hope this is clear to everyone. If not, maybe time to say this section is not needed to understand almost all of the manual, but I needed to write it to be able to maintain in future my own software.

3.14. Expansion (for geeks only)

As mentioned already, the parsers are compatible with expansion-only context.

Also, they expand the expression piece by piece: the normal mode of operation of the parsers is to unveil the parsed material token by token. Unveiling is a process combining space swallowing, brace removal (one level generally), and *f*-expansion.

For example a closing parenthesis after some function arguments does not have to be immediately visible, it and the arguments themselves may arise from *f*-expansion (applied before grabbing each successive token). Even the ending \relax may arise from expansion. Even though the \xinteval user interface means that the package has at some point the entire expression in its hands, it immediately re-inserts it into token stream with an additional postfixed \relax and from this point on has lost any ways (a simple-minded delimited macro won't do because the expression is allowed to contain sub-\xintexpressions, even nested) to manipulate formally again the whole thing; it can only re-discover it one token at a time.

This general behaviour (which allows much more freedom in assembling expressions than is usually the case with familiar programming languages such as Python, although admittedly that freedom will prove useful only to power-TEXusers and possibly does not have that many significant use cases) has significative exceptions. These exceptions are mostly related to "pseudo"-functions. A "pseudo"-function will grab some of its arguments via delimited macros. For example subs(expr
↪ 1,x=expr2) needs to see the comma, equal sign and closing parenthesis. But it has mechanisms to allow expr1 and expr2 to possess their own commas and parentheses.

Inner semi-colons on the other hand currently always can originate from expansion. Defining functions or variables requires a visible semi-colon acting as delimiter of the expression, but inner semi-colons do not need to be hidden within braces or macros.

The expansion stops only when the ending \relax has been found (it is then removed from the token stream).

For catcode related matters see \xintexprSafeCatcodes.

A word of warning on the bracketed optional argument of respectively \xintfloatexpr and \xintiexpr. When defining macros which will hand over some argument to one of these two parsers, the argument may potentially start with a left square bracket [(e.g. argument could be [1, 2, 3]) and this will break the parser. The fix is to use in the macro definition \xintfloatexpr\empty. This extra \empty token will prevent the parser from thinking there is an optional argument and it will then disappear during expansion.

If comparing to other languages able to handle floating point numbers or big integers, such as Python, one should take into account that what the xint packages manipulate are streams of ascii bytes, one per digit. At no time (due to expandability) is it possible to store intermediate results in an arithmetic CPU register; each elementary operation via \the\numexpr will output digit tokens (hence as many bytes), not things such as handles to memory locations where some numbers are stored as memory words. The process can never put aside things but can only possibly permute them with upcoming

tokens, to use them later, or, via combinations of \expanded and \unexpanded or some other more antiquated means grab some tokens and shift the expansion to some distant locations to later come back. The process is a never-ending one-dimensional one...

3.15. Known bugs/features (last updated at 1.4n)

\xinteval{\xintLength{\par\par\par}} complains about a Runaway argument:

\xintLength{\par\par\par} has no issue as \xintLength is a \long macro but this is not the case of \xinteval. Most macros of a non arithmetic nature in xintkernel and xinttools are declared \long but absolutely none in xintexpr, and its dependencies xint, etc... As a remark in passing, I could not use the LaTeX \item directly:[25]

```
Runaway argument?
{|\xinteval {\xintLength {
! Paragraph ended before \@item was complete.
<to be read again>
                   \par
l.4434 \item[{|\xinteval{\xintLength{\par
                                        \par\par}| complains about a Runawa...
```

This is the reason I guess why everything is a priori \long in the LaTeX3 interface except if asked for otherwise (as far as I know).

Although most macros are dealing with inputs which can only be with digits and some other character tokens, it would still be quite some work to chase all top-level ones. Besides, in pratice, it does help better locate ill-formed input.

if(100>0,(100,125),(100,128)) breaks my code: This is a feature. This is a syntax error, as the comma serves to contatenate "oples" (see subsection 3.13), and parentheses do not create analogs of "tuples", so this input is parsed the same as

```
if(100>0,100,125,100,128)
```

which is an error as if() requires exactly three arguments, not five. Use:

```
if(100>0,[100,125],[100,128])
```

which will expand to the "tuple" [100,125].

\xintdeffunc foo(x):= gcd((x>0)?{[x,125]}{[x,128]}); creates a broken function: Bug. Normally gcd() (and other multi-arguments functions) work both with open lists of arguments or bracketed lists ("nutples") and the above syntax would work perfectly fine in numerical context. But the presence of the ? breaks in \xintdeffunc context the flexibility of gcd().

Currently working alternatives:

```
\xintdeffunc foo(x) := gcd(if(x>0, [x,125], [x,128]));
\xintdeffunc foo(x) := if(x>0, gcd(x,125), gcd(x,128));
\xintdeffunc foo(x) := if(x>0, gcd([x,125]), gcd([x,128]));
\xintdeffunc foo(x) := gcd((x>0)?{x,125}{x,128});
\xintdeffunc foo(x) := (x>0)?{gcd(x,125)}{gcd(x,128)};
\xintdeffunc foo(x) := (x>0)?{gcd([x,125])}{gcd([x,128])};
```

The same problem will arise with an ?? nested inside gcd() or similar functions, in an \xintdeffunc.

\xinteval{0^-.5} says "0 raised to power -1" Prior to 2.0, half-integer exponents were handled via a square-root extraction, hence the reported power -1. This specialization has been removed.

[25] For those who wonder my custom \verb employs a \scantokens approach, so it can be used in the argument of a macro, for example \footnote {\verb |\verb |}.

Comparison operator == crashes with nutples Not yet implemented...

I liked the "broadcasting" `[1..10]^10` syntax, but it was removed at 1.4 Patience... `seq(x^10,x=1..10)` is alternative (add external `[..]` to get a nutple).

`1e\numexpr 5+2\relax` crashes Not clear yet if bug or feature. The syntax accepted in the scientific part is limited, and failure is expected: hitting a `\numexpr` when parsing a number triggers insertion of a tacit multiplication and then `1e` is missing the scientific exponent. The same happens with `1e(2+3)`. Use syntax such as `1e\the\numexpr5+2\relax`, or `1e\xinteval{5+2}` (although here this relies on output format of `\xinteval` using integer notation with no decoration in this case).

`seq(1e-i,i=1..5)` crashes Not clear if bug or feature. Use `seq(1e\xinteval{-i},i=1..5)` or, as a possibly faster way `seq(1e\xintiieval{-i},i=\xintiiexpr1..5\relax)`.

omit/abort if nested and not last in the sub-expression cause a crash For example `seq(subs((i)?{i}{abort},t=i)+10, i=-2, -1, 0, 1)` crashes, due to the presence of the `+10`. This is a long-standing limitation, applying ever since `omit/abort` were added to the syntax at 1.1. Even without the `+10` the nested case was broken by a 1.4 regression and got fixed only at 1.4h. The non-nested case `seq((i)?{i}{abort}+10, i=-2, -1, 0, 1)` works and the "must be last in expression if nested" limitation is currently considered a feature.

`\xintdeffunc X(a,k)= add(n^k, n=1..a);` creates a broken function: Bug. Sadly `\xintdeffunc` has problems. When it does not work, use `\xintNewFunction` to define the function. Examples:

```
  \xintNewFunction{myX}[2]{add(n^#2, n=1..#1)}\xinteval{myX(10,7)}\newline
  \xintNewFunction{myY}[2]{mul(1 - j/#1, j=1..#2-1)}\xintfloateval{myY(10,10)}
18080425
0.00036288
```

`seq([i,i^2], i=1..10)` crashes with Ooops, looks like we are missing a]. Aborting! Bug. The cause is that the square brackets do not hide the comma from `seq()` parsing. Contrarily to what happens with parentheses, there is no balancing mechanism for square brackets. Work-arounds: either use an extra pair of parentheses `seq(([i,i^2]), ...)` or hide the inner comma within braces `seq([i{,}i^2], ...)`.

`subs([x,x^2],x=3);` crashes with Ooops, looks like we are missing a]. Aborting! Bug. Same cause as previous one. Workaround: use parentheses `subs(([x,x^2]),x=3` or curly braces `subs({[x,x^2]},x=3`.

```
  \xinteval{subs({[x^10,x^20,x^30]}, x=17)}
[2015993900449, 4064231406647572522401601, 8193465725814765556554001028792218849]
```

`seq([x,2x,3x],x=3..5);` crashes with Ooops, looks like we are missing a]. Aborting! Bug. Same cause as previous one. The workaround is again to use braces to hide the inner commas.

```
  \xinteval{seq({[x^10,x^20,x^30]}, x=1, 2, 3)}
[1, 1, 1], [1024, 1048576, 1073741824], [59049, 3486784401, 205891132094649]
```

`iter([1,10^6];[sqrt(@[0]*@[1]),(@[0]+@[1])/2], i=1..7)` also complains with Ooops, looks like we are missing a]. Aborting! Bug. Turns out that braces would not do the job here, but parentheses do work:

```
  % 32 digits and 8 iterations v------ parentheses added ------v
  \xintfloateval{iter([1,10^6];([sqrt(@[0]*@[1]),(@[0]+@[1])/2]), i=1..8)}
[103329.59376570941022723837701642, 103329.59376570941022723837701642]
```

\xintNewFunction {foo}[1]{#1} Functions defined via \xintNewFunction cannot currently be used reliably in function definitions made using \xintdeffunc syntax. For example, after the previous definition the definition \xintdeffunc bar(x):=foo(x)*x; triggers a spurious error message about a missing closing parenthesis.

More bugs are known to the author and many more no doubt exist.

4. The macros of xintexpr (ancient documentation, mostly)

.1 Introduction 71
.2 The \xintexpr expressions 71
.3 \numexpr or \dimexpr expressions, count and dimension registers and variables 74
.4 Catcodes and spaces 74
.5 Expandability, \xintexpro 75
.6 \xintDigits*, \xintSetDigits* 76
.7 \xintiexpr, \xinttheiexpr 76
.8 \xintiiexpr, \xinttheiiexpr 76
.9 \xintboolexpr, \xintthebooleexpr 77
.10 \xintfloatexpr, \xintthefloatexpr 78
.11 \xinteval, \xintieval, \xintiieval, \xintfloateval 78
.12 Using an expression parser within another one 79
.13 The \xintthecoords macro 79
.14 The \xintthespaceseparated macro 81
.15 \xintifboolexpr, \xintifboolfloatexpr, \xintifbooliiexpr 81
.16 \xintifsgnexpr, \xintifsgnfloatexpr, \xintifsgniiexpr 81
.17 The \xintNewExpr, \xintNewIIExpr, \xintNewFloatExpr, \xintNewIExpr, and \xintNewBoolExpr macros 82
.18 Analogies and differences of \xintiiexpr with \numexpr 83
.19 Chaining expressions for expandable algorithmics 84
.20 When expandability is too much 87
.21 Acknowledgements (2013/05/25) 88

4.1. Introduction

The xintexpr package was first released with version 1.07 (2013/05/25) of the xint bundle. It was substantially enhanced with release 1.1 from 2014/10/28.

The 1.4 release from 2020/01/31 maintains the same general architecture but needed adapting all the code base for the switch from \csname to \expanded techniques. On this occasion the mechanism for defining functions was substantially strengthened. The parser core mechanisms were improved too.

The package loads automatically xintfrac and xinttools.

This section should be trimmed to contain only information not already covered in section 3.

4.2. The \xintexpr expressions

code

x ★ An xintexpression is a construct \xintexpr⟨*expandable_expression*⟩\relax where the expandable expression is read and completely expanded from left to right.

An \xintexpr...\relax *must* end in a \relax (which will be absorbed). Contrarily to a \numexpr expression, it is printable as is without a prefix \the or \number (don't use them with \xintexpr this will raise an error).

But one can use \xintthe prefix if one does need the explicit digits and other characters as in the final typesetted result.

As an alternative and equivalent syntax to

```
\xintexpr round(<expression>, D)\relax
```

there is

```
\xintiexpr [D] <expression> \relax
```

For D>0 this produces a decimal number with D figures after the decimal mark, which is the rounding of the expression. For D=0 the rounding to an integer is produced. For D<0 (and this was changed at 1.4f), the rounded quotient of the expression by 1e|D| is produced.

- the expression may contain arbitrarily many levels of nested parenthesized sub-expressions,
- the expression may contain explicitely or from a macro expansion a sub-expression \xintexpr..\relax, which itself may contain a sub-expressions etc. . .
- to let sub-contents evaluate as a sub-unit it should thus be either
  1. parenthesized,

2. or a sub-expression \xintexpr...\relax.

- to use an expression as argument to macros from xintfrac, or more generally to macros which expand their arguments, one must use the \xinttheexpr...\relax or \xintthe\xintexpr...\relax forms.
- one should not use \xintthe\xintexpr...\relax as a sub-constituent of another expression but only the \xintexpr...\relax form which is more efficient in this context.
- each xintexpression, whether prefixed or not with \xintthe, is completely expandable and obtains its result in two expansion steps.

The information now following is possibly in need of updates.

- An expression is built the standard way with opening and closing parentheses, infix operators, and (big) numbers, with possibly a fractional part, and/or scientific notation (except for \xintiiexpr which only admits big integers). All variants work with comma separated expressions. On output each comma will be followed by a space. A decimal number must have digits either before or after the decimal mark.
- As everything gets expanded, the characters ., +, -, *, /, ^, !, &, |, ?, :, <, >, =, (,), ",], [, @ and the comma , should not (if used in the expression) be active.

NEW WITH 1.4n

— Babel-activated characters (for example !, ?, ; and : with French) are not a problem.
— If the character is active due to some other mechanism, prefix it with \string.
— A few syntax elements involving the comma, the equal sign and the closing parenthesis are implemented using delimited macros. They are not allowed to be catcode active (even via Babel) and \string will not work. Use then \xintexprSafeCatcodes, see next.

One can use \xintexprSafeCatcodes to reset all characters potentially needed by \xintexpr to their standard catcodes and \xintexprRestoreCatcodes then restores the former status.

Note that this is what \xintdefvar and \xintdeffunc do automatically. Expandable \xintfloateval et al. can't do that.

- Count registers and \numexpr-essions are accepted (LaTeX's counters can be inserted using \value) natively without \the or \number as prefix. Also dimen registers and control sequences, skip registers and control sequences (LaTeX's lengths), \dimexpr-essions, \glueexpr-essions are automatically unpacked using \number, discarding the stretch and shrink components and giving the dimension value in sp units (1/65536th of a TeX point). Furthermore, tacit multiplication is implied, when the (count or dimen or glue) register or variable, or the (\numexpr or \dimexpr or \glueexpr) expression is immediately prefixed by a (decimal) number. See subsection 3.8 for the complete rules of tacit multiplication.
- With a macro \x defined like this:

```
\def\x {\xintexpr \a + \b \relax} or \edef\x {\xintexpr \a+\b\relax}
```

one may then do \xintthe\x, either for printing the result on the page or to use it in some other macros expanding their arguments. The \edef does the computation immediately but keeps it in a protected form. Naturally, the \edef is only possible if \a and \b are already defined. With both approaches the \x can be inserted in other expressions, as for example (assuming naturally as we use an \edef that in the `yet-to-be computed' case the \a and \b now have some suitable meaning):

```
\edef\y {\xintexpr \x^3\relax}
```

- There is also \xintboolexpr ... \relax and \xinttheboolexpr ... \relax.
- See also \xintifboolexpr (subsection 4.15) and the bool() and togl() functions in section 3. Here is an example. Well in fact the example ended up using only \xintboolexpr so it was modified to use \xintifboolexpr.

```
\xintdeffunc A(p,q,r) =  p && (q || r) ;
\xintdeffunc B(p,q,r) =  p || (q && r) ;
\xintdeffunc C(p,q,r) =  xor(p, q, r)  ;

\centeredline{\normalcolor
\begin{tabular}{ccrclcl}
        \xintFor* #1 in  {{False}{True}} \do {%
        \xintFor* #2 in  {{False}{True}} \do {%
        \xintFor* #3 in  {{False}{True}} \do {%
    #1 &AND &(#2 &OR  &#3)&is&\textcolor[named]{OrangeRed}
                            {\xintifboolexpr{A(#1,#2,#3)}{true}{false}}\\
    #1 &OR  &(#2 &AND &#3)&is&\textcolor[named]{OrangeRed}
                            {\xintifboolexpr{B(#1,#2,#3)}{yes}{no}}\\
    #1 &XOR & #2 &XOR &#3 &is&\textcolor[named]{OrangeRed}
                            {\xintifboolexpr{C(#1,#2,#3)}{oui}{non}}\\
}}}
\end{tabular}%
}
```

False	AND	(False	OR	False)	is	false
False	OR	(False	AND	False)	is	no
False	XOR	False	XOR	False	is	non
False	AND	(False	OR	True)	is	false
False	OR	(False	AND	True)	is	no
False	XOR	False	XOR	True	is	oui
False	AND	(True	OR	False)	is	false
False	OR	(True	AND	False)	is	no
False	XOR	True	XOR	False	is	oui
False	AND	(True	OR	True)	is	false
False	OR	(True	AND	True)	is	yes
False	XOR	True	XOR	True	is	non
True	AND	(False	OR	False)	is	false
True	OR	(False	AND	False)	is	yes
True	XOR	False	XOR	False	is	oui
True	AND	(False	OR	True)	is	true
True	OR	(False	AND	True)	is	yes
True	XOR	False	XOR	True	is	non
True	AND	(True	OR	False)	is	true
True	OR	(True	AND	False)	is	yes
True	XOR	True	XOR	False	is	non
True	AND	(True	OR	True)	is	true
True	OR	(True	AND	True)	is	yes
True	XOR	True	XOR	True	is	oui

- See also \xintifsgnexpr.
- There is \xintfloatexpr ... \relax where the algebra is done in floating point approximation (also for each intermediate result). Use the syntax \xintDigits:=N\relax to set the precision. Default: 16 digits.

```
\xintthefloatexpr 2^100000\relax: 9.990020930143845e30102
```

The square-root operation can be used in \xintexpr, it is computed as a float with the precision set by \xintDigits or by the optional second argument:

```
\xinttheexpr sqrt(2,60)\relax\newline
Here the [60] is to avoid truncation to |\xintthe Digits| of precision on output.
\newline
```

```
\printnumber{\xintthefloatexpr [60] sqrt(2,60)\relax}
1.41421356237309504880168872420969
Here the [60] is to avoid truncation to \xinttheDigits of precision on output.
1.41421356237309504880168872420969
```

Floats are quickly indispensable when using the power function, as exact results will easily have hundreds, even thousands of digits.

```
\xintDigits:=48\relax \xintthefloatexpr 2^100000\relax
9.99002093014384507944032764330033590980429139054e30102
```

Non-integer exponents, integer exponents outside the \numexpr range, and certain powers near the exponent range limits use the logarithmic library.

- if one uses *macros* within \xintexpr..\relax one should obviously take into account that the parser will *not* see the macro arguments, hence one cannot use the syntax there, except if the arguments are themselves wrapped as \xinttheexpr...\relax and assuming the macro *f*-expands these arguments.

4.3. \numexpr or \dimexpr expressions, count and dimension registers and variables

Count registers, count control sequences, dimen registers, dimen control sequences (like \parindent), skips and skip control sequences, \numexpr, \dimexpr, \glueexpr, \fontdimen can be inserted directly, they will be unpacked using \number which gives the internal value in terms of scaled points for the dimensional variables: 1pt = 65536sp (stretch and shrink components are thus discarded).

Tacit multiplication (see subsection 3.8) is implied, when a number or decimal number prefixes such a register or control sequence. LaTeX lengths are skip control sequences and LaTeX counters should be inserted using \value.

Release 1.2 of the \xintexpr parser also recognizes and prefixes with \number the \ht, \dp, and \wd TeX primitives as well as the \fontcharht, \fontcharwd, \fontchardp and \fontcharic ε-TeX primitives.

In the case of numbered registers like \count255 or \dimen0 (or \ht0), the resulting digits will be re-parsed, so for example \count255 0 is like 100 if \the\count255 would give 10. The same happens with inputs such as \fontdimen6\font. And \numexpr 35+52\relax will be exactly as if 87 as been encountered by the parser, thus more digits may follow: \numexpr 35+52\relax 000 is like 87000. If a new \numexpr follows, it is treated as what would happen when \xintexpr scans a number and finds a non-digit: it does a tacit multiplication.

```
\xinttheexpr \numexpr 351+877\relax\numexpr 1000-125\relax\relax{} is the same
as \xinttheexpr 1228*875\relax.
1074500 is the same as 1074500.
```

Control sequences however (such as \parindent) are picked up as a whole by \xintexpr, and the numbers they define cannot be extended extra digits, a syntax error is raised if the parser finds digits rather than a legal operation after such a control sequence.

A token list variable must be prefixed by \the, it will not be unpacked automatically (the parser will actually try \number, and thus fail). Do not use \the but only \number with a dimen or skip, as the \xintexpr parser doesn't understand pt and its presence is a syntax error. To use a dimension expressed in terms of points or other TeX recognized units, incorporate it in \dimexpr...\relax.

Regarding how dimensional expressions are converted by TeX into scaled points see also subsection 9.7.

4.4. Catcodes and spaces

The main problems are caused by active characters, because \xintexpr et al. expand forward whatever comes from token stream; they apply \string only in a second step. For example the catcode

of & from && Boolean disjunction is not really important as long as it is not active, or comment, or escape... or brace... or ignored... in brief, as long as it is reasonable, and in particular whether @ is of catcode letter or other does not matter.

It is always possible to insert manually the \string in the expression before a problematic (but reasonable) character catcode, or even to use \detokenize for a big chunk.

4.4.1. \xintexprSafeCatcodes

code

Some problems with active characters can be resolved on the fly by prefixing them by \string but some aspects of the parsing done by \xintexpr involves delimited macros which need the comma, equality sign and closing parenthesis to have their standard catcodes.

So \xintexprSafeCatcodes is provided as a utility to set in one go catcodes of many characters to \xintexpr-safely compatible values. This is a non-expandable step as it changes catcodes.

\xintdefvar, \xintdeffunc, et al., use it, and then they restore catcodes to the prior state via \xintexprRestoreCatcodes.

4.4.2. \xintexprRestoreCatcodes

code

Restores the catcodes to the state prevailing at the time of the last executed \xintexprSafeCatcodes (if located at the same LaTeX environment or TeX grouping level).

Prior to 1.4k, in a situation like the following:

```
\xintexprSafeCatcodes
....stuff possibly changing catcodes
\xintexprSafeCatcodes
....stuff possibly changing catcodes
\xintexprSafeCatcodes
....stuff possibly changing catcodes
\xintexprRestoreCatcodes
```

On exit, the catcodes recovered their status as prior to the *first* \xintexprSafeCatcodes. Since 1.4k, they are set to what they were prior to the *last* \xintexprSafeCatcodes, i.e. the mechanism is now similar to a ``last in, first out'' stack.

Note that no global assignments are made so the behaviour can be modified by usage of TeX groups or LaTeX environments: e.g. if an \xintexprSafeCatcodes is issued inside a LaTeX environment it does not have to be paired by \xintexprRestoreCatcodes explicitly, the catcode scope is limited by the environment.

Spaces inside an \xinttheexpr...\relax should mostly be innocuous (except inside macro arguments).

\xintexpr and \xinttheexpr are for the most part agnostic regarding catcodes, but the characters in the expression should not be "active" (except on purpose) as everything is expanded along the way, and \xintexpr will choke on typesetting related commands. One can (in almost all cases) use \string to prefix a problematic character.

NEW WITH 1.4n

Babel-activated characters are not a problem.

Digits, slash, square brackets, minus sign, in the output from an \xinttheexpr are all of catcode 12. For \xintthefloatexpr the `e' in the output has its standard catcode ``letter''.

4.5. Expandability, \xintexpro

As is the case with all other package macros \xintexpr *f*-expands (in two steps) to its final (somewhat protected) result; and \xinttheexpr *f*-expands (in two steps) to the chain of digits (and

possibly minus sign -, decimal mark ., fraction slash /, scientific e, square brackets [,]) representing the result.

The once expanded \xintexpr is \romannumeral0\xintexpro. And there are similarly \xintiexpro \xintiiexpro and \xintfloatexpro. For an example see subsection 4.19.

An expression can only be legally terminated by a \relax token, which will be absorbed. This token may arose from expansion, it does not have to be immediately visible.

It is quite possible to nest expressions among themselves; for example, if one needs inside an \xintiiexpr...\relax to do some computations with fractions, rounding the final result to an integer, one just has to insert \xintiexpr...\relax. The functioning of the infix operators will not be in the least affected from the fact that the outer ``environment'' is the \xintiiexpr one.

4.6. \xintDigits*, \xintSetDigits*

These starred variants of \xintDigits and \xintSetDigits set the arithmetic precision and also reconstruct both scientific libraries at that precision (at most 100 digits). The non-starred variants leave the scientific-library precisions unchanged. An attempted reload above 100 digits issues a warning and retains the previous library precision. Use the non-starred forms for higher arithmetic precisions. If the arithmetic precision exceeds 100 when the libraries are first loaded, they are initialized at 100 digits, leaving arithmetic precision unchanged. See also \xintglobaldefstrue for the scope of library reconstruction.

The scientific functions use kernels at the next multiple of four and, when necessary, round their results to the requested library precision. At multiples of four this additional rounding is not needed. The arithmetic operations, square root and dedicated integer-power routine do not use this scheme.

4.7. code \xintiexpr, code \xinttheiexpr

x ★ Equivalent to doing \xintexpr round(...)\relax (more precisely, round is applied to each leaf item of the ople independently of its depth).

Intermediate calculations are exact, only the final output gets rounded. Half integers are rounded towards +∞ for positive numbers and towards -∞ for negative ones.

An optional parameter D within brackets, immediately after \xintiexpr is allowed: it instructs (for D>0) the expression to do its final rounding to the nearest value with that many digits after the decimal mark, i.e. \xintiexpr [D] <expression>\relax is equivalent (in case of a single expression) to \xintexpr round(<expression>, D)\relax.

\xintiexpr [0] ... is the same as \xintiexpr ... and rounds to an integer.

The case of negative D gives quantization to an integer multiple of 1e-D. This was modified at 1.4f and the produced value is now the rounded quotient by 1e-D (i.e. no trailing zeros nor scientific exponent in the output).

If truncation rather than rounding is needed on can use \xintexpr trunc(...)\relax for truncation to an integer or \xintexpr trunc(...,D)\relax for quantization to an integer multiple or 1eD (if D>0, for D<0 the analog would be trunc((...)/1e-D)). But this works only for a single scalar value.

> When defining a macro doing something such as \xintiexpr #1\relax, it is recommended to rather use \xintiexpr\empty #1\relax, as the #1 may start with a [which without the \empty would be interpreted by \xintiexpr as the start of the optional [D].

4.8. code \xintiiexpr, code \xinttheiiexpr

x ★ This variant does not know fractions. It deals almost only with long integers. Comma separated

lists of expressions are allowed.

> It maps / to the *rounded* quotient. The operator // is, like in \xintexpr...\relax, mapped to *truncated* division. The Euclidean quotient (which for positive operands is like the truncated quotient) was, prior to release 1.1, associated to /. The function quo(a,b) can still be employed.

The \xintiiexpr-essions use the `ii' macros for addition, subtraction, multiplication, power, square, sums, products, Euclidean quotient and remainder.

The floor() and ceil() functions are available in this integer only parser, with arguments allowed to be decimal numbers. But for example floor(5/3) returns the perhaps counter intuitive value 2, because / does the rounded division to an integer. To force the parser to allow fractions and not interpret / as integer-only operator, uses qfrac() as in the examples below.

Also round() and trunc() are allowed in \xintiiexpr-essions: they are mapped to \xintiRound and \xintiTrunc which explains how they behave with respect to their optional second argument.

```
\xinttheiiexpr 5/3, round(5/3,3), trunc(5/3,3),
\relax{} do the integer division
%
\xinttheiiexpr 5/3,  round(qfrac(5/3),3), trunc(qfrac(5/3),3), floor(qfrac(5/3)),
ceil(qfrac(5/3))\relax{} work!
```

2, 2000, 2000 do the integer division 2, 1667, 1666, 1, 2 work!

NEW WITH 2.0

> Scientific notation is perfectly legitimate if it defines an integer:
>
> ```
> \xintiieval{1e20}
> \xintiieval{100000000e-4}
> ```
>
> 100000000000000000000 10000 No need to use num().
>
> On the other hand the xintfrac "raw" format such as 5/3[0] **is no longer supported** for input in \xintiieval, even as first argument to trunc() or round().

The reduce function is not available and will raise an error. The frac function also. The sqrt function is mapped to \xintiiSqrt which gives a truncated square root. The sqrtr function is mapped to \xintiiSqrtR which gives a rounded square root.

One can use the Float macros if one is careful to use num, or round etc. . . on their output.

```
\xinttheiiexpr \xintFloatSqrt [20]{2},
               \xintFloatSqrt [20]{3}\relax % no operations
```

14142135623730950488[-19], 17320508075688772935[-19] The above went through because no actual operations were carried out. But it is dangerous because the ``printer'' for \xinttheiiexpr could choke on such values. By default however it does nothing.

In the next example there will be an addition. So we firt apply round to get integers (the second argument of round and trunc tells how many digits from after the decimal mark one should keep.)

```
\xinttheiiexpr round(\xintFloatSqrt [20]{2},19) +
               round(\xintFloatSqrt [20]{3},19)\relax
```

31462643699419723423

The whole point of \xintiiexpr is to gain some speed in *integer-only* algorithms, and the above explanations related to how to nevertheless use fractions therein are a bit peripheral. We observed (2013/12/18) of the order of 30% speed gain when dealing with numbers with circa one hundred digits (1.2: this info may be obsolete).

4.9. \xintboolexpr, \xinttheboolexpr

code *code*

x ★ Equivalent to doing \xintexpr ...\relax and returning true if the result does not vanish, and fal

↪se if the result is zero. As \xintexpr, this can be used on comma separated lists of expressions, and even bracketed lists.

It can be customized, one only needs to modify the following:

```
\def\xintboolexprPrintOne#1{\xintiiifNotZero{#1}{true}{talse}}%
```

Not only are true and false usable in input, also True and False are pre-declared variables.

There is quirk in case it is used as a sub-expression: the boolean expression needs at least one logic operation else the value is not standardized to 1 or 0, for example we get from

```
\xinttheexpr \xintboolexpr 1.23\relax\relax\newline
1.23
```

which is to be compared with

```
\xintttheboolexpr 1.23\relax
true
```

4.10. \xintfloatexpr, \xintthefloatexpr

code *code*

x ★ \xintfloatexpr...\relax is a variant of \xintexpr...\relax which does floating point operations.

The target precision for the computation is from the current setting of \xintDigits. Comma separated lists of expressions are allowed.

An optional parameter within brackets [Q] is allowed at the very start of the expression:

- if positive it instructs the macro to round the result to that many digits of precision. It thus makes sense to employ it only if this parameter is less than the \xintttheDigits precision.
- if negative it means to trim off that many digits (of course, in the sense of rounding the values to shorter mantissas). Don't use it to trim all digits (or more than all)!

Since 1.2f all float operations first round their arguments; a parsed number is not rounded prior to its use as operand to such a float operation.

```
\xintDigits:=36\relax
\xintthefloatexpr (1/13+1/121)*(1/179-1/173)/(1/19-1/18)\relax
0.00564487459334466559166166079096852897
\xintthefloatexpr\xintexpr (1/13+1/121)*(1/179-1/173)/(1/19-1/18)\relax\relax
0.00564487459334466559166166079096852912
```

The latter is the rounding of the exact result. The former one has its last three digits wrong due to the cumulative effect of rounding errors in the intermediate computations, as compared to exact evaluations.

I recall here from subsection 9.2 that with release 1.2f the float macros for addition, subtraction, multiplication and division round their arguments first to P significant places with P the asked-for precision of the output; and similarly the power macros and the square root macro. This does not modify anything for computations with arguments having at most P significant places already.

> When defining a macro doing something such as \xintfloatexpr #1\relax, it is recommended to rather use \xintfloatexpr\empty #1\relax, as the #1 may start with a [which without the \empty would be interpreted by \xintfloatexpr as the start of the optional [Q].

4.11. \xinteval, \xintieval, \xintiieval, \xintfloateval

code *code* *code* *code*

x ★ \xinteval is an *f*-expandable macro which is basically defined in such a way that \xinteval{⟨*expression*⟩} behaves like \xintttheexpr{⟨*expression*⟩}\relax. It expands completely in two steps and delivers its output using digits, the dot . as decimal separator, the letter e for scientific notation, the

slash / for fractions, as well as commas in case of multi-items expression and square brackets [and] for nesting.

x ★ \xintieval is similarly related to \xinttheiexpr. It admits an optional argument [D] which may be located in the expected location from conventions of LaTeX2e macros with optional argument, but had been long constrained (until 1.4k) to be inside the braces at the start of the expression.

```
  \xintieval[7]{355/113} = \xintieval{[7]355/113}
3.1415929 = 3.1415929
```

> When defining a macro doing something such as \xintieval{#1}, it is recommended to rather use \xintieval{\empty #1}, as the #1 may start with a [which without the \empty would be interpreted by \xintieval as the start of the optional [D].

x ★ \xintiieval is similarly related to \xinttheiiexpr.

x ★ \xintfloateval is similarly related to \xintthefloatexpr. It admits an optional argument [Q] which may be located either outside (since 1.4k) or inside the braces.

```
  \xintfloateval [7]{355/113} = \xintfloateval{[7] 355/113}
3.141593 = 3.141593
```

When negative, the optional argument tells how many digits to remove from the prevailing precision:

```
  \xintfloateval[-2]{355/113}=
  \xintfloateval{[-2]355/113} has \xinttheDigits\ minus 2 digits.
3.1415929203540= 3.1415929203540 has 16 minus 2 digits.
```

> When defining a macro doing something such as \xintfloateval{#1}, it is recommended to rather use \xintfloateval{\empty #1}, as the #1 may start with a [which without the \empty would be interpreted by \xintfloateval as the start of the optional [Q].

4.12. Using an expression parser within another one

This was already illustrated before. In the following:

```
  \xintfloatexpr \xintexpr add(1/i, i=1234..1243)\relax ^100\relax
5.136088460396579e-210
```

, the inner sum is computed exactly. Then it will be rounded to \xinttheDigits significant digits, and then its power will be evaluated as a float operation. One should avoid the "\xintthe" parsers in inner positions as this induces digit by digit parsing of the inner computation result by the outer parser. Here is the same computation done with floats all the way:

```
  \xintfloatexpr add(1/i, i=1234..1243)^100\relax
5.136088460396643e-210
```

Not surprisingly this differs from the previous one which was exact until raising to the 100th power.

The fact that the inner expression occurs inside a bigger one has nil influence on its behaviour. There is the limitation though that the outputs from \xintexpr and \xintfloatexpr can not be used directly in \xinttheiiexpr integer-only parser. But one can do:

```
  \xintiiexpr round(\xintfloatexpr 3.14^10\relax)\relax % or trunc
93174
```

4.13. The \xintthecoords macro

code

It converts (in two expansion steps) the expansion result of \xintfloatexpr (or \xintexpr or \xintiiexpr) into the (a, b) (c, d) ... format for list of coordinates as expected by the TikZ coordinates syntax.

For use in the following way:

```
coordinates {\xintthecoords\xintfloatexpr ... \relax}
```

or rather perhaps (pay attention with the doubled \relax)

```
coordinates {\xintthecoords\xintiexpr[5]\xintfloatexpr ... \relax\relax}
```

but attention, do NOT use it with \xintthefloatexpr only with \xintfloatexpr.

The syntax with \xintiexpr and a rounding argument [N] immediately following it is to make sure that TikZ's parser only gets some decimal numbers with a N decimal places. Without it, i.e. with only \xintfloatexpr (and possibly some float rounding optional argument [P]), TikZ could receive numbers in scientific notation (but attow I don't recall if that is a problem and I don't have time available to test).

TODO: donner un exemple plus joli.

```
\begin{figure}[htbp]
\centering\begin{tikzpicture}[scale=10]\xintDigits:=8\relax
  \clip (-1.1,-.25) rectangle (.3,.25);
  \draw [blue] (-1.1,0)--(1,0);
  \draw [blue] (0,-1)--(0,+1);
  \draw [red] plot[smooth] coordinates {%
%%% \xintthecoords converts output of next expression into the
%%%            (x1, y1) (x2, y2) ...
%%% format
    \xintthecoords\xintfloatexpr
%%%    This syntax -1+[0..4]/2 is currenty dropped at xint 1.4
%%%    seq((x^2-1,mul(x-t,t=-1+[0..4]/2)),x=-1.2..[0.1]..+1.2)\relax
%%%    Use this:
    seq((x^2-1,mul(x-t,t=seq(-1+u/2, u=0..4))),x=-1.2..[0.1]..+1.2)
    \relax
     };
\end{tikzpicture}
\caption{Coordinates with \csbxint{thecoords}.}
\end{figure}
```

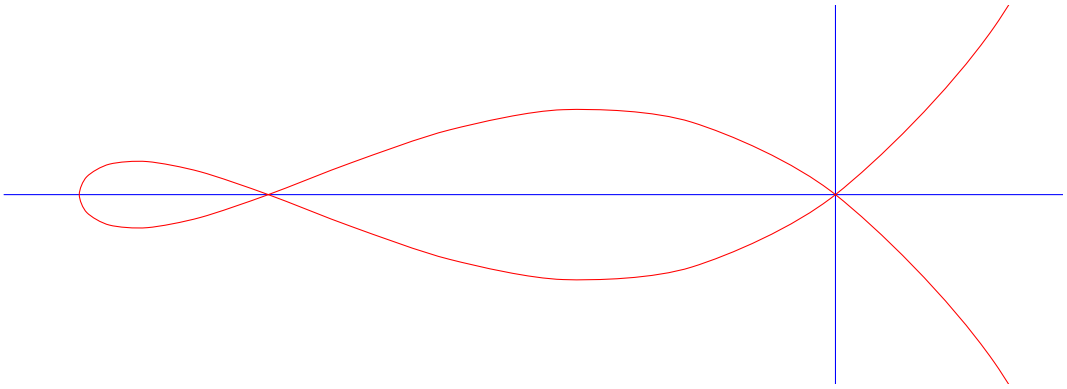


Figure 2: Coordinates with \xintthecoords.

UNSTABLE! It is currently undecided how \xintthecoords should handle bracketed data. Currently, it (or TikZ) will break it the input contains nested structures. One can use it with flat() which removes all nesting. And in combination with zip() it is easy to plot data given by some mechanism in separate lists of x- and y-coordinates (see an example in next section)

4.14. The \xintthespaceseparated macro

code

It converts (in two expansion steps) the expansion result of \xintfloatexpr (or \xintexpr or \xintiiexpr) into the space separated format suitable for usage with PS-Tricks \listplot macro.

Here is for example some syntax (the replacement text of \foo, which is used here only to show that indeed complete expansion is attained in two steps) which can be used as argument to \listplot. Using 4 fractional decimal digits is sufficient when unit is the centimeter (it gives a fixed point precision of one micron, amply enough for plots...).

```
\oodef\foo{%
\xintthespaceseparated
\xintiexpr[4]\xintfloatexpr seq((i, log10(i)), i=1..[0.5]..10)\relax\relax
}\meaning\foo
```

```
macro:->1.0000 0 1.5000 0.1761 2.0000 0.3010 2.5000 0.3979 3.0000 0.4771 3.5000 0.5441 4.0000 0.6021 4.5000 0.6532 5.0000 0.6990 5.5000 0.7404 6.0000 0.7782 6.5000 0.8129 7.0000 0.8451 7.5000 0.8751 8.0000 0.9031 8.5000 0.9294 9.0000 0.9542 9.5000 0.9777 10.0000 1.0000
```

Here we don't really need the inner \xintfloatexpr...\relax because the log10() function works the same in the exact parser \xintexpr but in general this is recommended.

UNSTABLE! It is currently undecided how \xintthespaceseparated should handle bracketed data. Currently, it (or \listplot) will break if the input contains nested structures. One can use it with flat() which removes all nesting. And in combination with zip() it is easy to plot data given by some mechanism in separate lists of x- and y-coordinates.

```
% let's imagine we have something like this
\def\Xcoordinates{1, 3, 5, 7, 9}
\def\Ycoordinates{1, 9, 25, 49, 81}
% then:
|\xintthespaceseparated\xintexpr flat(zip([\Xcoordinates], [\Ycoordinates]))\relax|
is suitable to use as argument to |\listplot|, as it expands to
\xintthespaceseparated\xintexpr flat(zip([\Xcoordinates], [\Ycoordinates]))\relax
```

```
\xintthespaceseparated\xintexpr flat(zip([\Xcoordinates], [\Ycoordinates]))\relax is suitable to use as argument to \listplot, as it expands to 1 1 3 9 5 25 7 49 9 81
```

4.15. \xintifboolexpr, \xintifboolfloatexpr, \xintifbooliiexpr

code *code* *code*

xnn ⋆ \xintifboolexpr{⟨*expr*⟩}{⟨*YES*⟩}{⟨*NO*⟩} does \xinttheexpr<expr>\relax and then executes the ⟨*YES*⟩ or the ⟨*NO*⟩ branch depending on whether the outcome was non-zero or zero. Thus one can read *if bool expr* as meaning *if not zero*:

if ⟨*expr*⟩-ession does not vanish do ⟨*YES*⟩ else do ⟨*NO*⟩

The expression is not limited to using only comparison operators and Boolean logic (<, >, ==, !=, &&, ||, all(), any(), xor(), bool(), togl(), ...), it can be the most general computation.

xnn ⋆ \xintifboolfloatexpr{⟨*expr*⟩}{⟨*YES*⟩}{⟨*NO*⟩} does \xintthefloatexpr⟨*expr*⟩\relax and then executes the ⟨*YES*⟩ or the ⟨*NO*⟩ branch depending on whether the outcome was non zero or zero.

xnn ⋆ \xintifbooliiexpr{⟨*expr*⟩}{⟨*YES*⟩}{⟨*NO*⟩} does \xinttheiiexpr⟨*expr*⟩\relax and then executes the ⟨*YES*⟩ or the ⟨*NO*⟩ branch depending on whether the outcome was non zero or zero.

The expression argument must be a single one, comma separated sub-expressions will cause low-level errors.

4.16. \xintifsgnexpr, \xintifsgnfloatexpr, \xintifsgniiexpr

code *code* *code*

xnnn ⋆ \xintifsgnexpr{⟨*expr*⟩}{⟨*<0*⟩}{⟨*=0*⟩}{⟨*>0*⟩} evaluates the \xintexpression and chooses the branch corresponding to its sign.

xnnn ⋆ \xintifsgnfloatexpr{⟨*expr*⟩}{⟨*<0*⟩}{⟨*=0*⟩}{⟨*>0*⟩} evaluates the \xintfloatexpression and chooses the branch corresponding to its sign.

xnnn ⋆ `\xintifsgniiexpr{⟨expr⟩}{⟨<0⟩}{⟨=0⟩}{⟨>0⟩}` evaluates the `\xintiiexpr`ession and chooses the branch corresponding to its sign.

The expression argument must be a single one, comma separated sub-expressions will cause low-level errors.

4.17. The `\xintNewExpr`, `\xintNewIIExpr`, `\xintNewFloatExpr`, `\xintNewIExpr`, and `\xintNewBoolExpr` macros

code *code* *code* *code* *code*

`\xintNewExpr` macro is used as:

```
\xintNewExpr{\myformula}[n]{⟨stuff⟩}, where
```

- ⟨*stuff*⟩ will be inserted inside `\xinttheexpr . . . \relax`,
- `n` is an integer between zero and nine, inclusive, which is the number of parameters of `\myformula`,
- the placeholders `#1`, `#2`, ..., `#n` are used inside ⟨*stuff*⟩ in their usual rôle,[26] [27]
- the `[n]` is *mandatory*, even for `n=0`.[28]
- the macro `\myformula` is defined without checking if it already exists, LaTeX users might prefer to do first `\newcommand*\myformula {}` to get a reasonable error message in case `\myformula` already exists,
- the protection against active characters is done automatically (as long as the whole thing has not already been fetched as a macro argument and the catcodes correspondingly already frozen).

It (if it succeeds) will be a completely expandable macro entirely built-up using `\xintAdd`, `\xintSub`, `\xintMul`, `\xintDiv`, `\xintPow`, etc. . . as corresponds to the expression written with the infix operators. Macros created by `\xintNewExpr` can thus be nested.

```
    \xintNewFloatExpr \FA [2]{(#1+#2)^10}
    \xintNewFloatExpr \FB [2]{sqrt(#1*#2)}
\begin{enumerate}[nosep]
    \item \FA {5}{5}
    \item \FB {30}{10}
    \item \FA {\FB {30}{10}}{\FB {40}{20}}
\end{enumerate}
```

1. 1e10
2. 17.32050807568877
3. 3.891379490446502e16

The documentation is much shortened here because `\xintNewExpr` and `\xintdeffunc` are very much related one with the other.



ATTENTION!

The original spirit of `\xintNewExpr` was to define a (possibly very big) macro using only xintfrac, and this means in particular that it must be used only with arguments compatible with the xintfrac input format.

Thus an `\xintexpr` declared variable has no chance to work, it must be wrapped explicitly in `\xinteval{...}` to be fetched as argument to a macro constructed by `\xintNewExpr`.

[26] if `\xintNewExpr` is used inside a macro, the `#`'s must be doubled as usual. [27] the `#`'s will in pratice have their usual catcode, but category code other `#`'s are accepted too. [28] there is some use for `\xintNewExpr[0]` compared to an `\edef` as `\xintNewExpr` has some built-in catcode protection.

They share essentially the same limitations.

Notice though that \xintNewFloatExpr accepts and recognizes the optional argument [Q] of \xintfloatexpr, contrarily to \xintdeffloatfunc. Use an \empty in case the contents are not known in advance.

Historical note: prior to 1.4, xintexpr used a \csname..\endcsname encapsulation technique which impacted the string pool memory. The \xintNewExpr was designed as a method to pre-parse the expression and produce one single, gigantic, nested usage of the relevant xintfrac macros. This way, only those macros were expanded which had nil impact on the TEX string pool.

Later on it was found that this mechanism could be employed to define functions. Basically underneath 98% of \xintNewExpr and \xintdeffunc are using the same shared code.

4.18. Analogies and differences of \xintiiexpr with \numexpr

code

\xintiiexpr..\relax is a parser of expressions knowing only (big) integers. There are, besides the enlarged range of allowable inputs, some important differences of syntax between \numexpr and \xintiiexpr and variants:

- Contrarily to \numexpr, the \xintiiexpr parser will stop expanding only after having encountered (and swallowed) a *mandatory* \relax token.
- In particular, spaces between digits (and not only around infix operators or parentheses) do not stop \xintiiexpr, contrarily to the situation with numexpr: \the\numexpr 7 + 3 5\relax expands (in one step)[29] to 105\relax, whereas \xintthe\xintiiexpr 7 + 3 5\relax expands (in two steps) to 42.[30]
- Inside an \edef, an expression \xintiiexpr...\relax get fully evaluated, whereas \numexpr without \the or \number prefix would not, if not itself embedded in another \the\numexpr or similar context.
- (ctd.) The private format to which \xintiiexpr...\relax (et al.) evaluates may use \xintthe prefix to turn into explicit digits, (for example in arguments to some macros which expand their arguments). The \the TEX primitive prefix would not work here.
- (ctd.) One can embed a \numexpr...\relax (with its \relax!) inside an \xintiiexpr...\relax without \the or \number, but the reverse situation requires usage of \xintthe or \xinteval user interface,
- \the\numexpr -(1)\relax is illegal. In contrast \xinttheiiexpr -(1)\relax is perfectly legal and gives the expected result (what else ?).
- \the\numexpr 2+-(1+1)\relax is illegal. In contrast \xinttheiiexpr 2+-(1+1)\relax is legal.
- \the\numexpr 2\cnta\relax is illegal (with \cnta a \count register.) In contrast \xinttheiiexpr 2\cnta\relax is perfectly legal and will do the tacit multiplication.
- \the\numexpr or \number\numexpr expands in one step, but \xintthe\xintiiexpr or \xinttheiiexpr needs two steps.

[29] The \numexpr triggers continued expansion after the space following the 3 to check if some operator like + is upstream. But after having found the 5 it treats it as and end-marker. [30] Since 1.21 one can also use the underscore _ to separate digits for readability of long numbers.

4.19. Chaining expressions for expandable algorithmics

We will see in this section how to chain \xintexpr-essions with \expandafter's, like it is possible with \numexpr. For this it is convenient to use \romannumeral0\xintexpro which is the once-expanded form of \xintexpr, as we can then chain using only one \expandafter each time.

The background of page 2 of this document displays the 1250th Fibonacci number. For its computation the following \FibonacciN macro (or rather a very close variant) was used, which takes as input a positive integer N (we did not add an overhead for N negative or zero). It depends on another macro, called \FibonacciPair which outputs F_N and F_{N-1} one after the other both inside braces.

A technicality is that the Fibonacci numbers are left in a private format which needs \xintthe to resolve to digit tokens. But the output of \FibonacciN can also be used directly in the typesetting phase.

As per the algorithm it is based upon the quadratic relations which allow to go from (F_n, F_{n-1}) to either (F_{2n}, F_{2n-1}) (operation of type A) or to (F_{2n+1}, F_{2n}) (operation of type B). We first convert n into binary base, and read the digits from most significant to least significant applying each time either type A or type B. Each one of type A or B will cost only 3 multiplications and a bunch of additions, which of course are done using \xintiiexpr. In total the algorithm costs (because $3\log_2 10 \approx 10$) about 10L multiplications (doubling is not counted as a multiplication) where L is the number of decimal digits of N (of course the cost of each multiplication increases with the length of operands).

```
\makeatletter
\newcommand\FibonacciPair[1]{%
    \expandafter\Fibo@start
    \romannumeral0\xintdectobin{\the\numexpr#1\relax};%
}
% To chain \xintiiexpr's easily we use the once-expanded form which is
% \romannumeral0\xintiiexpro.  We could also have used \expanded and \xintiieval
% calls, that would have been simpler but old habits die hard.
%
% Gobble the first digit which is 1 (this is where we could have tested for a
% negative or zero input).  Prepare encapsulated 1 and 0.  Then repeatedly
% dispatch according to binary digits (which are strategically located at end
% of input stream to optimize), or the semicolon which signals completion.
%
% We use \csname-governed branching (because it is cool).  We could have
% used some \if test or even an \ifcase with an {-1} at end, or any other
% of the many possibilities opened up by TeX syntax.
\def\Fibo@start 1#1{%
    \csname Fibo@#1\expandafter\endcsname
        \romannumeral0\xintiiexpro 1\expandafter\relax\expandafter;%
        \romannumeral0\xintiiexpro 0\relax;%
}
\@namedef{Fibo@0}#1;#2;#3{%
    \csname Fibo@#3\expandafter\endcsname
       \romannumeral0\xintiiexpro (#1+2*#2)*#1\expandafter\relax\expandafter;%
       \romannumeral0\xintiiexpro #1*#1+#2*#2\relax;%
}
\@namedef{Fibo@1}#1;#2;#3{%
    \csname Fibo@#3\expandafter\endcsname
        \romannumeral0\xintiiexpro 2*(#1+#2)*#1+#2*#2\expandafter\relax\expandafter;%
        \romannumeral0\xintiiexpro (#1+2*#2)*#1\relax;%
}
```

```
% #1 is F_N (and #2 is F_{N-1}) and both are in need of unpacking.
% #1 is F_N (and #2 is F_{N-1}) and both in need of unpacking.
\@namedef{Fibo@;}#1;#2;{{#1}{#2}}

\newcommand\FibonacciN{\expandafter\@firstoftwo\romannumeral0\FibonacciPair}
\makeatother
```

Here a code snippet which checks the routine via a \message of the first 50 Fibonacci numbers (this is not an efficient way to generate a sequence of such numbers, it is only for validating \FibonacciN).

```
\def\Fibo #1.{\xintthe\FibonacciN {#1}}%
\message{\xintiloop [1+1] \expandafter\Fibo\xintiloopindex.,
                          \ifnum\xintiloopindex<49 \repeat \xintthe\FibonacciN{50}.}
```

The way we use \expandafter's to chain successive \xintiiexpro evaluations is exactly analogous to what is possible with \numexpr. The various \romannumeral0\xintiiexpro could very well all have been \xintiiexpr's but then we would have needed \expandafter\expandafter\expandafter each time.

> There is a difference though: \numexpr does *NOT* expand inside an \edef, and to force its expansion we must prefix it with \the or \number or \romannumeral or another \numexpr which is itself prefixed, etc. . . .
>
> But \xintexpr, \xintiexpr, ..., expand fully in an \edef, with the completely expanded result encapsulated in a private format.
>
> Using \xintthe as prefix is necessary to print the result (like \the or \number in the case of \numexpr), but it is not necessary to get the computation done (contrarily to the situation with \numexpr).

Our \Fibonacci expands completely under *f*-expansion, so we can use \fdef rather than \edef in a situation such as

```
\fdef \X {\FibonacciN {100}} ,
```

but it is usually about as efficient to employ \edef. And if we want

```
\edef \Y {(\FibonacciN{100},\FibonacciN{200})} ,
```

then \edef is necessary.

Allright, so let's now give the code[31] to generate {F(N)}{F(N+1)}{F(N+2)}..., using \FibonacciPair for the first two and then using the standard recursion F(N+2)=F(N+1)+F(N):

```
\makeatletter
\newcommand\FibonacciSeq[2]{%#1=starting index, #2>#1=ending index
    \expandafter\Fibonacci@Seq
    \the\numexpr #1\expandafter.\the\numexpr #2.%
}%
% The #1+1 is because \FibonacciPair{N} expands to {F_{N}}{F_{N-1}},
% so here we will have {F_{A+1}}{F_{A}} as starting point.
% We want up to {F_B}.  If B=A+1 there will be nothing to do.
\def\Fibonacci@Seq #1.#2.{%
    \expandafter\Fibonacci@Seq@loop
    \the\numexpr #2-#1-1\expandafter.\romannumeral0\FibonacciPair{#1+1}%
}%
% Now leave in stream one coefficient, test if we have reached B
% and until then apply standard Fibonacci recursion.
\def\Fibonacci@Seq@loop #1.#2#3{% standard Fibonacci recursion
    {#3}\ifnum #1=\z@ \expandafter\Fibonacci@Seq@end\fi
    \expandafter\Fibonacci@Seq@loop
    \the\numexpr #1-1\expandafter.\expandafter
```

NEW WITH 1.4p

[31] Coding for \FibonacciPair and \FibonacciSeq was improved at 1.4p after ten years kept unchanged...

```
        {\romannumeral0\xintiiexpro #2+#3\relax}{#2}%
  }%
  \def\Fibonacci@Seq@end#1\expandafter.\expandafter#2{}%
  \makeatother
```

Again, let's repeat that here the braced values require \xintthe to expand to digit tokens in an \edef (but not for typesetting).

This \FibonacciSeq macro is completely expandable but it is not *f*-expandable.

This is not a problem in the next example which uses \xintFor* as the latter applies repeatedly full expansion to what comes next each time it fetches an item from its list argument. Thus \xintFor* still manages to generate the list via iterated full expansion.

30.	832040	0	**60.**	1548008755920	0	**90.**	2880067194370816120	0
31.	1346269	514229	**61.**	2504730781961	1	**91.**	4660046610375530309	514229
32.	2178309	514229	**62.**	4052739537881	1	**92.**	7540113804746346429	514229
33.	3524578	196418	**63.**	6557470319842	2	**93.**	12200160415121876738	196418
34.	5702887	710647	**64.**	10610209857723	3	**94.**	19740274219868223167	710647
35.	9227465	75025	**65.**	17167680177565	5	**95.**	31940434634990099905	75025
36.	14930352	785672	**66.**	27777890035288	8	**96.**	51680708854858323072	785672
37.	24157817	28657	**67.**	44945570212853	13	**97.**	83621143489848422977	28657
38.	39088169	814329	**68.**	72723460248141	21	**98.**	135301852344706746049	814329
39.	63245986	10946	**69.**	117669030460994	34	**99.**	218922995834555169026	10946
40.	102334155	825275	**70.**	190392490709135	55	**100.**	354224848179261915075	825275
41.	165580141	4181	**71.**	308061521170129	89	**101.**	573147844013817084101	4181
42.	267914296	829456	**72.**	498454011879264	144	**102.**	927372692193078999176	829456
43.	433494437	1597	**73.**	806515533049393	233	**103.**	1500520536206896083277	1597
44.	701408733	831053	**74.**	1304969544928657	377	**104.**	2427893228399975082453	831053
45.	1134903170	610	**75.**	2111485077978050	610	**105.**	3928413764606871165730	610
46.	1836311903	831663	**76.**	3416454622906707	987	**106.**	6356306993006846248183	831663
47.	2971215073	233	**77.**	5527939700884757	1597	**107.**	10284720757613717413913	233
48.	4807526976	831896	**78.**	8944394323791464	2584	**108.**	16641027750620563662096	831896
49.	7778742049	89	**79.**	14472334024676221	4181	**109.**	26925748508234281076009	89
50.	12586269025	831985	**80.**	23416728348467685	6765	**110.**	43566776258854844738105	831985
51.	20365011074	34	**81.**	37889062373143906	10946	**111.**	70492524767089125814114	34
52.	32951280099	832019	**82.**	61305790721611591	17711	**112.**	114059301025943970552219	832019
53.	53316291173	13	**83.**	99194853094755497	28657	**113.**	184551825793033096366333	13
54.	86267571272	832032	**84.**	160500643816367088	46368	**114.**	298611126818977066918552	832032
55.	139583862445	5	**85.**	259695496911122585	75025	**115.**	483162952612010163284885	5
56.	225851433717	832037	**86.**	420196140727489673	121393	**116.**	781774079430987230203437	832037
57.	365435296162	2	**87.**	679891637638612258	196418	**117.**	1264937032042997393488322	2
58.	591286729879	832039	**88.**	1100087778366101931	317811	**118.**	2046711111473984623691759	832039
59.	956722026041	1	**89.**	1779979416004714189	514229	**119.**	3311648143516982017180081	1

Some Fibonacci numbers together with their residues modulo F(30)=832040

```
  \newcounter{myindex}% not "index", which would overwrite theindex environment!
  % (many have probably been bitten by this trap)
  \tabskip 1ex
    \fdef\Fibxxx{\FibonacciN {30}}%
    \setcounter{myindex}{30}%
  \vbox{\halign{\bfseries#.\hfil&#\hfil &\hfil #\cr
    \xintFor* #1 in {\FibonacciSeq {30}{59}}\do
    {\themyindex &\xintthe#1 &
      \xintiiRem{\xintthe#1}{\xintthe\Fibxxx}\stepcounter{myindex}\cr }}%
  }\vrule
```

```
\vbox{\halign{\bfseries#.\hfil&#\hfil &\hfil #\cr
  \xintFor* #1 in {\FibonacciSeq {60}{89}}\do
  {\themyindex &\xintthe#1 &
    \xintiiRem{\xintthe#1}{\xintthe\Fibxxx}\stepcounter{myindex}\cr }}%
}\vrule
\vbox{\halign{\bfseries#.\hfil&#\hfil &\hfil #\cr
  \xintFor* #1 in {\FibonacciSeq {90}{119}}\do
  {\themyindex &\xintthe#1 &
   \xintiiRem{\xintthe#1}{\xintthe\Fibxxx}\stepcounter{myindex}\cr }}%
}%
```

This produces the Fibonacci numbers from F(30) to F(119), and computes also all the congruence classes modulo F(30). The output has been put in a float, which appears on the preceding page. I leave to the mathematically inclined readers the task to explain the visible patterns...;-).

4.20. When expandability is too much

Let's use the macros of subsection 4.19 related to Fibonacci numbers. Notice that the 47th Fibonacci number is 2971215073 thus already too big for TEX and ε-TEX.

The \FibonacciN macro found in subsection 4.19 is completely expandable, it is even *f*-expandable. We need a wrapper with \xintthe prefix

```
\def\theFibonacciN{\xintthe\FibonacciN}
```

to print in the document or to use within \message (or LaTeX typeout) to write to the log and terminal.

The \xintthe prefix also allows its use it as argument to the xint macros: for example if we are interested in knowing how many digits F(1250) has, it suffices to issue \xintLen {\theFibonacciN {1250}} (which expands to 261). Or if we want to check the formula gcd(F(1859), F(1573)) = F(gcd(1859, 1573)) = F(143), we only need[32]

```
$\xintiiGCD{\theFibonacciN{1859}}{\theFibonacciN{1573}}=%
 \theFibonacciN{\xintiiGCD{1859}{1573}}$
```

which produces:

```
34335830278418729487027505833 7 = 34335830278418729487027505833 7
```

The \theFibonacciN macro expanded its \xintiiGCD{1859}{1573} argument via the services of \numexpr: this step allows only things obeying the TEX bound, naturally! (but F(2147483648) would be rather big anyhow...).

This is very convenient but of course it repeats the complete evaluation each time it is done. In practice, it is often useful to store the result of such evaluations in macros. Any \edef will break expandability, but if the goal is at some point to print something to the dvi or pdf output, and not only to the log file, then expandability has to be broken one day or another!

Hence, in practice, if we want to print in the document some computation results, we can proceed like this and avoid having to repeat identical evaluations:

```
\begingroup
  \def\A {1859}  \def\B {1573}
  \edef\X {\theFibonacciN\A}  \edef\Y {\theFibonacciN\B}
  \edef\GCDAB {\xintiiGCD\A\B}\edef\Z {\theFibonacciN\GCDAB}
  \edef\GCDXY{\xintiiGCD\X\Y}
The identity $\gcd(F(\A),F(\B))=F(\gcd(\A,\B))$ can be checked via evaluation
of both sides: $\gcd(F(\A),F(\B))=\gcd(\printnumber\X,\printnumber\Y)=
\printnumber{\GCDXY} = F(\gcd(\A,\B)) = F(\GCDAB) =\printnumber\Z$.\par
          % some further computations involving \A, \B, \X, \Y
\endgroup % closing the group removes assignments to \A, \B, ...
% or choose longer names less susceptible to overwrite something.
% Note: there is no LaTeX \newecommand which would be to \edef like
```

[32] The \xintiiGCD macro is provided by both the xintgcd package (since 1.0) and by the xint package (since 1.3d).

```
  % \newcommand is to \def
The identity gcd(F(1859),F(1573)) = F(gcd(1859,1573)) can be checked via evaluation of both
sides: gcd(F(1859),F(1573)) = gcd(1440582791304425119877168915150404286991316149502348101422
↪ 6686367010882725975754947224824377535296194597948692273576288822163093580182640808517753199 7
↪ 4256956055294350288615852451737250886736422228492908228952455838894954421926557604129992902 5
↪ 5659797113378761054522176234908415299798114131996600875176897034109975200799936107075760195 2
↪ 0876324584695551467505894985013610208598628752325727241,2443841925195118573328279459777626197
↪ 9853990248157061923260536090078401339403674321244522327895990951586958110318917797690580327 4
↪ 1516325953076166866610137252008667540965698889510100228880168314593473101315665177215932493 4
↪ 4798634399479371195758766544765827958909282390070313197135548122004938644531329524847747273 1
↪ 66471511289078393) = 34335830278418729487027505833 7 = F(gcd(1859,1573)) = F(143) = 343358302784
↪ 187294870275058337.
```

One may legitimately ask the author: why expandability to such extremes, for things such as big fractions or floating point numbers (even continued fractions...) which anyhow can not be used directly within TEX's primitives such as \ifnum? Why insist on a concept which is foreign to the vast majority of TEX users and even programmers?

I have no answer: it made definitely sense at the start of xint (see subsection 9.13) and once started I could not stop.

4.21. Acknowledgements (2013/05/25)

I was greatly helped in my preparatory thinking, prior to producing such an expandable parser, by the commented source of the l3fp package, specifically the l3fp-parse.dtx file (in the version of April-May 2013; I think there was in particular a text called ``roadmap'' which was helpful). Also the source of the calc package was instructive, despite the fact that here for \xintexpr the principles are necessarily different due to the aim of achieving expandability.

5. The calxtrig.tex library

.1 Introduction 89
.2 \xintreloadcalxtrig, \xintreloadxinttrig 89
.3 Constants 89
.4 Functions 90
.5 Important implementation notes 91
.6 Some example evaluations 91

5.1. Introduction

This library (not to be loaded directly by user) provides trigonometric functions for use within \xintfloateval (and \xinteval), up to a maximal accuracy of 100 floating point digits.

NEW WITH 2.0a1

The first version supporting trigonometry was release 1.3e (2019/04/05) of xintexpr. It was completely rewritten at 2.0a1. It now supports precisions up to 100, is faster and more accurate. It defines no user level macro.

For the scientific-library precision set to a multiple of four, the accuracy targets of 2.0a1 apply. At other precisions the additional rounding can cause double rounding.

At a library precision of 8 (or less), the target error bound before rounding is about ten times larger. Assuming independent uniform distributions of the signed error and of the position of the exact result relative to rounding ties, the estimated percentages of correct rounding are 97.5% at 8 digits and 99.75% at higher precisions divisible by four. In both cases the target after rounding is an error of at most 1ulp. The trigonometric tests indicate a higher percentage of correct rounding.

Acknowledgements: I finally decided to release some such functions under friendly pressure of Jürgen Gilg and Thomas Ossig, let them both be thanked here.

Jürgen passed away in 2022. I will miss our friendship which was born and grew from numerous and regular exchanges on topics not limited to this package or even the TEX world. Let's now continue to "take care and keep motivated"!

5.2. \xintreloadcalxtrig, \xintreloadxinttrig

code *code*

The second name is retained for backward compatibility.

Sets the precision of the trigonometric library to the current \xinttheDigits, reconstructing its tables when necessary, and re-assigns the trigonometric constants below. The precision of the logarithmic library is unchanged.

5.3. Constants

They are left free to use, or modified, or \xintunassignvar'd, as this will have no impact whatsoever on the functions. They are provided as backward compatible convenience.

twoPi what could that be?

threePiover2

Pi

Piover2

oneRadian this is one radian in degrees: 180/π

oneDegree this is one degree in radian: π/180

5.4. Functions

5.4.1. Direct trigonometry

With the variable in radians:

sin(x) sine

cos(x) cosine

tan(x) tangent

tg(x) tangent

cot(x) cotangent

cotg(x) cotangent

sec(x) secant

csc(x) cosecant

sinc(x) sinc(x) = sin(x)/x

Some have variants handling an input in degrees:

sind(x) sine

cosd(x) cosine

tand(x) tangent

cotd(x) cotangent

secd(x) secant

cscd(x) cosecant

5.4.2. Inverse trigonometry

With the value in radians:

asin(x) arcsine

arcsin(x) arcsine

acsc(x) arccosecant

arccsc(x) arccosecant

acos(x) arccosine

arccos(x) arccosine

asec(x) arcsecant

arcsec(x) arcsecant

atan(x) arctangent

arctg(x) arctangent

Arg(x, y) the main branch of the argument of the complex number x+iy, from -π (excluded) to π (included). As the output is rounded -Pi (its value) is a possible return value.

pArg(x, y) the branch of the argument of the complex number x+iy with values going from 0 (included) to 2π (excluded). Inherent rounding makes twoPi (value) a possible return value.

atan2(y, x) is Arg(x, y). Note the reversal of the arguments, this seems to be the most frequently encountered convention across languages.

With the value in degrees:

asind(x) arcsine

acosd(x) arccosine

atand(x) arctangent

Argd(x, y) the main branch of the argument of the complex number x+iy, from -180 (excluded) to 180 (included). Inherent rounding of output can cause -180 to be returned.

pArgd(x, y) the branch of the argument of the complex number x+iy with values going from 0 (included) to 360 (excluded). Inherent rounding of output can cause 360 to be returned.

atan2d(y, x) is Argd(x, y). Note the reversal of the arguments, this seems to be the most frequently encountered convention across languages.

5.4.3. Conversion functions (optional definitions left to user decision)

Python provides functions degrees() and radians(). But as most of the calx functions are already defined for the two units, I felt this was not really needed. It is a oneliner to add them:

```
\xintdefloatfunc radians(x) := x * oneDegree;
\xintdefloatfunc degrees(x) := x * oneRadian;
```

5.5. Important implementation notes

- Floating point numbers with large exponents are integers and are multiples of 1000; hence modulo 360 all such ``angles'' are integer multiples of 280 degrees, i.e. are modulo 360 among the integer multiples of 40 degrees, which means that there are *only 10 such angles*. Needless to say, considering usage of the sind() and cosd() functions with such large floating point numbers is **meaningless**, or at least **illusory** (contrarily to say compute a square, a square-root, or a multiplication). This is the author's opinion, and you will have a very hard time making me change my mind. A floating point number represents a range of real numbers, and large floating point numbers represent thus each a range encompassing multiple consecutive batches of 360 degrees, there is no way you can seriously think that one can give a meaning to the sind() or cosd() then apart from their averabe value which is zero (same for sin() and cos()).

5.6. Some example evaluations

```
\xintDigits* := 48\relax
Digits at \xinttheDigits:\newline
$sind(17)\approx\xintfloateval{sind(17)}$\newline
$cosd(17)\approx\xintfloateval{cosd(17)}$\newline
$tand(17)\approx\xintfloateval{tand(17)}$\newline
$sind(43)\approx\xintfloateval{sind(43)}$\newline
```

```
  $cosd(43)\approx\xintfloateval{cosd(43)}$\newline
  $tand(43)\approx\xintfloateval{tand(43)}$\newline
  $asind(0.3)\approx\xintfloateval{asind(0.3)}$\newline
  $acosd(0.3)\approx\xintfloateval{acosd(0.3)}$\newline
  $atand(3)\approx\xintfloateval{atand(3)}$\newline
  $tan(atan(7))\approx\xintfloateval{tan(atan(7))}$\newline
  $asind(sind(25))\approx\xintfloateval{asind(sind(25))}$\par\medskip
  \noindent\xintDigits* := 24\relax
  Digits at \xinttheDigits:\newline
  $sind(17)\approx\xintfloateval{sind(17)}$\newline
  $cosd(17)\approx\xintfloateval{cosd(17)}$\newline
  $tand(17)\approx\xintfloateval{tand(17)}$\newline
  $sind(43)\approx\xintfloateval{sind(43)}$\newline
  $cosd(43)\approx\xintfloateval{cosd(43)}$\newline
  $tand(43)\approx\xintfloateval{tand(43)}$\newline
  $asind(0.3)\approx\xintfloateval{asind(0.3)}$\newline
  $acosd(0.3)\approx\xintfloateval{acosd(0.3)}$\newline
  $atand(3)\approx\xintfloateval{atand(3)}$\newline
  $tan(atan(7))\approx\xintfloateval{tan(atan(7))}$\newline
  $asind(sind(25))\approx\xintfloateval{asind(sind(25))}$\par
  \xintDigits* := 16\relax
```

Digits at 48:
sind(17) ≈ 0.292371704722736728097468695377143252664687186183
cosd(17) ≈ 0.956304755963035481338650816618418962009410343991
tand(17) ≈ 0.305730681458660355734541958996550716146250221387
sind(43) ≈ 0.681998360062498500442225784711125580340433802762
cosd(43) ≈ 0.731353701619170483287543608275622403378396544763
tand(43) ≈ 0.932515086137661705612185627426186654353537299494
asind(0.3) ≈ 17.457603123722092290246045792444944183216636440147
acosd(0.3) ≈ 72.542396876277907709753954207555058178336355985 3
atand(3) ≈ 71.565051177077989351572193720453294671204214299 6
tan(atan(7)) ≈ 7.000000000000000000000000000000000000000000000000005
asind(sind(25)) ≈ 25

Digits at 24:
sind(17) ≈ 0.29237170472273672809746
cosd(17) ≈ 0.95630475596303548133865
tand(17) ≈ 0.30573068145866035573454
sind(43) ≈ 0.68199836006249850044222
cosd(43) ≈ 0.73135370161917048328754
tand(43) ≈ 0.93251508613766170561218
asind(0.3) ≈ 17.4576031237220922902460
acosd(0.3) ≈ 72.5423968762779077097540
atand(3) ≈ 71.5650511770779893515722
tan(atan(7)) ≈ 7
asind(sind(25)) ≈ 25

6. The calxlog.tex library

.1 Introduction 93
.2 \xintreloadcalxlog, \xintreloadxintlog 93
.3 Functions 93

6.1. Introduction

This package provides logarithms, exponentials and fractional powers for use within \xintfloateval (and \xinteval).

The first release of xintexpr supporting logarithms and exponentials was 1.3e (2019/04/05). It is automatically loaded by xintexpr. Since 2.0a1 2026/11/01 the maximal supported precision is of 100 digits.

The ln() alias, expm1(), log1p(), hyperbolic functions, and their inverses were added at release 1.4p (March 2026).

For the scientific-library precision set to a multiple of four, the accuracy targets of 2.0a1 apply. At other precisions the additional rounding can cause double rounding.

At a library precision of 8 (or less), the target error bound before rounding is about ten times larger. Assuming independent uniform distributions of the signed error and of the position of the exact result relative to rounding ties, the estimated percentages of correct rounding are 97.5% at 8 digits and 99.75% at higher precisions divisible by four. In both cases the target after rounding is an error of at most 1ulp.

6.2. \xintreloadcalxlog, \xintreloadxintlog

code *code*

The second name is retained for backward compatibility.

Sets the precision of the logarithmic and exponential library to the current \xinttheDigits, reconstructing its tables when necessary. The precision of the trigonometric library is unchanged.

6.3. Functions

log10(x) logarithm in base 10

pow10(x) fractional powers of 10

log(x) natural logarithm

ln(x) natural logarithm

log1p(x) computes log(1+x)

exp(x) exponential function

expm1(x) computes e^x-1

pow(x, y) computes x^y. In \xintfloateval, exponents recognized as integers in the \numexpr range normally use binary exponentiation at arithmetic precision (see \xintFloatPower). Otherwise the logarithmic route computes exp(y*log(x)). For this route the exponent y is not rounded to the target floating point precision beforehand, but to this target precision plus 14. It does not have to be, if integer, at most the TEX bound for integers (which is 2147483647). But the output must have a decimal exponent obeying this bound (with some safety margin)

```
\calxeval{3^3350214776, 2^5309964787}\newline
\calxeval{3^3350214776/2^5309964787}
```

```
3.202379781557598e1598458677, 6.404759563014461e1598458676
5.000000000078641
```

Inside the \xinteval evaluator, and if the exponent is integer, and if it is anticipated that the output will not exceed (or will only slightly exceed) 10000 digits (separately for numerator and denominator), then the power is computed *exactly*. To compute exactly higher powers than 2^9999 or 9^9999 or 99^5000 or 999^3333, etc..., use \xintiieval. See \xintiiPow for related comments if you don't want to melt your CPU.

sinh(x) hyperbolic sine

sh(x) hyperbolic sine

cosh(x) hyperbolic cosine

ch(x) hyperbolic cosine

tanh(x) hyperbolic tangent

th(x) hyperbolic tangent

coth(x) hyperbolic cotangent

sech(x) hyperbolic secant

csch(x) hyperbolic cosecant

asinh(x) inverse hyperbolic sine

argsh(x) inverse hyperbolic sine

acosh(x) inverse hyperbolic cosine

argch(x) inverse hyperbolic cosine

atanh(x) inverse hyperbolic tangent

argth(x) inverse hyperbolic tangent

acoth(x) inverse hyperbolic cotangent

argcoth(x) inverse hyperbolic cotangent

asech(x) inverse hyperbolic secant

argsech(x) inverse hyperbolic secant

acsch(x) inverse hyperbolic cosecant

argcsch(x) inverse hyperbolic cosecant

```
\xintDigits*:=24\relax
\noindent\xintfloateval{ln(2), exp(1)}\newline
\xintfloateval{2^(1/3), 2^10000}\newline
\xintfloateval{sinh(1), cosh(1), tanh(1)}\newline
\xintfloateval{sh(1), ch(1), th(1)}
```

```
0.693147180559945309417232, 2.718281828459045235360029
1.259921049894873164767210, 1.995063116880758384883743010
1.175201193643801456882382, 1.543080634815243778477906, 0.761594155955764888119458
1.175201193643801456882382, 1.543080634815243778477906, 0.761594155955764888119458
```

7. Macros of the xinttools package

.1 Introduction 95
.2 \xintRevWithBraces 95
.3 \xintZapFirstSpaces, \xintZapLastSpaces, \xintZapSpaces, \xintZapSpacesB 96
.4 \xintCSVtoList 96
.5 \xintNthElt 98
.6 \xintNthOnePy 98
.7 \xintKeep 98
.8 \xintKeepUnbraced 99
.9 \xintTrim 99
.10 \xintTrimUnbraced 100
.11 \xintListWithSep 100
.12 \xintApply 100
.13 \xintApplyUnbraced 101
.14 \xintSeq 101
.15 \xintloop, \xintbreakloop, \xintbreakloopanddo, \xintloopskiptonext 102
.16 \xintiloop, \xintiloopindex, \xintouteriloopindex, \xintbreakiloop, \xintbreakiloopanddo, \xintiloopskiptonext, \xintiloopskipandredo 104
.17 \xintApplyInline 107
.18 \xintFor, \xintFor* 108
.19 \xintifForFirst, \xintifForLast 111
.20 \xintBreakFor, \xintBreakForAndDo 111
.21 \xintintegers, \xintdimensions, \xintrationals 111
.22 \xintForpair, \xintForthree, \xintForfour 113
.23 \xintAssign 113
.24 \xintAssignArray 114
.25 \xintDigitsOf 115
.26 \xintRelaxArray 115

7.1. Introduction

These utilities used to be provided within the xint package; since 1.09g (2013/11/22) they have been moved to an independently usable package xinttools, which has none of the xint facilities regarding big numbers. Whenever relevant release 1.09h has made the macros \long so they accept \par tokens on input.

The completely expandable utilities (up to \xintiloop) are documented first, then the non expandable utilities.

section 8 gives additional (some quite dated) examples of use of macros of this package.

xinttools is automatically loaded by xintexpr.

7.2. \xintRevWithBraces

code

f ⋆ \xintRevWithBraces{⟨*list*⟩} first does the *f*-expansion of its argument then it reverses the order of the tokens, or braced material, it encounters, maintaining existing braces and adding a brace pair around each naked token encountered. Space tokens (in-between top level braces or naked tokens) are gobbled. This macro is mainly thought out for use on a ⟨*list*⟩ of such braced material; with such a list as argument the *f*-expansion will only hit against the first opening brace, hence do nothing, and the braced stuff may thus be macros one does not want to expand.

```
\edef\x{\xintRevWithBraces{12345}}
\meaning\x:macro:->{5}{4}{3}{2}{1}
\edef\y{\xintRevWithBraces\x}
\meaning\y:macro:->{1}{2}{3}{4}{5}
```

The examples above could be defined with \edef's because the braced material did not contain macros. Alternatively:

```
\expandafter\def\expandafter\w\expandafter
{\romannumeral0\xintrevwithbraces{{\A}{\B}{\C}{\D}{\E}}}
\meaning\w:macro:->{\E }{\D }{\C }{\B }{\A }
```

n ⋆ The macro \xintReverseWithBracesNoExpand does the same job without the initial expansion of its argument.

7.3. \xintZapFirstSpaces, \xintZapLastSpaces, \xintZapSpaces, \xintZapSpacesB

code code code code

n ★ \xintZapFirstSpaces{⟨*stuff*⟩} does not do *any* expansion of its argument, nor brace removal of any sort, nor does it alter ⟨*stuff*⟩ in anyway apart from stripping away all *leading* spaces.

This macro will be mostly of interest to programmers who will know what I will now be talking about. *The essential points, naturally, are the complete expandability and the fact that no brace removal nor any other alteration is done to the input.*

TEX's input scanner already converts consecutive blanks into single space tokens, but \xintZapFirstSpaces handles successfully also inputs with consecutive multiple space tokens. However, it is assumed that ⟨*stuff*⟩ does not contain (except inside braced sub-material) space tokens of character code distinct from 32.

It expands in two steps, and if the goal is to apply it to the expansion text of \x to define \y, then one can do: \odef\y{\romannumeral0\expandafter\xintzapfirstspaces\expandafter{\x}} (one can also define a wrapper macro to \xintZapFirstSpaces in order to expand once the argument first, but xinttools not being a programming layer, it provides no "Generate Variants" facilities).

Other use case: inside a macro which received a parameter #1, one can do \oodef\x{\xintZapFirs
↪ tSpaces {#1}}, or, if #1, after leading spaces have been stripped can accept \edef expansion, one can do \edef\x{\xintZapFirstSpaces{#1}}.

```
\xintZapFirstSpaces { \a { \X } { \b \Y } }->\a { \X } { \b \Y } +++
```

n ★ \xintZapLastSpaces{⟨*stuff*⟩} does not do *any* expansion of its argument, nor brace removal of any sort, nor does it alter ⟨*stuff*⟩ in anyway apart from stripping away all *ending* spaces. The same remarks as for \xintZapFirstSpaces apply.

```
\xintZapLastSpaces { \a { \X } { \b \Y } }-> \a { \X } { \b \Y }+++
```

n ★ \xintZapSpaces{⟨*stuff*⟩} does not do *any* expansion of its argument, nor brace removal of any sort, nor does it alter ⟨*stuff*⟩ in anyway apart from stripping away all *leading* and all *ending* spaces. The same remarks as for \xintZapFirstSpaces apply.

```
\xintZapSpaces { \a { \X } { \b \Y } }->\a { \X } { \b \Y }+++
```

n ★ \xintZapSpacesB{⟨*stuff*⟩} does not do *any* expansion of its argument, nor does it alter ⟨*stuff*⟩ in anyway apart from stripping away all leading and all ending spaces and possibly removing one level of braces if ⟨*stuff*⟩ had the shape <spaces>{braced}<spaces>. The same remarks as for \xintZapFirstSpaces apply.

```
\xintZapSpacesB { \a { \X } { \b \Y } }->\a { \X } { \b \Y }+++
\xintZapSpacesB { { \a { \X } { \b \Y } } }-> \a { \X } { \b \Y } +++
```

The spaces here at the start and end of the output come from the braced material, and are not removed (one would need a second application for that; recall though that the xint zapping macros do not expand their argument).

7.4. \xintCSVtoList

code

f ★ \xintCSVtoList{a,b,c...,z} returns {a}{b}{c}...{z}. A *list* is by convention in this manual simply a succession of tokens, where each braced thing will count as one item (``items'' are defined according to the rules of TEX for fetching undelimited parameters of a macro, which are exactly the same rules as for LATEX and macro arguments [they are the same things]). The word `list' in `comma separated list of items' has its usual linguistic meaning, and then an ``item'' is what is delimited by commas.

So \xintCSVtoList takes on input a `comma separated list of items' and converts it into a `TEX list of braced items'. The argument to \xintCSVtoList may be a macro: it will first be *f*-expanded. Hence the item before the first comma, if it is itself a macro, will be expanded which may or may not be a good thing. A space inserted at the start of the first item serves to stop that expansion (and
n ★ disappears). The macro \xintCSVtoListNoExpand does the same job without the initial expansion of the list argument.

Apart from that no expansion of the items is done and the list items may thus be completely arbitrary (and even contain perilous stuff such as unmatched \if and \fi tokens).

Contiguous spaces and tab characters, are collapsed by TEX into single spaces. All such spaces around commas[33] are removed, as well as the spaces at the start and the spaces at the end of the list.[34] The items may contain explicit \par's or empty lines (converted by the TEX input parsing into \par tokens).

```
\xintCSVtoList { 1 ,{ 2 , 3 , 4 , 5 }, a , {b,T} U , { c , d } , { {x , y} } }
->{1}{2 , 3 , 4 , 5}{a}{{b,T} U}{ c , d }{ {x , y} }
```

One sees on this example how braces protect commas from sub-lists to be perceived as delimiters of the top list. Braces around an entire item are removed, even when surrounded by spaces before and/or after. Braces for sub-parts of an item are not removed.

We observe also that there is a slight difference regarding the brace stripping of an item: if the braces were not surrounded by spaces, also the initial and final (but no other) spaces of the *enclosed* material are removed. This is the only situation where spaces protected by braces are nevertheless removed.

From the rules above: for an empty argument (only spaces, no braces, no comma) the output is {} (a list with one empty item), for ``<opt. spaces>{}<opt. spaces>'' the output is {} (again a list with one empty item, the braces were removed), for ``{ }'' the output is {} (again a list with one empty item, the braces were removed and then the inner space was removed), for `` { }'' the output is {} (again a list with one empty item, the initial space served only to stop the expansion, so this was like ``{ }'' as input, the braces were removed and the inner space was stripped), for `` { } '' the output is { } (this time the ending space of the first item meant that after brace removal the inner spaces were kept; recall though that TEX collapses on input consecutive blanks into one space token), for ``,'' the output consists of two consecutive empty items {}{}. Recall that on output everything is braced, a {} is an ``empty'' item. Most of the above is mainly irrelevant for every day use, apart perhaps from the fact to be noted that an empty input does not give an empty output but a one-empty-item list (it is as if an ending comma was always added at the end of the input).

```
\def\y{ \a,\b,\c,\d,\e} \xintCSVtoList\y->{\a }{\b }{\c }{\d }{\e }
\def\t {{\if},\ifnum,\ifx,\ifdim,\ifcat,\ifmmode}
\xintCSVtoList\t->{\if }{\ifnum }{\ifx }{\ifdim }{\ifcat }{\ifmmode }
```

The results above were automatically displayed using TEX's primitive \meaning, which adds a space after each control sequence name. These spaces are not in the actual braced items of the produced lists. The first items \a and \if were either preceded by a space or braced to prevent expansion. The macro \xintCSVtoListNoExpand would have done the same job without the initial expansion of the list argument, hence no need for such protection but if \y is defined as \def\y{\a,\b,\c,\d,\e} we then must do:

```
\expandafter\xintCSVtoListNoExpand\expandafter {\y}
```

Else, we may have direct use:

```
\xintCSVtoListNoExpand {\if,\ifnum,\ifx,\ifdim,\ifcat,\ifmmode}
->{\if }{\ifnum }{\ifx }{\ifdim }{\ifcat }{\ifmmode }
```

Again these spaces are an artefact from the use in the source of the document of \meaning (or rather here, \detokenize) to display the result of using \xintCSVtoListNoExpand (which is done for real in this document source).

f ★ *n* ★ For the similar conversion from comma separated list to braced items list, but without removal of spaces around the commas, there is \xintCSVtoListNonStripped and \xintCSVtoListNonStrippedNoExpand.

[33] and multiple space tokens are not a problem; but those at the top level (not hidden inside braces) *must* be of character code 32. [34] let us recall that this is all done completely expandably... There is absolutely no alteration of any sort of the item apart from the stripping of initial and final space tokens (of character code 32) and brace removal if and only if the item apart from intial and final spaces (or more generally multiple char 32 space tokens) is braced.

code
7.5. \xintNthElt

num x f ★ \xintNthElt{x}{⟨list⟩} gets (expandably) the xth item of the ⟨list⟩. A braced item will lose one level of brace pairs. The token list is first f-expanded.

Items are counted starting at one.

```
\xintNthElt {3}{{agh}\u{zzz}\v{Z}} is zzz
\xintNthElt {3}{{agh}\u{{zzz}}\v{Z}} is {zzz}
\xintNthElt {2}{{agh}\u{{zzz}}\v{Z}} is \u
\xintNthElt {37}{\xintiiFac {100}}=9 is the thirty-seventh digit of 100!.
\xintNthElt {10}{\xintFtoCv {566827/208524}}=1457/536
is the tenth convergent of 566827/208524 (uses xintcfrac package).
\xintNthElt {7}{\xintCSVtoList {1,2,3,4,5,6,7,8,9}}=7
\xintNthElt {0}{\xintCSVtoList {1,2,3,4,5,6,7,8,9}}=9
\xintNthElt {-3}{\xintCSVtoList {1,2,3,4,5,6,7,8,9}}=7
```

If x=0, the macro returns the *length* of the expanded list: this is not equivalent to \xintLength which does no pre-expansion. And it is different from \xintLen which is to be used only on integers or fractions.

If x<0, the macro returns the |x|th element from the end of the list. Thus for example x=-1 will fetch the last item of the list.

```
\xintNthElt {-5}{{{agh}}\u{zzz}\v{Z}} is {agh}
```

num x n ★ The macro \xintNthEltNoExpand does the same job but without first expanding the list argument: \xintNthEltNoExpand {-4}{\u\v\w T\x\y\z} is T.

If x is strictly larger (in absolute value) than the length of the list then \xintNthElt produces empty contents.

code
7.6. \xintNthOnePy

num x f ★ \xintNthOnePy{x}{⟨list⟩} gets (expandably) the xth item of the ⟨list⟩, adding a brace pair if there wasn't one.

Attention, items are counted starting at zero. For negative index, behaves as \xintNthElt.

If the index is out of range, the empty output is returned. If the input list was empty (had no items) the empty output is returned.

code
7.7. \xintKeep

num x f ★ \xintKeep{x}{⟨list⟩} expands the token list argument L and produces a new list, depending on the value of x:

- if x>0, the new list contains the first x items from L (counting starts at one.) *Each such item will be output within a brace pair.* Use \xintKeepUnbraced if this is not desired. This means that if the list item was braced to start with, there is no modification, but if it was a token without braces, then it acquires them.
- if x>=length(L), the new list is the old one with all its items now braced.
- if x=0 the empty list is returned.
- if x<0 the last |x| elements compose the output in the same order as in the initial list; as the macro proceeds by removing head items the kept items end up in output as they were in input: no added braces.
- if x<=-length(L) the output is identical with the input.

\xintKeepNoExpand does the same without first f-expanding its list argument.

```
\fdef\test {\xintKeep {17}{\xintKeep {-69}{\xintSeq {1}{100}}}}\meaning\test\par
\noindent\fdef\test {\xintKeep {7}{{1}{2}{3}{4}{5}{6}{7}{8}{9}}}\meaning\test\par
\noindent\fdef\test {\xintKeep {-7}{{1}{2}{3}{4}{5}{6}{7}{8}{9}}}\meaning\test\par
\noindent\fdef\test {\xintKeep {7}{123456789}}\meaning\test\par
```

```
  \noindent\fdef\test {\xintKeep {-7}{123456789}}\meaning\test\par
macro:->{32}{33}{34}{35}{36}{37}{38}{39}{40}{41}{42}{43}{44}{45}{46}{47}{48}
macro:->{1}{2}{3}{4}{5}{6}{7}
macro:->{3}{4}{5}{6}{7}{8}{9}
macro:->{1}{2}{3}{4}{5}{6}{7}
macro:->3456789
```

7.8. \xintKeepUnbraced

code

Same as \xintKeep but no brace pairs are added around the kept items from the head of the list in the case x>0: each such item will lose one level of braces. Thus, to remove braces from all items of the list, one can use \xintKeepUnbraced with its first argument larger than the length of the list; the same is obtained from \xintListWithSep{}{⟨*list*⟩}. But the new list will then have generally many more items than the original ones, corresponding to the unbraced original items.

For x<0 the macro is no different from \xintKeep. Hence the name is a bit misleading because brace removal will happen only if x>0.

\xintKeepUnbracedNoExpand does the same without first *f*-expanding its list argument.

```
  \fdef\test {\xintKeepUnbraced {10}{\xintSeq {1}{100}}}\meaning\test\par
  \noindent\fdef\test {\xintKeepUnbraced {7}{{1}{2}{3}{4}{5}{6}{7}{8}{9}}}%
           \meaning\test\par
  \noindent\fdef\test {\xintKeepUnbraced {-7}{{1}{2}{3}{4}{5}{6}{7}{8}{9}}}%
           \meaning\test\par
  \noindent\fdef\test {\xintKeepUnbraced {7}{123456789}}\meaning\test\par
  \noindent\fdef\test {\xintKeepUnbraced {-7}{123456789}}\meaning\test\par
macro:->12345678910
macro:->1234567
macro:->{3}{4}{5}{6}{7}{8}{9}
macro:->1234567
macro:->3456789
```

7.9. \xintTrim

code

num x f ★ \xintTrim{x}{⟨*list*⟩} expands the list argument and gobbles its first x elements.

- if x>0, the first x items from L are gobbled. The remaining items are not modified.
- if x>=length(L), the returned list is empty.
- if x=0 the original list is returned (with no added braces.)
- if x<0 the last |x| items of the list are removed. *The head items end up braced in the output.* Use \xintTrimUnbraced if this is not desired.
- if x<=-length(L) the output is empty.

\xintTrimNoExpand does the same without first *f*-expanding its list argument.

```
  \fdef\test {\xintTrim {17}{\xintTrim {-69}{\xintSeq {1}{100}}}}\meaning\test\par
  \noindent\fdef\test {\xintTrim {7}{{1}{2}{3}{4}{5}{6}{7}{8}{9}}}\meaning\test\par
  \noindent\fdef\test {\xintTrim {-7}{{1}{2}{3}{4}{5}{6}{7}{8}{9}}}\meaning\test\par
  \noindent\fdef\test {\xintTrim {7}{123456789}}\meaning\test\par
  \noindent\fdef\test {\xintTrim {-7}{123456789}}\meaning\test\par
macro:->{18}{19}{20}{21}{22}{23}{24}{25}{26}{27}{28}{29}{30}{31}
macro:->{8}{9}
macro:->{1}{2}
macro:->89
macro:->{1}{2}
```

7.10. \xintTrimUnbraced

code

Same as \xintTrim but in case of a negative x (cutting items from the tail), the kept items from the head are not enclosed in brace pairs. They will lose one level of braces. The name is a bit misleading because when x>0 there is no brace-stripping done on the kept items, because the macro works simply by gobbling the head ones.

\xintTrimUnbracedNoExpand does the same without first f-expanding its list argument.

```
\fdef\test {\xintTrimUnbraced {-90}{\xintSeq {1}{100}}}\meaning\test\par
\noindent\fdef\test {\xintTrimUnbraced {7}{{1}{2}{3}{4}{5}{6}{7}{8}{9}}}%
         \meaning\test\par
\noindent\fdef\test {\xintTrimUnbraced {-7}{{1}{2}{3}{4}{5}{6}{7}{8}{9}}}%
         \meaning\test\par
\noindent\fdef\test {\xintTrimUnbraced {7}{123456789}}\meaning\test\par
\noindent\fdef\test {\xintTrimUnbraced {-7}{123456789}}\meaning\test\par
```

```
macro:->12345678910
macro:->{8}{9}
macro:->12
macro:->89
macro:->12
```

7.11. \xintListWithSep

code

n f ⋆ \xintListWithSep{⟨*sep*⟩}{⟨*list*⟩} inserts the separator ⟨*sep*⟩ in-between all items of the given list of braced items (or individual tokens). The items are fetched as does TEX with undelimited macro arguments, thus they end up unbraced in output. If the ⟨*list*⟩ is only one (or multiple) space tokens, the output is empty.

The list argument ⟨*list*⟩ gets f-expanded first (thus if it is a macro whose contents are braced items, the first opening brace stops the expansion, and it is as if the macro had been expanded once.) The separator ⟨*sep*⟩ is not pre-expanded, it ends up as is in the output (if the ⟨*list*⟩ contained at least two items.)

n n ⋆ The variant \xintListWithSepNoExpand does the same job without the initial expansion of the ⟨*list*⟩ argument.

```
\edef\foo{\xintListWithSep{, }{123456789{10}{11}{12}}}\meaning\foo\newline
\edef\foo{\xintListWithSep{:}{\xintiiFac{20}}}\meaning\foo\newline
\oodef\FOO{\xintListWithSepNoExpand{\FOO}{\bat\baz\biz\buz}}\meaning\FOO\newline
% a braced item or a space stops the f-expansion:
\oodef\foo{\xintListWithSep{\FOO}{{\bat}\baz\biz\buz}}\meaning\foo\newline
\oodef\foo{\xintListWithSep{\FOO}{ \bat\baz\biz\buz}}\meaning\foo\par
```

```
macro:->1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12
macro:->2:4:3:2:9:0:2:0:0:8:1:7:6:6:4:0:0:0:0
macro:->\bat \FOO \baz \FOO \biz \FOO \buz
macro:->\bat \FOO \baz \FOO \biz \FOO \buz
macro:->\bat \FOO \baz \FOO \biz \FOO \buz
```

7.12. \xintApply

code

f f ⋆ \xintApply{\macro}{⟨*list*⟩} expandably applies the one parameter macro \macro to each item in the ⟨*list*⟩ given as second argument and returns a new list with these outputs: each item is given one after the other as parameter to \macro which is expanded at that time (as usual, *i.e.* fully for what comes first), the results are braced and output together as a succession of braced items (if \macro is defined to start with a space, the space will be gobbled and the \macro will not be expanded; it is allowed to have its own arguments, the list items serve as last arguments to \macro). Hence

\xintApply{\macro}{{1}{2}{3}} returns {\macro{1}}{\macro{2}}{\macro{3}} where all instances of \macro have been already *f*-expanded.

Trick: {\macro} is perfectly allowed to be {\macro{17}} if \macro is actually a macro with *two* arguments. It will thus be expanded with the first one being always here {17}.

Being expandable, \xintApply is useful for example inside alignments where implicit groups make standard loops constructs usually fail. In such situation it is often not wished that the new list elements be braced, see \xintApplyUnbraced. The \macro does not have to be expandable: \xintApply will try to expand it, the expansion may remain partial.

The ⟨*list*⟩ may itself be some macro expanding (in the previously described way) to the list of tokens to which the macro \macro will be applied. For example, if the ⟨*list*⟩ expands to some positive number, then each digit will be replaced by the result of applying \macro on it.

```
\def\macro #1{\the\numexpr 9-#1\relax}
\xintApply\macro{\xintiiFac {20}}=756709799182335999
```

fn ⋆ The macro \xintApplyNoExpand does the same job without the first initial expansion which gave the ⟨*list*⟩ of braced tokens to which \macro is applied.

7.13. \xintApplyUnbraced

code

ff ⋆ \xintApplyUnbraced{\macro}{⟨*list*⟩} is like \xintApply. The difference is that after having expanded its list argument, and applied \macro in turn to each item from the list, it reassembles the outputs without enclosing them in braces. The net effect is the same as doing

```
\xintListWithSep {}{\xintApply {\macro}{⟨list⟩}}
```

This is useful for preparing a macro which will itself define some other macros or make assignments, as the scope will not be limited by brace pairs.

```
\def\macro #1{\expandafter\def\csname myself#1\endcsname {#1}}
\xintApplyUnbraced\macro{{elta}{eltb}{eltc}}
\begin{enumerate}[nosep,label=(\arabic{*})]
\item \meaning\myselfelta
\item \meaning\myselfeltb
\item \meaning\myselfeltc
\end{enumerate}
```

(1) macro:->elta
(2) macro:->eltb
(3) macro:->eltc

fn ⋆ The macro \xintApplyUnbracedNoExpand does the same job without the first initial expansion which gave the ⟨*list*⟩ of braced tokens to which \macro is applied.

7.14. \xintSeq

code

$[^{\text{num}}_{\ }]\ {}^{\text{num}}_{x}\ {}^{\text{num}}_{x}$ ⋆ \xintSeq[d]{x}{y} generates expandably {x}{x+d}... up to and possibly including {y} if d>0 or down to and including {y} if d<0. Naturally {y} is omitted if y-x is not a multiple of d. If d=0 the macro returns {x}. If y-x and d have opposite signs, the macro returns nothing. If the optional argument d is omitted it is taken to be the sign of y-x. Hence \xintSeq {1}{0} is not empty but {1}{0}. But \xintSeq [1]{1}{0} is empty.

The arguments x and y are expanded inside a \numexpr so they may be count registers or a LaTeX \value{countername}, or arithmetic with such things.

```
\xintListWithSep{,\hskip2pt plus 1pt minus 1pt }{\xintSeq {12}{-25}}
```

12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0, -1, -2, -3, -4, -5, -6, -7, -8, -9, -10, -11, -12, -13, -14, -15, -16, -17, -18, -19, -20, -21, -22, -23, -24, -25

```
\xintiiSum{\xintSeq [3]{1}{1000}}
```

167167

When the macro is used without the optional argument d, it can only generate up to about 5000 numbers, the precise value depends upon some TEX memory parameter (input save stack).

With the optional argument d the macro proceeds differently (but less efficiently) and does not stress the input save stack.

7.15. \xintloop, \xintbreakloop, \xintbreakloopanddo, \xintloopskiptonext

code code code code

☆ \xintloop⟨*stuff*⟩\if<test>...\repeat is an expandable loop compatible with nesting. However to break out of the loop one almost always need some un-expandable step. The cousin \xintiloop is \xintloop with an embedded expandable mechanism allowing to exit from the loop. The iterated macros may contain \par tokens or empty lines.

If a sub-loop is to be used all the material from the start of the main loop and up to the end of the entire subloop should be braced; these braces will be removed and do not create a group. The simplest to allow the nesting of one or more sub-loops is to brace everything between \xintloop and \repeat, being careful not to leave a space between the closing brace and \repeat.

As this loop and \xintiloop will primarily be of interest to experienced TEX macro programmers, my description will assume that the user is knowledgeable enough. Some examples in this document will be perhaps more illustrative than my attemps at explanation of use.

One can abort the loop with \xintbreakloop; this should not be used inside the final test, and one should expand the \fi from the corresponding test before. One has also \xintbreakloopanddo whose first argument will be inserted in the token stream after the loop; one may need a macro such as \xint_afterfi to move the whole thing after the \fi, as a simple \expandafter will not be enough.

One will usually employ some count registers to manage the exit test from the loop; this breaks expandability, see \xintiloop for an expandable integer indexed loop. Use in alignments will be complicated by the fact that cells create groups, and also from the fact that any encountered un-expandable material will cause the TEX input scanner to insert \endtemplate on each encountered & or \cr; thus \xintbreakloop may not work as expected, but the situation can be resolved via \xint_firstofone{&} or use of \TAB with \def\TAB{&}. It is thus simpler for alignments to use rather than \xintloop either the expandable \xintApplyUnbraced or the non-expandable but alignment compatible \xintApplyInline, \xintFor or \xintFor*.

As an example, let us suppose we have two macros \A{⟨*i*⟩}{⟨*j*⟩} and \B{⟨*i*⟩}{⟨*j*⟩} behaving like (small) integer valued matrix entries, and we want to define a macro \C{⟨*i*⟩}{⟨*j*⟩} giving the matrix product (i and j may be count registers). We will assume that \A[I] expands to the number of rows, \A[J] to the number of columns and want the produced \C to act in the same manner. The code is very dispendious in use of \count registers, not optimized in any way, not made very robust (the defined macro can not have the same name as the first two matrices for example), we just wanted to quickly illustrate use of the nesting capabilities of \xintloop.[35]

```
\newcount\rowmax   \newcount\colmax   \newcount\summax
\newcount\rowindex \newcount\colindex \newcount\sumindex
\newcount\tmpcount
\makeatletter
\def\MatrixMultiplication #1#2#3{%
    \rowmax #1[I]\relax
    \colmax #2[J]\relax
    \summax #1[J]\relax
    \rowindex 1
    \xintloop % loop over row index i
    {\colindex 1
     \xintloop % loop over col index k
```

[35] for a more sophisticated implementation of matrix multiplication, inclusive of determinants, inverses, and display utilities, with entries big integers or decimal numbers or even fractions see some code online posted from November 11, 2013.

```
      {\tmpcount 0
       \sumindex 1
       \xintloop % loop over intermediate index j
       \advance\tmpcount \numexpr #1\rowindex\sumindex*#2\sumindex\colindex\relax
       \ifnum\sumindex<\summax
          \advance\sumindex 1
       \repeat }%
      \expandafter\edef\csname\string#3{\the\rowindex.\the\colindex}\endcsname
       {\the\tmpcount}%
      \ifnum\colindex<\colmax
          \advance\colindex 1
      \repeat }%
     \ifnum\rowindex<\rowmax
     \advance\rowindex 1
     \repeat
     \expandafter\edef\csname\string#3{I}\endcsname{\the\rowmax}%
     \expandafter\edef\csname\string#3{J}\endcsname{\the\colmax}%
     \def #3##1{\ifx[##1\expandafter\Matrix@helper@size
                    \else\expandafter\Matrix@helper@entry\fi #3{##1}}%
}%
\def\Matrix@helper@size #1#2#3]{\csname\string#1{#3}\endcsname }%
\def\Matrix@helper@entry #1#2#3%
   {\csname\string#1{\the\numexpr#2.\the\numexpr#3}\endcsname }%
\def\A #1{\ifx[#1\expandafter\A@size
            \else\expandafter\A@entry\fi {#1}}%
\def\A@size #1#2]{\ifx I#23\else4\fi}% 3rows, 4columns
\def\A@entry #1#2{\the\numexpr #1+#2-1\relax}% not pre-computed...
\def\B #1{\ifx[#1\expandafter\B@size
            \else\expandafter\B@entry\fi {#1}}%
\def\B@size #1#2]{\ifx I#24\else3\fi}% 4rows, 3columns
\def\B@entry #1#2{\the\numexpr #1-#2\relax}% not pre-computed...
\makeatother
\MatrixMultiplication\A\B\C \MatrixMultiplication\C\C\D
\MatrixMultiplication\C\D\E \MatrixMultiplication\C\E\F
\begin{multicols}2
  \[\begin{pmatrix}
    \A11&\A12&\A13&\A14\\
    \A21&\A22&\A23&\A24\\
    \A31&\A32&\A33&\A34
  \end{pmatrix}
  \times
  \begin{pmatrix}
    \B11&\B12&\B13\\
    \B21&\B22&\B23\\
    \B31&\B32&\B33\\
    \B41&\B42&\B43
  \end{pmatrix}
  =
  \begin{pmatrix}
    \C11&\C12&\C13\\
    \C21&\C22&\C23\\
    \C31&\C32&\C33
```

```
  \end{pmatrix}\]
  \[\begin{pmatrix}
    \C11&\C12&\C13\\
    \C21&\C22&\C23\\
    \C31&\C32&\C33
  \end{pmatrix}^2 = \begin{pmatrix}
    \D11&\D12&\D13\\
    \D21&\D22&\D23\\
    \D31&\D32&\D33
  \end{pmatrix}\]
  \[\begin{pmatrix}
    \C11&\C12&\C13\\
    \C21&\C22&\C23\\
    \C31&\C32&\C33
  \end{pmatrix}^3 = \begin{pmatrix}
    \E11&\E12&\E13\\
    \E21&\E22&\E23\\
    \E31&\E32&\E33
  \end{pmatrix}\]
  \[\begin{pmatrix}
    \C11&\C12&\C13\\
    \C21&\C22&\C23\\
    \C31&\C32&\C33
  \end{pmatrix}^4 = \begin{pmatrix}
    \F11&\F12&\F13\\
    \F21&\F22&\F23\\
    \F31&\F32&\F33
  \end{pmatrix}\]
\end{multicols}
```

$$\begin{pmatrix}1&2&3&4\\2&3&4&5\\3&4&5&6\end{pmatrix}\times\begin{pmatrix}0&-1&-2\\1&0&-1\\2&1&0\\3&2&1\end{pmatrix}=\begin{pmatrix}20&10&0\\26&12&-2\\32&14&-4\end{pmatrix}$$

$$\begin{pmatrix}20&10&0\\26&12&-2\\32&14&-4\end{pmatrix}^3=\begin{pmatrix}20880&10160&-560\\24624&11968&-688\\28368&13776&-816\end{pmatrix}$$

$$\begin{pmatrix}20&10&0\\26&12&-2\\32&14&-4\end{pmatrix}^2=\begin{pmatrix}660&320&-20\\768&376&-16\\876&432&-12\end{pmatrix}$$

$$\begin{pmatrix}20&10&0\\26&12&-2\\32&14&-4\end{pmatrix}^4=\begin{pmatrix}663840&322880&-18080\\781632&380224&-21184\\899424&437568&-24288\end{pmatrix}$$

7.16. \xintiloop, \xintiloopindex, \xintouteriloopindex, \xintbreakiloop, \xintbreakiloopanddo, \xintiloopskiptonext, \xintiloopskipandredo

☆ \xintiloop[start+delta]⟨*stuff*⟩\if<test> ... \repeat is a completely expandable nestable loop. complete expandability depends naturally on the actual iterated contents, and complete expansion will not be achievable under a sole *f*-expansion, as is indicated by the hollow star in the margin; thus the loop can be used inside an \edef but not inside arguments to the package macros. It can be used inside an \xintexpr..\relax. The [start+delta] is mandatory, not optional.

This loop benefits via \xintiloopindex to (a limited access to) the integer index of the iteration. The starting value start (which may be a \count) and increment delta (*id.*) are mandatory arguments. A space after the closing square bracket is not significant, it will be ignored.

Spaces inside the square brackets will also be ignored as the two arguments are first given to a \numexpr...\relax. Empty lines and explicit \par tokens are accepted.

As with \xintloop, this tool will mostly be of interest to advanced users. For nesting, one puts inside braces all the material from the start (immediately after [start+delta]) and up to and inclusive of the inner loop, these braces will be removed and do not create a loop. In case of nesting, \xintouteriloopindex gives access to the index of the outer loop. If needed one could write on its model a macro giving access to the index of the outer outer loop (or even to the nth outer loop).

The \xintiloopindex and \xintouteriloopindex can not be used inside braces, and generally speaking this means they should be expanded first when given as argument to a macro, and that this macro receives them as delimited arguments, not braced ones. Or, but naturally this will break expandability, one can assign the value of \xintiloopindex to some \count. Both \xintiloopindex and \xintouteriloopindex extend to the litteral representation of the index, thus in \ifnum tests, if it comes last one has to correctly end the macro with a \space, or encapsulate it in a \numexpr..\relax.

When the repeat-test of the loop is, for example, \ifnum\xintiloopindex<10 \repeat, this means that the last iteration will be with \xintiloopindex=10 (assuming delta=1). There is also \ifnum\xintiloopindex=10 \else\repeat to get the last iteration to be the one with \xintiloopindex=10.

One has \xintbreakiloop and \xintbreakiloopanddo to abort the loop. The syntax of \xintbreakiloopanddo is a bit surprising, the sequence of tokens to be executed after breaking the loop is not within braces but is delimited by a dot as in:

```
\xintbreakiloopanddo <afterloop>.etc.. etc... \repeat
```

The reason is that one may wish to use the then current value of \xintiloopindex in <afterloop> but it can't be within braces at the time it is evaluated. However, it is not that easy as \xintiloopindex must be expanded before, so one ends up with code like this:

```
\expandafter\xintbreakiloopanddo\expandafter\macro\xintiloopindex.%
etc.. etc.. \repeat
```

As moreover the \fi from the test leading to the decision of breaking out of the loop must be cleared out of the way, the above should be a branch of an expandable conditional test, else one needs something such as:

```
\xint_afterfi{\expandafter\xintbreakiloopanddo\expandafter\macro\xintiloopindex.}%
\fi etc..etc.. \repeat
```

There is \xintiloopskiptonext to abort the current iteration and skip to the next, \xintiloopskipandredo to skip to the end of the current iteration and redo it with the same value of the index (something else will have to change for this not to become an eternal loop...).

Inside alignments, if the looped-over text contains a & or a \cr, any un-expandable material before a \xintiloopindex will make it fail because of \endtemplate; in such cases one can always either replace & by a macro expanding to it or replace it by a suitable \firstofone{&}, and similarly for \cr.

As an example, let us construct an \edef\z{...} which will define \z to be a list of prime numbers:

```
\begingroup
\edef\z
{\xintiloop [10001+2]
  {\xintiloop [3+2]
   \ifnum\xintouteriloopindex<\numexpr\xintiloopindex*\xintiloopindex\relax
         \xintouteriloopindex,
         \expandafter\xintbreakiloop
   \fi
   \ifnum\xintouteriloopindex=\numexpr
        (\xintouteriloopindex/\xintiloopindex)*\xintiloopindex\relax
   \else
```

```
    \repeat
   }% no space here
  \ifnum \xintiloopindex < 10999 \repeat }%
 \meaning\z\endgroup
```

```
macro:->10007, 10009, 10037, 10039, 10061, 10067, 10069, 10079, 10091, 10093, 10099, 10103,
10111, 10133, 10139, 10141, 10151, 10159, 10163, 10169, 10177, 10181, 10193, 10211, 10223, 10243,
10247, 10253, 10259, 10267, 10271, 10273, 10289, 10301, 10303, 10313, 10321, 10331, 10333, 10337,
10343, 10357, 10369, 10391, 10399, 10427, 10429, 10433, 10453, 10457, 10459, 10463, 10477, 10487,
10499, 10501, 10513, 10529, 10531, 10559, 10567, 10589, 10597, 10601, 10607, 10613, 10627, 10631,
10639, 10651, 10657, 10663, 10667, 10687, 10691, 10709, 10711, 10723, 10729, 10733, 10739, 10753,
10771, 10781, 10789, 10799, 10831, 10837, 10847, 10853, 10859, 10861, 10867, 10883, 10889, 10891,
10903, 10909, 10937, 10939, 10949, 10957, 10973, 10979, 10987, 10993,
```

and we should have taken some steps to not have a trailing comma, but the point was to show that one can do that in an `\edef`! See also subsection 8.4 which extracts from this code its way of testing primality.

Let us create an alignment where each row will contain all divisors of its first entry. Here is the output, thus obtained without any count register:

```
\begin{multicols}2
\tabskip1ex \normalcolor
\halign{&\hfil#\hfil\cr
    \xintiloop [1+1]
    {\expandafter\bfseries\xintiloopindex &
     \xintiloop [1+1]
     \ifnum\xintouteriloopindex=\numexpr
           (\xintouteriloopindex/\xintiloopindex)*\xintiloopindex\relax
     \xintiloopindex&\fi
     \ifnum\xintiloopindex<\xintouteriloopindex\space % CRUCIAL \space HERE
     \repeat \cr }%
    \ifnum\xintiloopindex<30
    \repeat
}
\end{multicols}
```

1 1
2 1 2
3 1 3
4 1 2 4
5 1 5
6 1 2 3 6
7 1 7
8 1 2 4 8
9 1 3 9
10 1 2 5 10
11 1 11
12 1 2 3 4 6 12
13 1 13
14 1 2 7 14
15 1 3 5 15
16 1 2 4 8 16
17 1 17
18 1 2 3 6 9 18
19 1 19
20 1 2 4 5 10 20
21 1 3 7 21
22 1 2 11 22
23 1 23
24 1 2 3 4 6 8 12 24
25 1 5 25
26 1 2 13 26
27 1 3 9 27
28 1 2 4 7 14 28
29 1 29
30 1 2 3 5 6 10 15 30

We wanted this first entry in bold face, but `\bfseries` leads to unexpandable tokens, so the `\exp`↩ `andafter` was necessary for `\xintiloopindex` and `\xintouteriloopindex` not to be confronted with a hard to digest `\endtemplate`. An alternative way of coding:

```
\tabskip1ex
\def\firstofone #1{#1}%
```

```
\halign{&\hfil#\hfil\cr
  \xintiloop [1+1]
    {\bfseries\xintiloopindex\firstofone{&}%
    \xintiloop [1+1] \ifnum\xintouteriloopindex=\numexpr
    (\xintouteriloopindex/\xintiloopindex)*\xintiloopindex\relax
    \xintiloopindex\firstofone{&}\fi
    \ifnum\xintiloopindex<\xintouteriloopindex\space % \space is CRUCIAL
    \repeat \firstofone{\cr}}%
  \ifnum\xintiloopindex<30 \repeat }
```

The next utilities are not compatible with expansion-only context.

7.17. \xintApplyInline

code

*o *f* \xintApplyInline{\macro}{⟨*list*⟩} works non expandably. It applies the one-parameter \macro to the first element of the expanded list (\macro may have itself some arguments, the list item will be appended as last argument), and is then re-inserted in the input stream after the tokens resulting from this first expansion of \macro. The next item is then handled.

This is to be used in situations where one needs to do some repetitive things. It is not expandable and can not be completely expanded inside a macro definition, to prepare material for later execution, contrarily to what \xintApply or \xintApplyUnbraced achieve.

```
\def\Macro #1{\advance\cnta #1 , \the\cnta}
\cnta 0
0\xintApplyInline\Macro {3141592653}.
```

0, 3, 4, 8, 9, 14, 23, 25, 31, 36, 39. The first argument \macro does not have to be an expandable macro.

\xintApplyInline submits its second, token list parameter to an *f*-expansion. Then, each *unbraced* item will also be *f*-expanded. This provides an easy way to insert one list inside another. *Braced* items are not expanded. Spaces in-between items are gobbled (as well as those at the start or the end of the list), but not the spaces *inside* the braced items.

\xintApplyInline, despite being non-expandable, does survive to contexts where the executed \macro closes groups, as happens inside alignments with the tabulation character &. This tabular provides an example:

```
\centerline{\normalcolor\begin{tabular}{ccc}
     $N$ & $N^2$ & $N^3$ \\ \hline
     \def\Row #1{ #1 & \xintiiSqr {#1} & \xintiiPow {#1}{3} \\ \hline }%
     \xintApplyInline \Row {\xintCSVtoList{17,28,39,50,61}}
\end{tabular}}\medskip
```

N	N^2	N^3
17	289	4913
28	784	21952
39	1521	59319
50	2500	125000
61	3721	226981

We see that despite the fact that the first encountered tabulation character in the first row close a group and thus erases \Row from TEX's memory, \xintApplyInline knows how to deal with this.

Using \xintApplyUnbraced is an alternative: the difference is that this would have prepared all rows first and only put them back into the token stream once they are all assembled, whereas with \xintApplyInline each row is constructed and immediately fed back into the token stream: when one does things with numbers having hundreds of digits, one learns that keeping on hold and shuffling

around hundreds of tokens has an impact on TEX's speed (make this ``thousands of tokens'' for the impact to be noticeable).

One may nest various \xintApplyInline's. For example (see the table on the current page):

```
\begin{figure*}[ht!]
  \centering\phantomsection\label{float}
  \def\Row #1{#1:\xintApplyInline {\Item {#1}}{0123456789}\\ }%
  \def\Item #1#2{&\xintiiPow {#1}{#2}}%
  \centeredline {\begin{tabular}{ccccccccccc} &0&1&2&3&4&5&6&7&8&9\\ \hline
      \xintApplyInline \Row {0123456789}
    \end{tabular}}
\end{figure*}
```

	0	1	2	3	4	5	6	7	8	9
0:	1	0	0	0	0	0	0	0	0	0
1:	1	1	1	1	1	1	1	1	1	1
2:	1	2	4	8	16	32	64	128	256	512
3:	1	3	9	27	81	243	729	2187	6561	19683
4:	1	4	16	64	256	1024	4096	16384	65536	262144
5:	1	5	25	125	625	3125	15625	78125	390625	1953125
6:	1	6	36	216	1296	7776	46656	279936	1679616	10077696
7:	1	7	49	343	2401	16807	117649	823543	5764801	40353607
8:	1	8	64	512	4096	32768	262144	2097152	16777216	134217728
9:	1	9	81	729	6561	59049	531441	4782969	43046721	387420489

One could not move the definition of \Item inside the tabular, as it would get lost after the first &. But this works:

```
\begin{tabular}{ccccccccccc}
    &0&1&2&3&4&5&6&7&8&9\\ \hline
    \def\Row #1{#1:\xintApplyInline {&\xintiiPow {#1}}{0123456789}\\ }%
    \xintApplyInline \Row {0123456789}
\end{tabular}
```

A limitation is that, contrarily to what one may have expected, the \macro for an \xintApplyInline can not be used to define the \macro for a nested sub-\xintApplyInline. For example, this does not work:

```
  \def\Row #1{#1:\def\Item ##1{&\xintiiPow {#1}{##1}}%
                \xintApplyInline \Item {0123456789}\\ }%
  \xintApplyInline \Row {0123456789} % does not work
```

But see \xintFor.

7.18. \xintFor, \xintFor*

code *code*

on \xintFor is a new kind of for loop.[36] Rather than using macros for encapsulating list items, its behaviour is like a macro with parameters: #1, #2, ..., #9 are used to represent the items for up to nine levels of nested loops. Here is an example:

```
\xintFor #9 in {1,2,3} \do {%
  \xintFor #1 in {4,5,6} \do {%
    \xintFor #3 in {7,8,9} \do {%
      \xintFor #2 in {10,11,12} \do {%
      $$#9\times#1\times#3\times#2=\xintiiPrd{{#1}{#2}{#3}{#9}}$$}}}}
```

This example illustrates that one does not have to use #1 as the first one: the order is arbitrary. But each level of nesting should have its specific macro parameter. Nine levels of nesting is

[36] first introduced with xint 1.09c of 2013/10/09.

presumably overkill, but I did not know where it was reasonable to stop. \par tokens are accepted in both the comma separated list and the replacement text.

TEXnical notes:

- The #1 is replaced in the iterated-over text exactly as in general TEX macros or LATEX commands. This spares the user quite a few \expandafter's or other tricks needed with loops which have the values encapsulated in macros, like LATEX's \@for and \@tfor.
- \xintFor (and \xintFor*) isn't purely expandable: one can not use it inside an \edef. But it may be used, as will be shown in examples, in some contexts such as LATEX's tabular which are usually hostile to non-expandable loops.
- \xintFor (and \xintFor*) does some assignments prior to executing each iteration of the replacement text, but it acts purely expandably after the last iteration, hence if for example the replacement text ends with a \\, the loop can be used insided a tabular and be followed by a \hline without creating the dreaded ``Misplaced \noalign'' error.
- As stated in previous item the first iteration follows some non-expandable internal dealings. This means for example that in LATEX, one can not inject a \multicolumn in the first iteration. Sometimes one way work around this by injecting father &\multicolumn or \\ \multicolumn.
- It does not create groups.
- It makes no global assignments.
- The iterated replacement text may close a group which was opened even before the start of the loop (typical example being with & in alignments).

```
\begin{tabular}{rccccc}
    \hline
    \xintFor #1 in {A, B, C} \do {%
      #1:\xintFor #2 in {a, b, c, d, e} \do {&($ #2 \to #1 $)}\\ }%
    \hline
\end{tabular}
```

A:	(a → A)	(b → A)	(c → A)	(d → A)	(e → A)
B:	(a → B)	(b → B)	(c → B)	(d → B)	(e → B)
C:	(a → C)	(b → C)	(c → C)	(d → C)	(e → C)

- There is no facility provided which would give access to a count of the number of iterations as it is technically not easy to do so it in a way working with nested loops while maintaining the ``expandable after done'' property; something in the spirit of \xintiloopindex is possible but this approach would bring its own limitations and complications. Hence the user is invited to update her own count or LATEX counter or macro at each iteration, if needed.
- A \macro whose definition uses internally an \xintFor loop may be used inside another \xintFor loop even if the two loops both use the same macro parameter. The loop definition inside \macro must use ## as is the general rule for definitions done inside macros.
- \xintFor is for comma separated values and \xintFor* for lists of braced items; their respective expansion policies differ. They are described later.

Regarding \xintFor:

- the spaces between the various declarative elements are all optional,
- in the list of comma separated values, spaces around the commas or at the start and end are ignored,
- if an item must contain itself its own commas, then it should be braced, and the braces will be removed before feeding the iterated-over text,
- the list may be a macro, it is expanded only once,
- items are not pre-expanded. The first item should be braced or start with a space if the list is explicit and the item should not be pre-expanded,
- empty items give empty #1's in the replacement text, they are not skipped,
- an empty list executes once the replacement text with an empty parameter value,
- the list, if not a macro, must be braced.

fn Regarding \xintFor:

- it handles lists of braced items (or naked tokens),
- it *f*-expands the list,
- and more generally it *f*-expands each naked token encountered before assigning the #1 values (gobbling spaces in the process); this makes it easy to simulate concatenation of multiple lists\x, \y: if \x expands to {1}{2}{3} and \y expands to {4}{5}{6} then {\x\y} as argument to \xintFor* has the same effect as {{1}{2}{3}{4}{5}{6}}.
  For a further illustration see the use of \xintFor* at the end of subsection 4.19.
- spaces at the start, end, or in-between items are gobbled (but naturally not the spaces inside *braced* items),
- except if the list argument is a macro (with no parameters), it must be braced.,
- an empty list leads to an empty result.

The macro \xintSeq which generates arithmetic sequences is to be used with \xintFor* as its output consists of successive braced numbers (given as digit tokens).

```
\xintFor* #1 in {\xintSeq [+2]{-7}{+2}}\do {stuff
    with #1\xintifForLast{\par}{\newline}}
```

```
stuff with -7
stuff with -5
stuff with -3
stuff with -1
stuff with 1
```

When nesting \xintFor* loops, using \xintSeq in the inner loops is inefficient, as the arithmetic sequence will be re-created each time. A more efficient style is:

```
\edef\innersequence {\xintSeq[+2]{-50}{50}}%
\xintFor* #1 in {\xintSeq {13}{27}} \do
    {\xintFor* #2 in \innersequence \do {stuff with #1 and #2}%
     .. some other macros .. }
```

This is a general remark applying for any nesting of loops, one should avoid recreating the inner lists of arguments at each iteration of the outer loop.

When the loop is defined inside a macro for later execution the # characters must be doubled.[37] For example:

```
\def\T{\def\z {}%
  \xintFor* ##1 in {{u}{v}{w}} \do {%
    \xintFor ##2 in {x,y,z} \do {%
      \expandafter\def\expandafter\z\expandafter {\z\sep (##1,##2)} }%
  }%
}%
\T\def\sep {\def\sep{, }}\z
```

```
(u,x), (u,y), (u,z), (v,x), (v,y), (v,z), (w,x), (w,y), (w,z)
```

Similarly when the replacement text of \xintFor defines a macro with parameters, the macro char-

[37] sometimes what seems to be a macro argument isn't really; in \raisebox{1cm}{\xintFor #1 in {a,b,c}\do {#1}} no doubling should be done.

acter # must be doubled.

The iterated macros as well as the list items are allowed to contain explicit \par tokens.

7.19. \xintifForFirst, \xintifForLast

nn ★ \xintifForFirst{YES branch}{NO branch} and \xintifForLast{YES branch}{NO branch} execute the YES or NO branch if the \xintFor or \xintFor* loop is currently in its first, respectively last, iteration.

Designed to work as expected under nesting (but see frame next.) Don't forget an empty brace pair {} if a branch is to do nothing. May be used multiple times in the replacement text of the loop.

Pay attention to these implementation features:

- *if an inner \xintFor loop is positioned before the \xintifForFirst or \xintifForLast of the outer loop it will contaminate their settings. This applies also naturally if the inner loop arises from the expansion of some macro located before the outer conditionals.*

  One fix is to make sure that the outer conditionals are expanded before the inner loop is executed, e.g. this will be the case if the inner loop is located inside one of the branches of the conditional.

  Another approach is to enclose, if feasible, the inner loop in a group of its own.
- *if the replacement text closes a group (e.g. from a & inside an alignment), the conditionals will lose their ascribed meanings and end up possibly undefined, depending whether there is some outer loop whose execution started before the opening of the group.*

  The fix is to arrange things so that the conditionals are expanded before TEX encounters the closing-group token.

7.20. *code* \xintBreakFor, *code* \xintBreakForAndDo

One may immediately terminate an \xintFor or \xintFor* loop with \xintBreakFor.

As it acts by clearing up all the rest of the replacement text when encountered, it will not work from inside some \if...\fi without suitable \expandafter or swapping technique.

Also it can't be used from inside braces as from there it can't see the end of the replacement text.

There is also \xintBreakForAndDo. Both are illustrated by various examples in the next section which is devoted to ``forever'' loops.

7.21. *code* \xintintegers, *code* \xintdimensions, *code* \xintrationals

If the list argument to \xintFor (or \xintFor*, both are equivalent in this context) is \xintintegers (equivalently \xintegers) or more generally \xintintegers[start+delta] (*the whole within braces*!)[38], then \xintFor does an infinite iteration where #1 (or #2, ..., #9) will run through the arithmetic sequence of (short) integers with initial value start and increment delta (default values: start=1, delta=1; if the optional argument is present it must contains both of them, and they may be explicit integers, or macros or count registers). The #1 (or #2, ..., #9) will stand for \numexpr <opt sign><digits>\relax, and the litteral representation as a string of

[38] the start+delta optional specification may have extra spaces around the plus sign of near the square brackets, such spaces are removed. The same applies with \xintdimensions and \xintrationals.

digits can thus be obtained as `\the#1` or `\number#1`. Such a #1 can be used in an \ifnum test with no need to be postfixed with a space or a \relax and one should *not* add them.

If the list argument is \xintdimensions or more generally \xintdimensions[start+delta] (*within braces*!), then \xintFor does an infinite iteration where #1 (or #2, ..., #9) will run through the arithmetic sequence of dimensions with initial value start and increment delta. Default values: start=0pt, delta=1pt; if the optional argument is present it must contain both of them, and they may be explicit specifications, or macros, or dimen registers, or length macros in LaTeX (the stretch and shrink components will be discarded). The #1 will be \dimexpr <opt sign><digits>sp\relax, from which one can get the litteral (approximate) representation in points via \the#1. So #1 can be used anywhere TeX expects a dimension (and there is no need in conditionals to insert a \relax, and one should *not* do it), and to print its value one uses `\the#1`. The chosen representation guarantees exact incrementation with no rounding errors accumulating from converting into points at each step.

If the list argument to \xintFor (or \xintFor*) is \xintrationals or more generally \xintrationals[start+delta] (*within braces*!), then \xintFor does an infinite iteration where #1 (or #2, ..., #9) will run through the arithmetic sequence of xintfrac fractions with initial value start and increment delta (default values: start=1/1, delta=1/1). This loop works *only with xintfrac loaded*. if the optional argument is present it must contain both of them, and they may be given in any of the formats recognized by xintfrac (fractions, decimal numbers, numbers in scientific notations, numerators and denominators in scientific notation, etc...) , or as macros or count registers (if they are short integers). The #1 (or #2, ..., #9) will be an a/b fraction (without a [n] part), where the denominator b is the product of the denominators of start and delta (for reasons of speed #1 is not reduced to irreducible form, and for another reason explained later start and delta are not put either into irreducible form; the input may use explicitely \xintIrr to achieve that).

```
\begingroup\small
\noindent\parbox{\dimexpr\linewidth-3em}{\color[named]{OrangeRed}%
\xintFor #1 in {\xintrationals [10/21+1/21]} \do
{#1=\xintifInt {#1}
    {\textcolor{blue}{\xintTrunc{10}{#1}}}
    {\xintTrunc{10}{#1}}% display in blue if an integer
    \xintifGt {#1}{1.123}{\xintBreakFor}{, }%
  }}
\endgroup\smallskip
```

10/21=0.4761904761, 11/21=0.5238095238, 12/21=0.5714285714, 13/21=0.6190476190, 14/21=0.6666666666, 15/21=0.7142857142, 16/21=0.7619047619, 17/21=0.8095238095, 18/21=0.8571428571, 19/21=0.9047619047, 20/21=0.9523809523, 21/21=1.0000000000, 22/21=1.0476190476, 23/21=1.0952380952, 24/21=1.1428571428

The example above confirms that computations are done exactly, and illustrates that the two initial (reduced) denominators are not multiplied when they are found to be equal. It is thus recommended to input start and delta with a common smallest possible denominator, or as fixed point numbers with the same numbers of digits after the decimal mark; and this is also the reason why start and delta are not by default made irreducible. As internally the computations are done with numerators and denominators completely expanded, one should be careful not to input numbers in scientific notation with exponents in the hundreds, as they will get converted into as many zeroes.

```
\noindent\parbox{\dimexpr.7\linewidth}{\raggedright
\xintFor #1 in {\xintrationals [0.000+0.125]} \do
{\edef\tmp{\xintTrunc{3}{#1}}%
 \xintifInt {#1}
    {\textcolor{blue}{\tmp}}
    {\tmp}%
    \xintifGt {#1}{2}{\xintBreakFor}{, }%
```

```
   }}\smallskip
0, 0.125, 0.250, 0.375, 0.500, 0.625, 0.750, 0.875, 1.000, 1.125,
1.250, 1.375, 1.500, 1.625, 1.750, 1.875, 2.000, 2.125
```

We see here that \xintTrunc outputs (deliberately) zero as 0, not (here) 0.000, the idea being not to lose the information that the truncated thing was truly zero. Perhaps this behaviour should be changed? or made optional? Anyhow printing of fixed points numbers should be dealt with via dedicated packages such as numprint or siunitx.

7.22. \xintForpair, \xintForthree, \xintForfour

code *code* *code*

on The syntax is illustrated in this example. The notation is the usual one for n-uples, with parentheses and commas. Spaces around commas and parentheses are ignored.

```
{\centering\begin{tabular}{cccc}
    \xintForpair #1#2 in { ( A , a ) , ( B , b ) , ( C , c ) } \do {%
      \xintForpair #3#4 in { ( X , x ) , ( Y , y ) , ( Z , z ) } \do {%
        $\Biggl($\begin{tabular}{cc}
          -#1- & -#3-\\
          -#4- & -#2-\\
        \end{tabular}$\Biggr)$&}\\\noalign{\vskip1\jot}}%
\end{tabular}\\}
```

$$\begin{pmatrix} -A- & -X- \\ -x- & -a- \end{pmatrix} \quad \begin{pmatrix} -A- & -Y- \\ -y- & -a- \end{pmatrix} \quad \begin{pmatrix} -A- & -Z- \\ -z- & -a- \end{pmatrix}$$
$$\begin{pmatrix} -B- & -X- \\ -x- & -b- \end{pmatrix} \quad \begin{pmatrix} -B- & -Y- \\ -y- & -b- \end{pmatrix} \quad \begin{pmatrix} -B- & -Z- \\ -z- & -b- \end{pmatrix}$$
$$\begin{pmatrix} -C- & -X- \\ -x- & -c- \end{pmatrix} \quad \begin{pmatrix} -C- & -Y- \\ -y- & -c- \end{pmatrix} \quad \begin{pmatrix} -C- & -Z- \\ -z- & -c- \end{pmatrix}$$

\xintForpair must be followed by either #1#2, #2#3, #3#4, ..., or #8#9 with #1 usable as an alias for #1#2, #2 as alias for #2#3, etc ... and similarly for \xintForthree (using #1#2#3 or simply #1, #2#3#4 or simply #2, ...) and \xintForfour (with #1#2#3#4 etc...).

Nesting works as long as the macro parameters are distinct among #1, #2, ..., #9. A macro which expands to an \xintFor or a \xintFor(pair,three,four) can be used in another one with no constraint about using distinct macro parameters.

\par tokens are accepted in both the comma separated list and the replacement text.

7.23. \xintAssign

code

\xintAssign⟨*braced things*⟩\to⟨*as many cs as they are things*⟩ defines (without checking if something gets overwritten) the control sequences on the right of \to to expand to the successive tokens or braced items located to the left of \to. \xintAssign is not an expandable macro.

f-expansion is first applied to the material in front of \xintAssign which is fetched as one argument if it is braced. Then the expansion of this argument is examined and successive items are assigned to the macros following \to. There must be exactly as many macros as items. No check is done. The macro assignments are done with removal of one level of brace pairs from each item.

After the initial *f*-expansion, each assigned (brace-stripped) item will be expanded according to the setting of the optional parameter.

For example \xintAssign [e]... means that all assignments are done using \edef. With [f] the assignments will be made using \fdef. The default is simply to make the definitions with \def, corresponding to an empty optional paramter []. Possibilities for the optional parameter are: [] ↪, [g], [e], [x], [o], [go], [oo], [goo], [f], [gf]. For example [oo] means a double expansion.

```
\xintAssign \xintiiDivision{1000000000000}{133333333}\to\Q\R
```

```
  \meaning\Q\newline
  \meaning\R\newline
  \xintAssign {{\xintiiDivision{1000000000000}{133333333}}}\to\X
  \meaning\X\newline
  \xintAssign [oo]{{\xintiiDivision{1000000000000}{133333333}}}\to\X
  \meaning\X\newline
  \xintAssign \xintiiPow{7}{13}\to\SevenToThePowerThirteen
  \meaning\SevenToThePowerThirteen\par
```

```
macro:->7500
macro:->2500
macro:->\xintiiDivision {1000000000000}{133333333}
macro:->{7500}{2500}
macro:->96889010407
```

Two special cases:

- if after this initial expansion no brace is found immediately after \xintAssign, it is assumed that there is only one control sequence following \to, and this control sequence is then defined via \def (or what is set-up by the optional parameter) to expand to the material between \xintAssign and \to.
- if the material between \xintAssign and \to is enclosed in two brace pairs, the first brace pair is removed, then the *f*-expansion is immediately stopped by the inner brace pair, hence \xintAssign now finds a unique item and thus defines only a single macro to be this item, which is now stripped of the second pair of braces.

Note: prior to release 1.09j, \xintAssign did an \edef by default for each item assignment but it now does \def corresponding to no or empty optional parameter.

It is allowed for the successive braced items to be separated by spaces. They are removed during the assignments. But if a single macro is defined (which happens if the argument after *f*-expansion does not start with a brace), naturally the scooped up material has all intervening spaces, as it is considered a single item. But an upfront initial space will have been absorbed by *f*-expansion.

```
  \def\X{ {a}  {b} {c}   {d} }\def\Y { u {a}  {b} {c}   {d} }
  \xintAssign\X\to\A\B\C\D
  \xintAssign\Y\to\Z
  \meaning\A, \meaning\B, \meaning\C, \meaning\D+++\newline
  \meaning\Z+++\par
```

```
macro:->a, macro:->b, macro:->c, macro:->d+++
macro:->u {a} {b} {c} {d} +++
```

As usual successive space characters in input make for a single TEX space token.

7.24. \xintAssignArray

code

\xintAssignArray⟨*braced things*⟩\to\myArray first expands fully what comes immediately after \xintAssignArray and expects to find a list of braced things {A}{B}... (or tokens). It then defines \myArray as a macro with one parameter, such that \myArray{x} expands to give the xth braced thing of this original list (the argument {x} itself is fed to a \numexpr by \myArray, and \myArray expands in two steps to its output). With 0 as parameter, \myArray{0} returns the number M of elements of the array so that the successive elements are \myArray{1}, . . . , \myArray{M}.

```
  \xintAssignArray \xintBezout {1000}{113}\to\Bez
```

will set \Bez{0} to 3, \Bez{1} to -20, \Bez{2} to 177, and \Bez{3} to 1: $-20\times 1000 + 177\times 113 = 1$. This macro is incompatible with expansion-only contexts.

\xintAssignArray admits an optional parameter, for example \xintAssignArray [e] means that the definitions of the macros will be made with \edef. The empty optional parameter (default) means that definitions are done with \def. Other possibilities: [], [o], [oo], [f]. Contrarily to

\xintAssign one can not use the g here to make the definitions global. For this, one should rather do \xintAssignArray within a group starting with \globaldefs 1.

7.25. \xintDigitsOf

code

fN This is a synonym for \xintAssignArray, to be used to define an array giving all the digits of a given (positive, else the minus sign will be treated as first item) number.

```
\xintDigitsOf\xintiiPow {7}{500}\to\digits
```

7^{500} has \digits{0}=423 digits, and the 123rd among them (starting from the most significant) is \digits{123}=3.

7.26. \xintRelaxArray

\xintRelaxArray\myArray (globally) sets to \relax all macros which were defined by the previous \xintAssignArray with \myArray as array macro.

8. Additional (old) examples with xinttools or xintexpr or both

.1 Introduction 116
.2 More examples with dummy variables 116
.3 Completely expandable prime test 117
.4 Another completely expandable prime test 118
.5 Miller-Rabin Pseudo-Primality expandably 120
.6 A table of factorizations 123
.7 Another table of primes 124
.8 Factorizing again 126
.9 The Quick Sort algorithm illustrated 127

8.1. Introduction

Note: xintexpr.sty automatically loads xinttools.sty.

The examples given here start to feel dated and are currently in need of some rewrite to better illustrate newer features of the package.

8.2. More examples with dummy variables

These examples were first added to this manual at the time of the 1.1 release (2014/10/29).

```
Prime numbers are always cool
\xinttheiiexpr seq((seq((subs((x/:m)?{(m*m>x)?{1}{0}}{-1},m=2n+1))
                       ??{break(0)}{omit}{break(1)},n=1++))?{x}{omit},
                 x=10001..[2]..10200)\relax
```

Prime numbers are always cool 10007, 10009, 10037, 10039, 10061, 10067, 10069, 10079, 10091, 10093, 10099, 10103, 10111, 10133, 10139, 10141, 10151, 10159, 10163, 10169, 10177, 10181, 10193

The syntax in this last example may look a bit involved (... and it is so I admit). First x/:m computes x modulo m (this is the modulo with respect to floored division). The (x)?{yes}{no} construct checks if x (which *must* be within parentheses) is true or false, i.e. non zero or zero. It then executes either the yes or the no branch, the non chosen branch is *not* evaluated. Thus if m divides x we are in the second (``false'') branch. This gives a -1. This -1 is the argument to a ?? branch which is of the type (y)??{y<0}{y=0}{y>0}, thus here the y<0, i.e., break(0) is chosen. This 0 is thus given to another ? which consequently chooses omit, hence the number is not kept in the list. The numbers which survive are the prime numbers.

```
The first Fibonacci number beyond |2^64|, |2^64|, and the index are respectively
\xinttheiiexpr subs(iterr(0,1;(@1>N)?{break(@1,N,i)}{@1+@2},i=1++),N=2^64)\relax.
```

The first Fibonacci number beyond 2^64, 2^64, and the index are respectively 19740274219868223167, 18446744073709551616, 94.

One more recursion:

```
\def\syr #1{\xinttheiiexpr
            rseq(#1; (@<=1)?{break(i)}{odd(@)?{3@+1}{@//2}},i=0++)\relax}
The 3x+1 problem: \syr{231}\par
```

The 3x+1 problem: 231, 694, 347, 1042, 521, 1564, 782, 391, 1174, 587, 1762, 881, 2644, 1322, 661, 1984, 992, 496, 248, 124, 62, 31, 94, 47, 142, 71, 214, 107, 322, 161, 484, 242, 121, 364, 182, 91, 274, 137, 412, 206, 103, 310, 155, 466, 233, 700, 350, 175, 526, 263, 790, 395, 1186, 593, 1780, 890, 445, 1336, 668, 334, 167, 502, 251, 754, 377, 1132, 566, 283, 850, 425, 1276, 638, 319, 958, 479, 1438, 719, 2158, 1079, 3238, 1619, 4858, 2429, 7288, 3644, 1822, 911, 2734, 1367, 4102, 2051, 6154, 3077, 9232, 4616, 2308, 1154, 577, 1732, 866, 433, 1300, 650, 325, 976, 488, 244, 122, 61, 184, 92, 46, 23, 70, 35, 106, 53, 160, 80, 40, 20, 10, 5, 16, 8, 4, 2, 1, 127

OK, a final one:

```
\def\syrMax #1{\xintiiexpr iterr(#1,#1;even(i)?
                                    {(@2<=1)?{break(@1,i//2)}
                                             {odd(@2)?{3@2+1}{@2//2}}}
                                    {(@1>@2)?{@1}{@2}},i=0++)\relax }
With initial value 1161, the maximal intermediate value and the number of steps
needed to reach 1 are respectively \syrMax{1161}.\par
```

With initial value 1161, the maximal intermediate value and the number of steps needed to reach 1 are respectively 190996, 181.

Look at the Brent-Salamin algorithm implementation for a more interesting recursion.

8.3. Completely expandable prime test

Let us now construct a completely expandable macro which returns 1 if its given input is prime and 0 if not:

```
\def\remainder #1#2{\the\numexpr #1-(#1/#2)*#2\relax }
\def\IsPrime #1%
 {\xintANDof {\xintApply {\remainder {#1}}{\xintSeq {2}{\xintiiSqrt{#1}}}}}
```

This uses \xintiiSqrt and assumes its input is at least 5. Rather than xint's own \xintiiRem we used a quicker \numexpr expression as we are dealing with short integers. Also we used \xintANDof which will return 1 only if all the items are non-zero. The macro is a bit silly with an even input, ok, let's enhance it to detect an even input:

```
\def\IsPrime #1%
   {\xintiiifOdd {#1}
        {\xintANDof % odd case
            {\xintApply {\remainder {#1}}
                        {\xintSeq [2]{3}{\xintiiSqrt{#1}}}%
            }%
        }
        {\xintifEq {#1}{2}{1}{0}}%
   }
```

We used the xint expandable tests (on big integers or fractions) in order for \IsPrime to be *f*-expandable.

Our integers are short, but without \expandafter's with \@firstoftwo, or some other related techniques, direct use of \ifnum..\fi tests is dangerous. So to make the macro more efficient we are going to use the expandable tests provided by the package etoolbox[39]. The macro becomes:

```
\def\IsPrime #1%
   {\ifnumodd {#1}
    {\xintANDof % odd case
     {\xintApply {\remainder {#1}}{\xintSeq [2]{3}{\xintiiSqrt{#1}}}}}
    {\ifnumequal {#1}{2}{1}{0}}}
```

In the odd case however we have to assume the integer is at least 7, as \xintSeq generates an empty list if #1=3 or 5, and \xintANDof returns 1 when supplied an empty list. Let us ease up a bit \xintANDof's work by letting it work on only 0's and 1's. We could use:

```
\def\IsNotDivisibleBy #1#2%
  {\ifnum\numexpr #1-(#1/#2)*#2=0 \expandafter 0\else \expandafter1\fi}
```

where the \expandafter's are crucial for this macro to be *f*-expandable and hence work within the applied \xintANDof. Anyhow, now that we have loaded etoolbox, we might as well use:

```
\newcommand{\IsNotDivisibleBy}[2]{\ifnumequal{#1-(#1/#2)*#2}{0}{0}{1}}
```

Let us enhance our prime macro to work also on the small primes:

```
\newcommand{\IsPrime}[1] % returns 1 if #1 is prime, and 0 if not
  {\ifnumodd {#1}
    {\ifnumless {#1}{8}
      {\ifnumequal{#1}{1}{0}{1}}% 3,5,7 are primes
      {\xintANDof
         {\xintApply
        { \IsNotDivisibleBy {#1}}{\xintSeq [2]{3}{\xintiiSqrt{#1}}}}%
        }}% END OF THE ODD BRANCH
     {\ifnumequal {#1}{2}{1}{0}}% EVEN BRANCH
```

[39] http://ctan.org/pkg/etoolbox

```
}
```

The input is still assumed positive. There is a deliberate blank before \IsNotDivisibleBy to use this feature of \xintApply: a space stops the expansion of the applied macro (and disappears). This expansion will be done by \xintANDof, which has been designed to skip everything as soon as it finds a false (i.e. zero) input. This way, the efficiency is considerably improved.

We did generate via the \xintSeq too many potential divisors though. Later sections give two variants: one with \xintiloop (subsection 8.4) which is still expandable and another one (subsection 8.7) which is a close variant of the \IsPrime code above but with the \xintFor loop, thus breaking expandability. The xintiloop variant does not first evaluate the integer square root, the xintFor variant still does. I did not compare their efficiencies.

Let us construct with this expandable primality test a table of the prime numbers up to 1000. We need to count how many we have in order to know how many tab stops one shoud add in the last row.[40] There is some subtlety for this last row. Turns out to be better to insert a \\ only when we know for sure we are starting a new row; this is how we have designed the \OneCell macro. And for the last row, there are many ways, we use again \xintApplyUnbraced but with a macro which gobbles its argument and replaces it with a tabulation character. The \xintFor* macro would be more elegant here.

```
\newcounter{primecount}
\newcounter{cellcount}
\newcommand{\NbOfColumns}{13}
\newcommand{\OneCell}[1]{%
    \ifnumequal{\IsPrime{#1}}{1}
     {\stepcounter{primecount}
      \ifnumequal{\value{cellcount}}{\NbOfColumns}
       {\\\setcounter{cellcount}{1}#1}
       {&\stepcounter{cellcount}#1}%
     } % was prime
  {}% not a prime, nothing to do
}
\newcommand{\OneTab}[1]{&}
\begin{tabular}{|*{\NbOfColumns}{r|}}
\hline
2  \setcounter{cellcount}{1}\setcounter{primecount}{1}%
   \xintApplyUnbraced \OneCell {\xintSeq [2]{3}{999}}%
   \xintApplyUnbraced \OneTab
      {\xintSeq [1]{1}{\the\numexpr\NbOfColumns-\value{cellcount}\relax}}%
    \\
\hline
\end{tabular}
There are \arabic{primecount} prime numbers up to 1000.
```

The table has been put in float which appears on the following page. We had to be careful to use in the last row \xintSeq with its optional argument [1] so as to not generate a decreasing sequence from 1 to 0, but really an empty sequence in case the row turns out to already have all its cells (which doesn't happen here but would with a number of columns dividing 168).

8.4. Another completely expandable prime test

The \IsPrime macro from subsection 8.3 checked expandably if a (short) integer was prime, here is a partial rewrite using \xintiloop. We use the etoolbox expandable conditionals for convenience, but not everywhere as \xintiloopindex can not be evaluated while being braced. This is also the reason why \xintbreakiloopanddo is delimited, and the next macro \SmallestFactor which

[40] although a tabular row may have less tabs than in the preamble, there is a problem with the | vertical rule, if one does that.

2	3	5	7	11	13	17	19	23	29	31	37	41
43	47	53	59	61	67	71	73	79	83	89	97	101
103	107	109	113	127	131	137	139	149	151	157	163	167
173	179	181	191	193	197	199	211	223	227	229	233	239
241	251	257	263	269	271	277	281	283	293	307	311	313
317	331	337	347	349	353	359	367	373	379	383	389	397
401	409	419	421	431	433	439	443	449	457	461	463	467
479	487	491	499	503	509	521	523	541	547	557	563	569
571	577	587	593	599	601	607	613	617	619	631	641	643
647	653	659	661	673	677	683	691	701	709	719	727	733
739	743	751	757	761	769	773	787	797	809	811	821	823
827	829	839	853	857	859	863	877	881	883	887	907	911
919	929	937	941	947	953	967	971	977	983	991	997	

There are 168 prime numbers up to 1000.

returns the smallest prime factor examplifies that. One could write more efficient completely expandable routines, the aim here was only to illustrate use of the general purpose \xintiloop. A little table giving the first values of \SmallestFactor follows, its coding uses \xintFor, which is described later; none of this uses count registers.

```
% clean up possible left-over mess from previous examples
\let\IsPrime\undefined \let\SmallestFactor\undefined
\newcommand{\IsPrime}[1] % returns 1 if #1 is prime, and 0 if not
  {\ifnumodd {#1}
    {\ifnumless {#1}{8}
      {\ifnumequal{#1}{1}{0}{1}}% 3,5,7 are primes
      {\if
       \xintiloop [3+2]
       \ifnum#1<\numexpr\xintiloopindex*\xintiloopindex\relax
           \expandafter\xintbreakiloopanddo\expandafter1\expandafter.%
       \fi
       \ifnum#1=\numexpr (#1/\xintiloopindex)*\xintiloopindex\relax
       \else
       \repeat 00\expandafter0\else\expandafter1\fi
      }%
    }% END OF THE ODD BRANCH
    {\ifnumequal {#1}{2}{1}{0}}% EVEN BRANCH
}%
\catcode`_ 11
\newcommand{\SmallestFactor}[1] % returns the smallest prime factor of #1>1
  {\ifnumodd {#1}
    {\ifnumless {#1}{8}
      {#1}% 3,5,7 are primes
      {\xintiloop [3+2]
       \ifnum#1<\numexpr\xintiloopindex*\xintiloopindex\relax
           \xint_afterfi{\xintbreakiloopanddo#1.}%
       \fi
       \ifnum#1=\numexpr (#1/\xintiloopindex)*\xintiloopindex\relax
           \xint_afterfi{\expandafter\xintbreakiloopanddo\xintiloopindex.}%
       \fi
       \iftrue\repeat
      }%
     }% END OF THE ODD BRANCH
```

```
    {2}% EVEN BRANCH
}%
\catcode`_ 8
{\centering
  \begin{tabular}{|c|*{10}c|}
    \hline
    \xintFor #1 in {0,1,2,3,4,5,6,7,8,9}\do {&\bfseries #1}\\
    \hline
    \bfseries 0&--&--&2&3&2&5&2&7&2&3\\
    \xintFor #1 in {1,2,3,4,5,6,7,8,9}\do
    {\bfseries #1%
      \xintFor #2 in {0,1,2,3,4,5,6,7,8,9}\do
      {&\SmallestFactor{#1#2}}\\}%
    \hline
  \end{tabular}\par
}
```

	0	1	2	3	4	5	6	7	8	9
0	--	--	2	3	2	5	2	7	2	3
1	2	11	2	13	2	3	2	17	2	19
2	2	3	2	23	2	5	2	3	2	29
3	2	31	2	3	2	5	2	37	2	3
4	2	41	2	43	2	3	2	47	2	7
5	2	3	2	53	2	5	2	3	2	59
6	2	61	2	3	2	5	2	67	2	3
7	2	71	2	73	2	3	2	7	2	79
8	2	3	2	83	2	5	2	3	2	89
9	2	7	2	3	2	5	2	97	2	3

8.5. Miller-Rabin Pseudo-Primality expandably

The isPseudoPrime(n) is usable in \xintiiexpr-essions and establishes if its (positive) argument is a Miller-Rabin PseudoPrime to the bases $2, 3, 5, 7, 11, 13, 17$. If this is true and $n < 341550071728321$ (which has 15 digits) then n really is a prime number.

Similarly $n = 3825123056546413051$ (19 digits) is the smallest composite number which is a strong pseudo prime for bases $2, 3, 5, 7, 11, 13, 17, 19$ and 23. It is easy to extend the code below to include these additional tests (we could make the list of tested bases an argument too, now that I think about it.)

For more information see
https://en.wikipedia.org/wiki/Miller%E2%80%93Rabin_primality_test#Deterministic_variants_of_the_test
and
http://primes.utm.edu/prove/prove2_3.html

In particular, according to Jaeschke *On strong pseudoprimes to several bases*, Math. Comp., 61 (1993) 915-926, if $n < 4,759,123,141$ it is enough to establish Rabin-Miller pseudo-primality to bases $a = 2, 7, 61$ to prove that n is prime. This range is enough for TEX numbers and we could then write a very fast expandable primality test for such numbers using only \numexpr. Left as an exercise...

```
% I ------------------------------- Modular Exponentiation
% Computes x^m modulo n (with m non negative).
% We will always use it with 1 < x < n
%
% With xint 1.4 we should use ? and ?? (although in the case at hand ifsgn()
% and if() would be ok but I should not say that).
%
```

```
\xintdefiifunc powmod_a(x, m, n) :=
       isone(m)?
           % m=1, return x modulo n
           {   x /: n  }
           % m > 1 test if odd or even and do recursive call
           {   odd(m)? {  x*sqr(powmod_a(x, m//2, n)) /: n }
                       {    sqr(powmod_a(x, m//2, n)) /: n }
            }
         ;
\xintdefiifunc powmod(x, m, n) := (m)?{powmod_a(x, m, n)}{1};

%% Syntax used before xint 1.4:
% \xintdefiifunc powmod_a(x, m, n) :=
%        ifone(m,
%            % m=1, return x modulo n
%               x /: n,
%            % m > 1 test if odd or even and do recursive call
%               if(odd(m), (x*sqr(powmod_a(x, m//2, n))) /: n,
%                             sqr(powmod_a(x, m//2, n))  /: n
%                 )
%          );
% \xintdefiifunc powmod(x, m, n) := if(m, powmod_a(x, m, n), 1);

% II ------------------------------ Miller-Rabin compositeness witness

% n=2^k m + 1 with m odd and k at least 1

% Choose 1<x<n.
% compute y=x^m modulo n
% if equals 1 we can't say anything
% if equals n-1 we can't say anything
% else put j=1, and
% compute repeatedly the square, incrementing j by 1 each time,
% thus always we have y^{2^{j-1}}
%   -> if at some point n-1 mod n found, we can't say anything and break out
%   -> if however we never find n-1 mod n before reaching
%        z=y^{2^{k-1}} with j=k
%        we then have z^2=x^{n-1}.
    % Suppose z is not -1 mod n. If z^2 is 1 mod n, then n can be prime only if
    % z is 1 mod n, and we can go back up, until initial y, and we have already
    % excluded y=1. Thus if z is not -1 mod n and z^2 is 1 then n is not prime.
    % But if z^2 is not 1, then n is not prime by Fermat. Hence (z not -1 mod n)
    % implies (n is composite). (Miller test)

% let's use again xintexpr indecipherable (except to author) syntax. Of course
% doing it with macros only would be faster.

% Here \xintdefiifunc is not usable because not compatible with iter, break, ...
% but \xintNewFunction comes to the rescue.

\xintNewFunction{isCompositeWitness}[4]{% x=#1, n=#2, m=#3, k=#4
   subs((y==1)?{0}
```

```
          {iter(y;(j==#4)?{break(!(@==#2-1))}
                        {(@==#2-1)?{break(0)}{sqr(@)/:#2}},j=1++)}
          ,y=powmod(#1,#3,#2))}

% added note (2018/03/07) it is possible in the above that m=#3 is never
% zero, so we should rather call powmod_a for a small gain, but I don't
% have time to re-read the code comments and settle this.

% III ------------------------------------ Strong Pseudo Primes

% cf
%  http://oeis.org/A014233
%     <http://mathworld.wolfram.com/Rabin-MillerStrongPseudoprimeTest.html>
%     <http://mathworld.wolfram.com/StrongPseudoprime.html>

% check if positive integer <49 si a prime.
% 2,3,5,7,11,13,17,19,23,29,31,37,41,43,47
\def\IsVerySmallPrime #1%
    {\ifnum#1=1 \xintdothis0\fi
     \ifnum#1=2 \xintdothis1\fi
     \ifnum#1=3 \xintdothis1\fi
     \ifnum#1=5 \xintdothis1\fi
     \ifnum#1=\numexpr (#1/2)*2\relax\xintdothis0\fi
     \ifnum#1=\numexpr (#1/3)*3\relax\xintdothis0\fi
     \ifnum#1=\numexpr (#1/5)*5\relax\xintdothis0\fi
     \xintorthat 1}

\xintNewFunction{isPseudoPrime}[1]{% n = #1
     (#1<49)?% use ? syntax to evaluate only what is needed
% prior to 1.4 we had \xintthe#1 here but the actual tokens represented
% by this #1 when isPseudoPrime() function expands have changed and
% the correct way is now \xintiieval{#1} to hand over explicit digits to
% the \IsVerySmallPrime macro.
       {\IsVerySmallPrime{\xintiieval{#1}}}
       {(even(#1))?
        {0}
        {subs(%
         % L expands to two values m, k hence isCompositeWitness does get
         % its four variables x, n, m, k
         isCompositeWitness(2, #1, L)?
          {0}%
          {isCompositeWitness(3, #1, L)?
           {0}%
           {isCompositeWitness(5, #1, L)?
            {0}%
            {isCompositeWitness(7, #1, L)?
             {0}%
% above enough for N<3215031751 hence all TeX numbers
             {isCompositeWitness(11, #1, L)?
              {0}%
% above enough for N<2152302898747, hence all 12-digits numbers
              {isCompositeWitness(13, #1, L)?
```

```
                {0}%
% above enough for N<3474749660383
                {isCompositeWitness(17, #1, L)?
                 {0}%
% above enough for N<341550071728321
                 {1}%
                }% not needed to comment-out end of lines spaces inside
               }%  \xintexpr but this is too much of a habit for me with TeX!
              }%   I left some after the ? characters.
             }%
            }%
           }% this computes (m, k) such that n = 2^k m + 1, m odd, k>=1
           , L=iter(#1//2;(even(@))?{@//2}{break(@,k)},k=1++))%
          }%
         }%
}

% if needed:
%\def\IsPseudoPrime #1{\xinttheiiexpr isPseudoPrime(#1)\relax}

\noindent The smallest prime number at least equal to 3141592653589 is
\xintiiexpr
   seq(isPseudoPrime(3141592653589+n)?
                    {break(3141592653589+n)}{omit}, n=0++)\relax.
% we could not use 3141592653589++ syntax because it works only with TeX numbers
\par
```

The smallest prime number at least equal to 3141592653589 is 3141592653601.

8.6. A table of factorizations

As one more example with \xintiloop let us use an alignment to display the factorization of some numbers. The loop will actually only play a minor rôle here, just handling the row index, the row contents being almost entirely produced via a macro \factorize. The factorizing macro does not use \xintiloop as it didn't appear to be the convenient tool. As \factorize will have to be used on \xintiloopindex, it has been defined as a delimited macro.

To spare some fractions of a second in the compilation time of this document (which has many many other things to do), 2147483629 and 2147483647, which turn out to be prime numbers, are not given to factorize but just typeset directly; this illustrates use of \xintiloopskiptonext.

The code next generates a table which has been made into a float appearing on page 125. Here is now the code for factorization; the conditionals use the package provided \xint_firstoftwo and \xint_secondoftwo, one could have employed rather LaTeX's own \@firstoftwo and \@secondoftwo, or, simpler still in LaTeX context, the \ifnumequal, \ifnumless . . ., utilities from the package etoolbox which do exactly that under the hood. Only TeX acceptable numbers are treated here, but it would be easy to make a translation and use the xint macros, thus extending the scope to big numbers; naturally up to a cost in speed.

The reason for some strange looking expressions is to avoid arithmetic overflow.

```
\catcode`_ 11
\def\abortfactorize #1\xint_secondoftwo\fi #2#3{\fi}

\def\factorize #1.{\ifnum#1=1 \abortfactorize\fi
          \ifnum\numexpr #1-2=\numexpr ((#1/2)-1)*2\relax
               \expandafter\xint_firstoftwo
          \else\expandafter\xint_secondoftwo
```

```
           \fi
          {2&\expandafter\factorize\the\numexpr#1/2.}%
          {\factorize_b #1.3.}}%

\def\factorize_b #1.#2.{\ifnum#1=1 \abortfactorize\fi
          \ifnum\numexpr #1-(#2-1)*#2<#2
                 #1\abortfactorize
          \fi
          \ifnum \numexpr #1-#2=\numexpr ((#1/#2)-1)*#2\relax
               \expandafter\xint_firstoftwo
          \else\expandafter\xint_secondoftwo
          \fi
          {#2&\expandafter\factorize_b\the\numexpr#1/#2.#2.}%
          {\expandafter\factorize_b\the\numexpr #1\expandafter.%
                                    \the\numexpr #2+2.}}%
\catcode`_ 8
\begin{figure*}[ht!]
\centering\phantomsection\label{floatfactorize}\normalcolor
\tabskip1ex
\centeredline{\vbox{\halign {\hfil\strut#\hfil&&\hfil#\hfil\cr\noalign{\hrule}
          \xintiloop ["7FFFFFE0+1]
          \expandafter\bfseries\xintiloopindex &
          \ifnum\xintiloopindex="7FFFFFED
               \number"7FFFFFED\cr\noalign{\hrule}
          \expandafter\xintiloopskiptonext
          \fi
          \expandafter\factorize\xintiloopindex.\cr\noalign{\hrule}
          \ifnum\xintiloopindex<"7FFFFFFE
          \repeat
          \bfseries \number"7FFFFFFF&\number "7FFFFFFF\cr\noalign{\hrule}
}}}
\centeredline{A table of factorizations}
\end{figure*}
```

8.7. Another table of primes

As a further example, let us dynamically generate a tabular with the first 50 prime numbers after 12345. First we need a macro to test if a (short) number is prime. Such a completely expandable macro was given in subsection 8.3, here we consider a variant which will be slightly more efficient. This new \IsPrime has two parameters. The first one is a macro which it redefines to expand to the result of the primality test applied to the second argument. For convenience we use the etoolbox wrappers to various \ifnum tests, although here there isn't anymore the constraint of complete expandability (but using explicit \if..\fi in tabulars has its quirks); equivalent tests are provided by xint, but they have some overhead as they are able to deal with arbitrarily big integers.

```
\def\IsPrime #1#2% #1=\Result, #2=tested number (assumed >0).
{\edef\TheNumber {\the\numexpr #2}% hence #2 may be a count or \numexpr.
 \ifnumodd {\TheNumber}
 {\ifnumgreater {\TheNumber}{1}
  {\edef\ItsSquareRoot{\xintiiSqrt \TheNumber}%
    \xintFor ##1 in {\xintintegers [3+2]}\do
    {\ifnumgreater {##1}{\ItsSquareRoot} % ##1 is a \numexpr.
               {\def#1{1}\xintBreakFor}
```

2147483616	2	2	2	2	2	3	2731	8191	
2147483617	6733	318949							
2147483618	2	7	367	417961					
2147483619	3	3	23	353	29389				
2147483620	2	2	5	4603	23327				
2147483621	14741	145681							
2147483622	2	3	17	467	45083				
2147483623	79	967	28111						
2147483624	2	2	2	11	13	1877171			
2147483625	3	5	5	5	7	199	4111		
2147483626	2	19	37	1527371					
2147483627	47	53	862097						
2147483628	2	2	3	3	59652323				
2147483629	2147483629								
2147483630	2	5	6553	32771					
2147483631	3	137	263	19867					
2147483632	2	2	2	2	7	73	262657		
2147483633	5843	367531							
2147483634	2	3	12097	29587					
2147483635	5	11	337	115861					
2147483636	2	2	536870909						
2147483637	3	3	3	13	6118187				
2147483638	2	2969	361651						
2147483639	7	17	18046081						
2147483640	2	2	2	3	5	29	43	113	127
2147483641	2699	795659							
2147483642	2	23	46684427						
2147483643	3	715827881							
2147483644	2	2	233	1103	2089				
2147483645	5	19	22605091						
2147483646	2	3	3	7	11	31	151	331	
2147483647	2147483647								

A table of factorizations

```
                {}%
      \ifnumequal {\TheNumber}{(\TheNumber/##1)*##1}
                  {\def#1{0}\xintBreakFor }
                  {}%
     }}
   {\def#1{0}}}% 1 is not prime
  {\ifnumequal {\TheNumber}{2}{\def#1{1}}{\def#1{0}}}%
 }
```

As we used `\xintFor` inside a macro we had to double the `#` in its `#1` parameter. Here is now the code which creates the prime table (the table has been put in a float, which should be found on page 126):

```
 \newcounter{primecount}
 \newcounter{cellcount}
 \begin{figure*}[ht!]
   \centering
   \begin{tabular}{|*{7}c|}
   \hline
```

```
  \setcounter{primecount}{0}\setcounter{cellcount}{0}%
  \xintFor #1 in {\xintintegers [12345+2]} \do
% #1 is a \numexpr.
  {\IsPrime\Result{#1}%
   \ifnumgreater{\Result}{0}
   {\stepcounter{primecount}%
    \stepcounter{cellcount}%
    \ifnumequal {\value{cellcount}}{7}
       {\the#1 \\\setcounter{cellcount}{0}}
       {\the#1 &}}
   {}%
    \ifnumequal {\value{primecount}}{50}
     {\xintBreakForAndDo
      {\multicolumn {6}{l|}{These are the first 50 primes after 12345.}\\}}
     {}%
  }\hline
\end{tabular}
\end{figure*}
```

12347	12373	12377	12379	12391	12401	12409
12413	12421	12433	12437	12451	12457	12473
12479	12487	12491	12497	12503	12511	12517
12527	12539	12541	12547	12553	12569	12577
12583	12589	12601	12611	12613	12619	12637
12641	12647	12653	12659	12671	12689	12697
12703	12713	12721	12739	12743	12757	12763
12781	These are the first 50 primes after 12345.					

8.8. Factorizing again

Here is an *f*-expandable macro which computes the factors of an integer. It uses the xint macros only.

```
\catcode`\@ 11
\let\factorize\relax
\newcommand\Factorize [1]
      {\romannumeral0\expandafter\factorize\expandafter{\romannumeral-`0#1}}%
\newcommand\factorize [1]{\xintiiifOne{#1}{ 1}{\factors@a #1.{#1};}}%
\def\factors@a #1.{\xintiiifOdd{#1}
   {\factors@c 3.#1.}%
   {\expandafter\factors@b \expandafter1\expandafter.\romannumeral0\xinthalf{#1}.}}%
\def\factors@b #1.#2.{\xintiiifOne{#2}
   {\factors@end {2, #1}}%
   {\xintiiifOdd{#2}{\factors@c 3.#2.{2, #1}}%
                    {\expandafter\factors@b \the\numexpr #1+\@ne\expandafter.%
                        \romannumeral0\xinthalf{#2}.}}%
}%
\def\factors@c #1.#2.{%
    \expandafter\factors@d\romannumeral0\xintiidivision {#2}{#1}{#1}{#2}%
}%
\def\factors@d #1#2#3#4{\xintiiifNotZero{#2}
   {\xintiiifGt{#3}{#1}
        {\factors@end {#4, 1}}% ultimate quotient is a prime with power 1
```

```
          {\expandafter\factors@c\the\numexpr #3+\tw@.#4.}}%
     {\factors@e 1.#3.#1.}%
  }%
  \def\factors@e #1.#2.#3.{\xintiiifOne{#3}
     {\factors@end {#2, #1}}%
     {\expandafter\factors@f\romannumeral0\xintiidivision {#3}{#2}{#1}{#2}{#3}}%
  }%
  \def\factors@f #1#2#3#4#5{\xintiiifNotZero{#2}
     {\expandafter\factors@c\the\numexpr #4+\tw@.#5.{#4, #3}}%
     {\expandafter\factors@e\the\numexpr #3+\@ne.#4.#1.}%
  }%
  \def\factors@end #1;{\xintlistwithsep{, }{\xintRevWithBraces {#1}}}%
  \catcode`@ 12
```

The macro will be acceptably efficient only with numbers having somewhat small prime factors.

```
  \Factorize{16246355912554185673266068721806243461403654781833}
```

```
16246355912554185673266068721806243461403654781833, 13, 5, 17, 8, 29, 5, 37, 6, 41, 4, 59, 6
```

It puts a little stress on the input save stack in order not be bothered with previously gathered things.[41]

Its output is a comma separated list with the number first, then its prime factors with multiplicity. Let's produce something prettier:

```
  \catcode`_ 11
  \def\ShowFactors #1{\expandafter
                      \ShowFactors_a\romannumeral-`0\Factorize{#1},\relax,\relax,}
  \def\ShowFactors_a #1,{#1=\ShowFactors_b}
  \def\ShowFactors_b #1,#2,{\if\relax#1\else#1^{#2}\expandafter\ShowFactors_b\fi}
  \catcode`_ 8
  $$\ShowFactors{16246355912554185673266068721806243461403654781833}$$
```

$$16246355912554185673266068721806243461403654781833 = 13^5 17^8 29^5 37^6 41^4 59^6$$

If we only considered small integers, we could write pure \numexpr methods which would be very much faster (especially if we had a table of small primes prepared first) but still ridiculously slow compared to any non expandable implementation, not to mention use of programming languages directly accessing the CPU registers...

8.9. The Quick Sort algorithm illustrated

First a completely expandable macro which sorts a comma separated list of numbers.[42]

The \QSx macro expands its list argument, which may thus be a macro; its comma separated items must expand to integers or decimal numbers or fractions or scientific notation as acceptable to xintfrac, but if an item is itself some (expandable) macro, this macro will be expanded each time the item is considered in a comparison test! This is actually good if the macro expands in one step to the digits, and there are many many digits, but bad if the macro needs to do many computations. Thus \QSx should be used with either explicit numbers or with items being macros expanding in one step to the numbers (particularly if these numbers are very big).

If the interest is only in TEX integers, then one should replace the \xintifCmp macro with a suitable conditional, possibly helped by tools such as \ifnumgreater, \ifnumequal and \ifnumles
↪ s from etoolbox (LATEX only; I didn't see a direct equivalent to \xintifCmp.) Or, if we are dealing

[41] 2015/11/18 I have not revisited this code for a long time, and perhaps I could improve it now with some new techniques.

[42] The code in earlier versions of this manual handled inputs composed of braced items. I have switched to comma separated inputs on the occasion of (link removed) The version here is like code 3 on (link removed) (which is about 3x faster than the earlier code it replaced in this manual) with a modification to make it more efficient if the data has many repeated values. A faster routine (for sorting hundreds of values) is provided as code 6 at the link mentioned in the footnote, it is based on Merge Sort, but limited to inputs which one can handle as TEX dimensions.This code 6 could be extended to handle more general numbers, as acceptable by xintfrac. I have also written a non expandable version, which is even faster, but this matters really only when handling hundreds or rather thousands of values.

with decimal numbers with at most four+four digits, then one should use suitable \ifdim tests. Naturally this will boost consequently the speed, from having skipped all the overhead in parsing fractions and scientific numbers as are acceptable by xintfrac macros, and subsequent treatment.

```
% THE QUICK SORT ALGORITHM EXPANDABLY
% \usepackage{xintfrac} in the preamble (latex)
\makeatletter
% use extra safe delimiters
\catcode`! 3 \catcode`? 3
\def\QSx {\romannumeral0\qsx }%
% first we check if empty list (else \qsx@finish will not find a comma)
\def\qsx   #1{\expandafter\qsx@a\romannumeral-`0#1,!,?}%
\def\qsx@a #1{\ifx,#1\expandafter\qsx@abort\else
                     \expandafter\qsx@start\fi #1}%
\def\qsx@abort #1?{ }%
\def\qsx@start {\expandafter\qsx@finish\romannumeral0\qsx@b,}%
\def\qsx@finish ,#1{ #1}%
%
% we check if empty of single and if not pick up the first as Pivot:
\def\qsx@b ,#1#2,#3{\ifx?#3\xintdothis\qsx@empty\fi
                    \ifx!#3\xintdothis\qsx@single\fi
                    \xintorthat\qsx@separate {#1#2}{}{}{#1#2}#3}%
\def\qsx@empty  #1#2#3#4#5{ }%
\def\qsx@single #1#2#3#4#5?{, #4}%
\def\qsx@separate #1#2#3#4#5#6,%
{%
    \ifx!#5\expandafter\qsx@separate@done\fi
    \xintifCmp {#5#6}{#4}%
          \qsx@separate@appendtosmaller
          \qsx@separate@appendtoequal
          \qsx@separate@appendtogreater {#5#6}{#1}{#2}{#3}{#4}%
}%
%
\def\qsx@separate@appendtoequal   #1#2{\qsx@separate {#2,#1}}%
\def\qsx@separate@appendtogreater #1#2#3{\qsx@separate {#2}{#3,#1}}%
\def\qsx@separate@appendtosmaller #1#2#3#4{\qsx@separate {#2}{#3}{#4,#1}}%
%
\def\qsx@separate@done\xintifCmp #1%
          \qsx@separate@appendtosmaller
          \qsx@separate@appendtoequal
          \qsx@separate@appendtogreater #2#3#4#5#6#7?%
{%
    \expandafter\qsx@f\expandafter {\romannumeral0\qsx@b #4,!,?}{\qsx@b #5,!,?}{#3}%
}%
%
\def\qsx@f #1#2#3{#2, #3#1}%
%
\catcode`! 12 \catcode`? 12
\makeatother

% EXAMPLE
\begingroup
\edef\z {\QSx {1.0, 0.5, 0.3, 1.5, 1.8, 2.0, 1.7, 0.4, 1.2, 1.4,
```

```
                1.3, 1.1, 0.7, 1.6, 0.6, 0.9, 0.8, 0.2, 0.1, 1.9}}
\meaning\z

\def\a {3.123456789123456789}\def\b {3.123456789123456788}
\def\c {3.123456789123456790}\def\d {3.123456789123456787}
\oodef\z {\QSx { \a, \b, \c, \d}}%
% The space before \a to let it not be expanded during the conversion from CSV
% values to List. The \oodef expands exactly twice (via a bunch of \expandafter's)
\meaning\z
\endgroup
```

macro:->0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2.0

macro:->\d , \b , \a , \c (the spaces after \d, etc... come from the use of the \meaning primitive.)

The choice of pivot as first element is bad if the list is already almost sorted. Let's add a variant which will pick up the pivot index randomly. The previous routine worked also internally with comma separated lists, but for a change this one will use internally lists of braced items (the initial conversion via \xintCSVtoList handles all potential spurious space problems).

```
% QuickSort expandably on comma separated values with random choice of pivots
% \usepackage{xintfrac, xinttools} in preamble
\makeatletter
\def\QSx {\romannumeral0\qsx }% This is a f-expandable macro.
% This converts from comma separated values on input and back on output.
% **** NOTE: these steps (and the other ones too, actually) are costly if input
%            has thousands of items.
\def\qsx #1{\xintlistwithsep{, }%
            {\expandafter\qsx@sort@a\expandafter{\romannumeral0\xintcsvtolist{#1}}}}%
%
% we check if empty or single or double and if not pick up the first as Pivot:
\def\qsx@sort@a #1%
    {\expandafter\qsx@sort@b\expandafter{\romannumeral0\xintlength{#1}}{#1}}%
\def\qsx@sort@b #1{\ifcase #1
                     \expandafter\qsx@sort@empty
                     \or\expandafter\qsx@sort@single
                     \or\expandafter\qsx@sort@double
                     \else\expandafter\qsx@sort@c\fi {#1}}%
\def\qsx@sort@empty  #1#2{ }%
\def\qsx@sort@single #1#2{#2}%
\catcode`_ 11
\def\qsx@sort@double #1#2{\xintifGt #2{\xint_exchangetwo_keepbraces}{}#2}%
\catcode`_ 8
\def\qsx@sort@c      #1#2{%
    \expandafter\qsx@sort@sep@a\expandafter
                {\romannumeral0\xintnthelt{\xintUniformDeviate #1+\@ne}{#2}}#2?}%
\def\qsx@sort@sep@a #1{\qsx@sort@sep@loop {}{}{}{#1}}%
\def\qsx@sort@sep@loop #1#2#3#4#5%
{%
    \ifx?#5\expandafter\qsx@sort@sep@done\fi
    \xintifCmp {#5}{#4}%
          \qsx@sort@sep@appendtosmaller
          \qsx@sort@sep@appendtoequal
          \qsx@sort@sep@appendtogreater {#5}{#1}{#2}{#3}{#4}%
```

```
}%
%
\def\qsx@sort@sep@appendtoequal   #1#2{\qsx@sort@sep@loop {#2{#1}}}%
\def\qsx@sort@sep@appendtogreater #1#2#3{\qsx@sort@sep@loop {#2}{#3{#1}}}%
\def\qsx@sort@sep@appendtosmaller #1#2#3#4{\qsx@sort@sep@loop {#2}{#3}{#4{#1}}}%
%
\def\qsx@sort@sep@done\xintifCmp #1%
          \qsx@sort@sep@appendtosmaller
          \qsx@sort@sep@appendtoequal
          \qsx@sort@sep@appendtogreater #2#3#4#5#6%
{%
    \expandafter\qsx@sort@recurse\expandafter
                {\romannumeral0\qsx@sort@a {#4}}{\qsx@sort@a {#5}}{#3}%
}%
%
\def\qsx@sort@recurse #1#2#3{#2#3#1}%
%
\makeatother

% EXAMPLES
\begingroup
\edef\z {\QSx {1.0, 0.5, 0.3, 1.5, 1.8, 2.0, 1.7, 0.4, 1.2, 1.4,
               1.3, 1.1, 0.7, 1.6, 0.6, 0.9, 0.8, 0.2, 0.1, 1.9}}
\meaning\z

\def\a {3.123456789123456789}\def\b {3.123456789123456788}
\def\c {3.123456789123456790}\def\d {3.123456789123456787}
\oodef\z {\QSx { \a, \b, \c, \d}}%
% The space before \a to let it not be expanded during the conversion from CSV
% values to List. The \oodef expands exactly twice (via a bunch of \expandafter's)
\meaning\z

\def\somenumbers{%
3997.6421, 8809.9358, 1805.4976, 5673.6478, 3179.1328, 1425.4503, 4417.7691,
2166.9040, 9279.7159, 3797.6992, 8057.1926, 2971.9166, 9372.2699, 9128.4052,
1228.0931, 3859.5459, 8561.7670, 2949.6929, 3512.1873, 1698.3952, 5282.9359,
1055.2154, 8760.8428, 7543.6015, 4934.4302, 7526.2729, 6246.0052, 9512.4667,
7423.1124, 5601.8436, 4433.5361, 9970.4849, 1519.3302, 7944.4953, 4910.7662,
3679.1515, 8167.6824, 2644.4325, 8239.4799, 4595.1908, 1560.2458, 6098.9677,
3116.3850, 9130.5298, 3236.2895, 3177.6830, 5373.1193, 5118.4922, 2743.8513,
8008.5975, 4189.2614, 1883.2764, 9090.9641, 2625.5400, 2899.3257, 9157.1094,
8048.4216, 3875.6233, 5684.3375, 8399.4277, 4528.5308, 6926.7729, 6941.6278,
9745.4137, 1875.1205, 2755.0443, 9161.1524, 9491.1593, 8857.3519, 4290.0451,
2382.4218, 3678.2963, 5647.0379, 1528.7301, 2627.8957, 9007.9860, 1988.5417,
2405.1911, 5065.8063, 5856.2141, 8989.8105, 9349.7840, 9970.3013, 8105.4062,
3041.7779, 5058.0480, 8165.0721, 9637.7196, 1795.0894, 7275.3838, 5997.0429,
7562.6481, 8084.0163, 3481.6319, 8078.8512, 2983.7624, 3925.4026, 4931.5812,
1323.1517, 6253.0945}%

\oodef\z {\QSx \somenumbers}% produced as a comma+space separated list
% black magic as workaround to the shrinkability of spaces in last line...
\hsize 87\fontcharwd\font`0
```

```
  \lccode`~=32
  \lowercase{\def~}{\discretionary{}{}{\kern\fontcharwd\font`0}}\catcode32 13
  \noindent\phantom{00}\scantokens\expandafter{\meaning\z}\par
  \endgroup
  macro:->0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7,
1.8, 1.9, 2.0
  macro:->\d , \b , \a , \c
  macro:->1055.2154, 1228.0931, 1323.1517, 1425.4503, 1519.3302, 1528.7301, 1560.2458, 1698.3952,
1795.0894, 1805.4976, 1875.1205, 1883.2764, 1988.5417, 2166.9040, 2382.4218, 2405.1911,
2625.5400, 2627.8957, 2644.4325, 2743.8513, 2755.0443, 2899.3257, 2949.6929, 2971.9166,
2983.7624, 3041.7779, 3116.3850, 3177.6830, 3179.1328, 3236.2895, 3481.6319, 3512.1873,
3678.2963, 3679.1515, 3797.6992, 3859.5459, 3875.6233, 3925.4026, 3997.6421, 4189.2614,
4290.0451, 4417.7691, 4433.5361, 4528.5308, 4595.1908, 4910.7662, 4931.5812, 4934.4302,
5058.0480, 5065.8063, 5118.4922, 5282.9359, 5373.1193, 5601.8436, 5647.0379, 5673.6478,
5684.3375, 5856.2141, 5997.0429, 6098.9677, 6246.0052, 6253.0945, 6926.7729, 6941.6278,
7275.3838, 7423.1124, 7526.2729, 7543.6015, 7562.6481, 7944.4953, 8008.5975, 8048.4216,
8057.1926, 8078.8512, 8084.0163, 8105.4062, 8165.0721, 8167.6824, 8239.4799, 8399.4277,
8561.7670, 8760.8428, 8809.9358, 8857.3519, 8989.8105, 9007.9860, 9090.9641, 9128.4052,
9130.5298, 9157.1094, 9161.1524, 9279.7159, 9349.7840, 9372.2699, 9491.1593, 9512.4667,
9637.7196, 9745.4137, 9970.3013, 9970.4849
```

All the previous examples were with numbers which could have been handled via \ifdim tests rather than the \xintifCmp macro from xintfrac; using \ifdim tests would naturally be faster. Even faster routine is code 6 at (link removed) which uses \pdfescapestring and a Merge Sort algorithm.

We then turn to a graphical illustration of the algorithm.[43] For simplicity the pivot is always chosen as the first list item. Then we also give a variant which picks up the last item as pivot.

```
% in LaTeX preamble:
% \usepackage{xintfrac, xinttools}
% \usepackage{color}
% or, when using Plain TeX:
% \input xintfrac.sty \input xinttools.sty
% \input color.tex
%
% Color definitions
\definecolor{LEFT}{RGB}{216,195,88}
\definecolor{RIGHT}{RGB}{208,231,153}
\definecolor{INERT}{RGB}{199,200,194}
\definecolor{INERTpiv}{RGB}{237,237,237}
\definecolor{PIVOT}{RGB}{109,8,57}
% Start of macro defintions
\makeatletter
% \catcode`? 3 % a bit too paranoid. Normal ? will do.
%
% argument will never be empty
\def\QS@cmp@a    #1{\QS@cmp@b  #1??}%
\def\QS@cmp@b    #1{\noexpand\QS@sep@A\@ne{#1}\QS@cmp@d {#1}}%
\def\QS@cmp@d    #1#2{\ifx ?#2\expandafter\QS@cmp@done\fi
                      \xintifCmp {#1}{#2}\tw@\@ne\z@{#2}\QS@cmp@d {#1}}%
\def\QS@cmp@done #1?{?}%
%
```

[43] I have rewritten (2015/11/21) the routine to do only once (and not thrice) the needed calls to \xintifCmp, up to the price of one additional \edef, although due to the context execution time on our side is not an issue and moreover is anyhow overwhelmed by the TikZ's activities. Simultaneously I have updated the code. The variant with the choice of pivot on the right has more overhead: the reason is simply that we do not convert the data into an array, but maintain a list of tokens with self-reorganizing delimiters.

```
\def\QS@sep@A #1?{\QSLr\QS@sep@L #1\thr@@?#1\thr@@?#1\thr@@?}%
\def\QS@sep@L #1#2{\ifcase #1{#2}\or\or\else
                                        \expandafter\QS@sep@I@start\fi \QS@sep@L}%
\def\QS@sep@I@start\QS@sep@L {\noexpand\empty?\QSIr\QS@sep@I}%
\def\QS@sep@I #1#2{\ifcase#1\or{#2}\or\else\expandafter\QS@sep@R@start\fi\QS@sep@I}%
\def\QS@sep@R@start\QS@sep@I {\noexpand\empty?\QSRr\QS@sep@R}%
\def\QS@sep@R #1#2{\ifcase#1\or\or{#2}\else\expandafter\QS@sep@done\fi\QS@sep@R}%
\def\QS@sep@done\QS@sep@R {\noexpand\empty?}%
%
\def\QS@loop {%
    \xintloop
    % pivot phase
    \def\QS@pivotcount{0}%
    \let\QSLr\DecoLEFTwithPivot  \let\QSIr \DecoINERT
    \let\QSRr\DecoRIGHTwithPivot \let\QSIrr\DecoINERT
    \centerline{\QS@list}%
    % sorting phase
    \ifnum\QS@pivotcount>\z@
            \def\QSLr {\QS@cmp@a}\def\QSRr {\QS@cmp@a}%
            \def\QSIr {\QSIrr}\let\QSIrr\relax
                \edef\QS@list{\QS@list}% compare
            \let\QSLr\relax\let\QSRr\relax\let\QSIr\relax
                \edef\QS@list{\QS@list}% separate
            \def\QSLr ##1##2?{\ifx\empty##1\else\noexpand \QSLr {{##1}##2}\fi}%
            \def\QSIr ##1##2?{\ifx\empty##1\else\noexpand \QSIr {{##1}##2}\fi}%
            \def\QSRr ##1##2?{\ifx\empty##1\else\noexpand \QSRr {{##1}##2}\fi}%
                \edef\QS@list{\QS@list}% gather
            \let\QSLr\DecoLEFT \let\QSRr\DecoRIGHT
            \let\QSIr\DecoINERTwithPivot \let\QSIrr\DecoINERT
            \centerline{\QS@list}%
    \repeat }%
%
% \xintFor* loops handle gracefully empty lists.
\def\DecoLEFT  #1{\xintFor* ##1 in {#1} \do {\colorbox{LEFT}{##1}}}%
\def\DecoINERT #1{\xintFor* ##1 in {#1} \do {\colorbox{INERT}{##1}}}%
\def\DecoRIGHT #1{\xintFor* ##1 in {#1} \do {\colorbox{RIGHT}{##1}}}%
\def\DecoPivot #1{\begingroup
                    \color{PIVOT}\advance\fboxsep-\fboxrule\fbox{#1}\endgroup}%
%
\def\DecoLEFTwithPivot #1{\xdef\QS@pivotcount{\the\numexpr\QS@pivotcount+\@ne}%
    \xintFor* ##1 in {#1} \do
        {\xintifForFirst {\DecoPivot {##1}}{\colorbox{LEFT}{##1}}}}%
\def\DecoINERTwithPivot #1{\xdef\QS@pivotcount{\the\numexpr\QS@pivotcount+\@ne}%
    \xintFor* ##1 in {#1} \do
        {\xintifForFirst {\colorbox{INERTpiv}{##1}}{\colorbox{INERT}{##1}}}}%
\def\DecoRIGHTwithPivot #1{\xdef\QS@pivotcount{\the\numexpr\QS@pivotcount+\@ne}%
    \xintFor* ##1 in {#1}  \do
        {\xintifForFirst {\DecoPivot {##1}}{\colorbox{RIGHT}{##1}}}}%
%
\def\QuickSort #1{% warning: not compatible with empty #1.
    % initialize, doing conversion from comma separated values
    % to a list of braced items
```

```
    \edef\QS@list{\noexpand\QSRr{\xintCSVtoList{#1}}}%
    % may \edef's are to follow anyhow
% earlier I did a first drawing of the list, here with the color of RIGHT elements,
% but the color should have been for example white, anyway I drop this first line
    %\let\QSRr\DecoRIGHT
    %\par\centerline{\QS@list}%
%
    % loop as many times as needed
    \QS@loop }%
%
% \catcode`? 12 % in case we had used a funny ? as delimiter.
\makeatother
%% End of macro definitions.
%% Start of Example
\begingroup\offinterlineskip
\small
% \QuickSort {1.0, 0.5, 0.3, 1.5, 1.8, 2.0, 1.7, 0.4, 1.2, 1.4,
%               1.3, 1.1, 0.7, 1.6, 0.6, 0.9, 0.8, 0.2, 0.1, 1.9}
% \medskip
% with repeated values
\QuickSort {1.0, 0.5, 0.3, 0.8, 1.5, 1.8, 2.0, 1.7, 0.4, 1.2, 1.4,
              1.3, 1.1, 0.7, 0.3, 1.6, 0.6, 0.3, 0.8, 0.2, 0.8, 0.7, 1.2}
\endgroup
```

1.0	0.5	0.3	0.8	1.5	1.8	2.0	1.7	0.4	1.2	1.4	1.3	1.1	0.7	0.3	1.6	0.6	0.3	0.8	0.2	0.8	0.7	1.2
0.5	0.3	0.8	0.4	0.7	0.3	0.6	0.3	0.8	0.2	0.8	0.7	1.0	1.5	1.8	2.0	1.7	1.2	1.4	1.3	1.1	1.6	1.2
0.5	0.3	0.8	0.4	0.7	0.3	0.6	0.3	0.8	0.2	0.8	0.7	1.0	1.5	1.8	2.0	1.7	1.2	1.4	1.3	1.1	1.6	1.2
0.3	0.4	0.3	0.3	0.2	0.5	0.8	0.7	0.6	0.8	0.8	0.7	1.0	1.2	1.4	1.3	1.1	1.2	1.5	1.8	2.0	1.7	1.6
0.3	0.4	0.3	0.3	0.2	0.5	0.8	0.7	0.6	0.8	0.8	0.7	1.0	1.2	1.4	1.3	1.1	1.2	1.5	1.8	2.0	1.7	1.6
0.2	0.3	0.3	0.3	0.4	0.5	0.7	0.6	0.7	0.8	0.8	0.8	1.0	1.1	1.2	1.2	1.4	1.3	1.5	1.7	1.6	1.8	2.0
0.2	0.3	0.3	0.3	0.4	0.5	0.7	0.6	0.7	0.8	0.8	0.8	1.0	1.1	1.2	1.2	1.4	1.3	1.5	1.7	1.6	1.8	2.0
0.2	0.3	0.3	0.3	0.4	0.5	0.6	0.7	0.7	0.8	0.8	0.8	1.0	1.1	1.2	1.2	1.3	1.4	1.5	1.6	1.7	1.8	2.0
0.2	0.3	0.3	0.3	0.4	0.5	0.6	0.7	0.7	0.8	0.8	0.8	1.0	1.1	1.2	1.2	1.3	1.4	1.5	1.6	1.7	1.8	2.0
0.2	0.3	0.3	0.3	0.4	0.5	0.6	0.7	0.7	0.8	0.8	0.8	1.0	1.1	1.2	1.2	1.3	1.4	1.5	1.6	1.7	1.8	2.0
0.2	0.3	0.3	0.3	0.4	0.5	0.6	0.7	0.7	0.8	0.8	0.8	1.0	1.1	1.2	1.2	1.3	1.4	1.5	1.6	1.7	1.8	2.0

Here is the variant which always picks the pivot as the rightmost element.

```
\makeatletter
%
\def\QS@cmp@a #1{\noexpand\QS@sep@A\expandafter\QS@cmp@d\expandafter
                 {\romannumeral0\xintnthelt{-1}{#1}}#1??}%
%
\def\DecoLEFTwithPivot #1{\xdef\QS@pivotcount{\the\numexpr\QS@pivotcount+\@ne}%
    \xintFor* ##1 in {#1} \do
        {\xintifForLast {\DecoPivot {##1}}{\colorbox{LEFT}{##1}}}}
\def\DecoINERTwithPivot #1{\xdef\QS@pivotcount{\the\numexpr\QS@pivotcount+\@ne}%
    \xintFor* ##1 in {#1} \do
        {\xintifForLast {\colorbox{INERTpiv}{##1}}{\colorbox{INERT}{##1}}}}
\def\DecoRIGHTwithPivot #1{\xdef\QS@pivotcount{\the\numexpr\QS@pivotcount+\@ne}%
    \xintFor* ##1 in {#1}  \do
        {\xintifForLast {\DecoPivot {##1}}{\colorbox{RIGHT}{##1}}}}
\def\QuickSort #1{%
    % initialize, doing conversion from comma separated values
    % to a list of braced items
    \edef\QS@list{\noexpand\QSLr {\xintCSVtoList{#1}}}%
```

```
    % many \edef's are to follow anyhow
    %
    % loop as many times as needed
    \QS@loop }%
\makeatother
\begingroup\offinterlineskip
\small
% \QuickSort {1.0, 0.5, 0.3, 1.5, 1.8, 2.0, 1.7, 0.4, 1.2, 1.4,
%               1.3, 1.1, 0.7, 1.6, 0.6, 0.9, 0.8, 0.2, 0.1, 1.9}
% \medskip
% with repeated values
\QuickSort {1.0, 0.5, 0.3, 0.8, 1.5, 1.8, 2.0, 1.7, 0.4, 1.2, 1.4,
              1.3, 1.1, 0.7, 0.3, 1.6, 0.6, 0.3, 0.8, 0.2, 0.8, 0.7, 1.2}
\endgroup
```

1.0 0.5 0.3 0.8 1.5 1.8 2.0 1.7 0.4 1.2 1.4 1.3 1.1 0.7 0.3 1.6 0.6 0.3 0.8 0.2 0.8 0.7 1.2
1.0 0.5 0.3 0.8 0.4 1.1 0.7 0.3 0.6 0.3 0.8 0.2 0.8 0.7 1.2 1.2 1.5 1.8 2.0 1.7 1.4 1.3 1.6
1.0 0.5 0.3 0.8 0.4 1.1 0.7 0.3 0.6 0.3 0.8 0.2 0.8 0.7 1.2 1.2 1.5 1.8 2.0 1.7 1.4 1.3 1.6
0.5 0.3 0.4 0.3 0.6 0.3 0.2 0.7 0.7 1.0 0.8 1.1 0.8 0.8 1.2 1.2 1.5 1.4 1.3 1.6 1.8 2.0 1.7
0.5 0.3 0.4 0.3 0.6 0.3 0.2 0.7 0.7 1.0 0.8 1.1 0.8 0.8 1.2 1.2 1.5 1.4 1.3 1.6 1.8 2.0 1.7
0.2 0.5 0.3 0.4 0.3 0.6 0.3 0.7 0.7 0.8 0.8 0.8 1.0 1.1 1.2 1.2 1.3 1.5 1.4 1.6 1.7 1.8 2.0
0.2 0.5 0.3 0.4 0.3 0.6 0.3 0.7 0.7 0.8 0.8 0.8 1.0 1.1 1.2 1.2 1.3 1.5 1.4 1.6 1.7 1.8 2.0
0.2 0.3 0.3 0.3 0.5 0.4 0.6 0.7 0.7 0.8 0.8 0.8 1.0 1.1 1.2 1.2 1.3 1.4 1.5 1.6 1.7 1.8 2.0
0.2 0.3 0.3 0.3 0.5 0.4 0.6 0.7 0.7 0.8 0.8 0.8 1.0 1.1 1.2 1.2 1.3 1.4 1.5 1.6 1.7 1.8 2.0
0.2 0.3 0.3 0.3 0.5 0.4 0.6 0.7 0.7 0.8 0.8 0.8 1.0 1.1 1.2 1.2 1.3 1.4 1.5 1.6 1.7 1.8 2.0
0.2 0.3 0.3 0.3 0.5 0.4 0.6 0.7 0.7 0.8 0.8 0.8 1.0 1.1 1.2 1.2 1.3 1.4 1.5 1.6 1.7 1.8 2.0
0.2 0.3 0.3 0.3 0.4 0.5 0.6 0.7 0.7 0.8 0.8 0.8 1.0 1.1 1.2 1.2 1.3 1.4 1.5 1.6 1.7 1.8 2.0
0.2 0.3 0.3 0.3 0.4 0.5 0.6 0.7 0.7 0.8 0.8 0.8 1.0 1.1 1.2 1.2 1.3 1.4 1.5 1.6 1.7 1.8 2.0
0.2 0.3 0.3 0.3 0.4 0.5 0.6 0.7 0.7 0.8 0.8 0.8 1.0 1.1 1.2 1.2 1.3 1.4 1.5 1.6 1.7 1.8 2.0
0.2 0.3 0.3 0.3 0.4 0.5 0.6 0.7 0.7 0.8 0.8 0.8 1.0 1.1 1.2 1.2 1.3 1.4 1.5 1.6 1.7 1.8 2.0

The choice of the first or last item as pivot is not a good one as nearly ordered lists will take quadratic time. But for explaining the algorithm via a graphical interpretation, it is not that bad. If one wanted to pick up the pivot randomly, the routine would have to be substantially rewritten: in particular the \Deco..withPivot macros need to know where the pivot is, and currently this is implemented by using either \xintifForFirst or \xintifForLast.

Part II.
The macro layer for expandable computations: xintcore, xint, xintfrac, and some extras

```
  WARNING !
  The documentation is getting old, and is in need of rewrites for many sections, particularly
for examples.
  We do try to keep updated the description of macros provided by the packages.
```

9 Introduction: the xint bundle 135

10 Macros of the xintkernel package 150

11 Macros of the xintcore package 154

12 Macros of the xint package 159

13 Macros of the xintfrac package 171

14 Macros of the xintbinhex package 202

15 Macros of the xintgcd package 208

16 Macros of the xintseries package 211

17 Macros of the xintcfrac package 227

9. Introduction: the xint bundle

.1 Characteristics 135
.2 Floating point evaluations 137
.3 Expansion matters 138
.4 Input formats for macros 140
.5 Output formats of macros 142
.6 Count registers and variables 142
.7 Dimension registers and variables 143
.8 \ifcase, \ifnum, ... constructs 144
.9 No variable declarations are needed 145
.10 Possible syntax errors to avoid 145
.11 Error messages 146
.12 Package namespace, catcodes 148
.13 Origins of the package 148

9.1. Characteristics

```
  The main characteristics are:

1. exact algebra on ``big numbers'', integers as well as fractions,
```

2. floating point variants with user-chosen precision,

3. the computational macros are compatible with expansion-only context,

4. the bundle comes with parsers (integer-only, or handling fractions, or doing floating point computations) of infix operations implementing beyond infix operations extra features such as dummy variables.

Since 1.2 ``big numbers'' must have less than about 19950 digits: the maximal number of digits for addition is at 19968 digits, and it is 19959 for multiplication. The reasonable range of use of the package is with numbers of up to a few hundred digits.[44]
TEX does not know off-hand how to print on the page such very long numbers, see subsection 2.8.

Integers with only 10 digits and starting with a 3 already exceed the TEX bound; and TEX does not have a native processing of floating point numbers (multiplication by a decimal number of a dimension register is allowed --- this is used for example by the pgf basic math engine.)

TEX elementary operations on numbers are done via the non-expandable *\advance*, *\multiply*, and *\divide* assignments. This was changed with ε-TEX's \numexpr which does expandable computations using standard infix notations with TEX integers. But ε-TEX did not modify the TEX bound on acceptable integers, and did not add floating point support.

The bigintcalc package by Heiko Oberdiek provided expandable macros (using some of \numexpr possibilities, when available) on arbitrarily big integers, beyond the TEX bound. It does not provide an expression parser.[45] xint did it again using more of \numexpr for higher speed, and in a later evolution added handling of exact fractions, of scientific numbers, and an expression parser. Arbitrary precision floating points operations were added as a derivative, and not part of the initial design goal.

The concept of signed infinities, signed zeroes, NaN's, error traps...,[46] have not been implemented, only the notion of `scientific notation with a given number of significant figures'.[47]

The LATEX3 project has implemented expandably floating-point computations with 16 significant figures (l3fp), including functions such as exp, log, sine and cosine.[48]

More directly related to the xint bundle there is the l3bigint package, also devoted to big integers and in development a.t.t.o.w (2015/10/09, no division yet). It is part of the experimental trunk of the LATEX3 Project and provides an expression parser for expandable arithmetic with big integers. Its author Bruno Le Floch succeeded brilliantly into implementing expandably the Karatsuba multiplication algorithm and he achieves *sub-quadratic growth for the computation time*. This shows up very clearly with numbers having thousands of digits, up to the maximum which a.t.t.o.w is at 8192 digits.

The l3bigint multiplication from late 2015 is observed to be roughly 3x--4x faster than the one from \xintiiexpr in the range of 4000 to 5000 digits integers, and isn't far from being 9x faster at 8000 digits. On the other hand \xintiiexpr's multiplication is found to be on average roughly

[44] For example multiplication of integers having from 50 to 100 digits takes roughly of the order of the millisecond on a 2012 desktop computer. I compared this to using Python3: using timeit module on a wrapper defined as `return w*z` with random integers of 100 digits, I observe on the same computer a computation time of roughly 4.10^{-7}s per call. And with `return str(w*z)` then this becomes more like 16.10^{-7}s per call. And with `return str(int(W)*int(Z))` where `W` and `Z` are strings, this becomes about 26.10^{-7}s (I am deliberately ignoring Python's Decimal module here...) Anyway, my sentence from earlier version of this documentation: *this is, I guess, at least about 1000 times slower than what can be expected with any reasonable programming language*, is about right. I then added: *nevertheless as compilation of a typical LATEX document already takes of the order of seconds and even dozens of seconds for long ones, this leaves room for reasonably many computations via xintexpr or via direct use of the macros of xint/xintfrac.* [45] One can currently use package bnumexpr to associate the bigintcalc macros with an expression parser. This may be unavailable in future if bnumexpr becomes more tightly associated with future evolutions or variants of xintcore. EDIT: still possible as of bnumexpr 1.6 2025/09/01. [46] The latter exist as work-in-progress for some time in the source code. [47] Multiplication of two floats with P=\xinttheDigits digits is first done exactly then rounded to P digits, rather than using a specially tailored multiplication for floating point numbers which would be more efficient (it is a waste to evaluate fully the multiplication result with 2P or 2P-1 digits.) [48] at the time of writing (2014/10/28) the l3fp (exactly represented) floating point numbers have their exponents limited to ±9999.

2.5x faster than l3bigint's for numbers up to 100 digits and the two packages achieve about the same speed at 900 digits: but each such multiplication of numbers of 900 digits costs about one or two tenths of a second on a 2012 desktop computer, whereas the order of magnitude is rather the ms for numbers with 50--100 digits.[49]

Even with the superior l3bigint Karatsuba multiplication it takes about 3.5s on this 2012 desktop computer for a single multiplication of two 5000-digits numbers. Hence it is not possible to do routinely such computations in a document. I have long been thinking that without the expandability constraint much higher speeds could be achieved, but perhaps I have not given enough thought to sustain that optimistic stance.[50]

I remain of the opinion that if one really wants to do computations with *thousands* of digits, one should drop the expandability requirement. Indeed, as clearly demonstrated long ago by the pi computing file by D. Roegel one can program TEX to compute with many digits at a much higher speed than what xint achieves: but, direct access to memory storage in one form or another seems a necessity for this kind of speed and one has to renounce at the complete expandability.[51]

9.2. Floating point evaluations

WARNING: THIS PART OF THE DOCUMENTATION IS IN DIRE NEED OF BEING REVISED AND MAY BE EVEN MORE BADLY OBSOLETE THAN THE REST OF THIS DOCUMENTATION.

Floating point macros are provided by package xintfrac to work with a given arbitrary precision P. The default value is P = 16 meaning that the significands of the produced (non-zero) numbers have 16 decimal digits. The syntax to set the precision to P is

```
\xintDigits:=P\relax
```

The value is local to the group or environment (if using LaTeX). To query the current value use \xinttheDigits.

Most floating point macros accept an optional first argument [P] which then sets the target precision and replaces the \xintDigits assigned value (the [P] must be repeated if the arguments are themselves xintfrac macros with arguments of their own.) In this section P refers to the prevailing \xinttheDigits float precision or to the target precision set in this way as an optional argument.

\xintfloatexpr[Q]...\relax also admits an optional argument [Q] but it has an altogether different meaning: the computations are always done with the prevailing \xinttheDigits precision and the optional argument Q is used for the final rounding. This makes sense only if Q<\xinttheDigits and is intended to clean up the result from dubious last digits (when Q<0 it indicates rather by how many digits one should reduce the mantissa lengths via a final rounding).

The IEEE 754[52] requirement of *correct rounding* for addition, subtraction, multiplication, division and square root is achieved (in arbitrary precision) by the macros of xintfrac hence also by the infix operators +, -, *, /.

This means that for operands given with at most P significant digits (and arbitrary exponents) the output coincides exactly with the rounding of the exact theoretical result (barring overflow or underflow).

Due to a typographical oversight, this documentation (up to 1.2j) adjoined ^ and ** to the above list of infix operators. But as is explained in subsection 13.98, what is guaranteed regarding integer powers by the legacy macros

[49] I have tested this again on 2016/12/19, but the macros have not changed on the l3bigint side and barely on the xintcore side, hence I got again the same results... [50] The apnum package implements (non-expandably) arbitrary precision fixed point algebra and (v1.6) functions exp, log, sqrt, the trigonometrical direct and inverse functions. [51] The LuaTEX project possibly makes endeavours such as xint appear even more insane that they are, in truth: xint is able to handle fast enough computations involving numbers with less than one hundred digits and brings this to all engines.

is an error of at most 0.52ulp, not the correct rounding. The native integer-power routine used by \xintfloateval at 2.0a2 does not guarantee correct rounding either.

The rounding mode is ``round to nearest, ties away from zero''. It is not customizable.

Currently xintfrac has no notion of NaNs or signed infinities or signed zeroes, but this is intended for the future.

Since release 1.2f, square root extraction achieves correct rounding in arbitrary precision.

The maximal floating point decimal exponent is currently 2147483647 which is the maximal number handled by TEX. The minimal exponent is its opposite. But this means that overflow or underflow are detected only via low-level \numexpr arithmetic overflows which are basically un-recoverable. Besides there are some border effects as the routines need to add or subtract lengths of numbers from exponents, possibly triggering the low-level overflows. In the future not only the Precision but also the maximal and minimal exponents Emin and Emax will be specifiable by the user.

Since 1.2f, the float macros round their inputs to the target precision P before further processing. Formerly, the initial rounding was done to P+2 digits (and at least P+3 for the power operation.)

The more ambitious model would be for the computing macros to obey the intrinsic precision of their inputs, i.e. to compute the correct rounding to P digits of the exact mathematical result corresponding to inputs allowed to have their own higher precision.[53] This would be feasible by xintfrac which after all knows how to compute exactly, but I have for the time being decided that for reasons of efficiency, the chosen model is the one of rounding inputs to the target precision first.

The float macros of xintfrac have to handle inputs which not only may have much more digits than the target float precision, but may even be fractions: in a way this means infinite precision.

From releases 1.08a to 1.2j a fraction input AeM/BeN had its numerator and denominator A and B truncated to Q+2 digits of precision, then the substituted fraction was correctly rounded to Q digits of precision (usually with Q set to P+2) and then the operation was implemented on such rounded inputs. But this meant that two fractions representing the same rational number could end up being rounded differently (with a difference of one unit in the last place), if it had numerators and denominators with at least Q+3 digits.

Starting with release 1.2k a fractional input AeM/BeN is handled intrinsically: the fraction, independently of its representation AeM/BeN, is *correctly rounded* to P digits during the input parsing. Hence the output depends only on its arguments as mathematical fractions and not on their representatives as quotients.

Notice that in float expressions, the / is treated as operator, and is applied to arguments which are generally already P-floats, hence the above discussion becomes relevant in this context only for the special input form qfloat(A/B) or when using a sub-expression \xintexpr A/B\relax embedded in the float expression with A or B having more digits than the prevailing float precision P.

9.3. Expansion matters

9.3.1. Full expansion of the first token

The whole business of xint is to build upon \numexpr and handle arbitrarily large numbers. Each basic operation is thus done via a macro: \xintiiAdd, \xintiiSub, \xintiiMul, \xintiiDivision. In order to handle more complex operations, it must be possible to nest these macros. An expandable macro can not execute a \def or an \edef. But the macro must expand its arguments to find the

[52] The IEEE 754-1985 standard was for hardware implementations of binary floating-point arithmetic with a specific value for the precision (24 bits for single precision, 53 bits for double precision). The newer IEEE 754-2008 (https://en.wikipedia.org/wiki/IEEE_floating_point) normalizes five basic formats, three binaries and two decimals (16 and 34 decimal digits) and discusses extended formats with higher precision. These standards are only indirectly relevant to libraries like xint dealing with arbitrary precision. [53] The MPFR library http://www.mpfr.org/ implements this but it does not know fractions!

digits it is supposed to manipulate. TEX provides a tool to do the job of (expandable !) repeated expansion of the first token found until hitting something non expandable, such as a digit, a \def token, a brace, a \count token, etc... is found. A space token also will stop the expansion (and be swallowed, contrarily to the non-expandable tokens).

By convention in this manual *f*-expansion (``full expansion'' or ``full first expansion'') will be this TEX process of expanding repeatedly the first token seen. For those familiar with LATEX3 (which is not used by xint) this is what is called in its documentation full expansion, whereas expansion inside \edef would be described I think as ``exhaustive'' expansion and will be referred too in this manual as *x*-expansion.

Most of the package macros, and all those dealing with computations[54], are expandable in the strong sense that they expand to their final result via this *f*-expansion. This will be signaled ★ in their descriptions via a star in the margin.

These macros not only have this property of *f*-expandability, they all begin by first applying *f* *f*-expansion to their arguments. Again from LATEX3's conventions this will be signaled by a margin annotation next to the description of the arguments.

9.3.2. Summary of important expandability aspects

1. the macros *f*-expand their arguments, this means that they expand the first token seen (for each argument), then expand, etc..., until something un-expandable such as a digit or a brace is hit against. This example
   ```
   \def\x{98765}\def\y{43210} \xintiiAdd {\x}{\x\y}
   ```
   is *not* a legal construct, as the \y will remain untouched by expansion and not get converted into the digits which are expected by the sub-routines of \xintiiAdd. It is a \numexpr which will expand it and an arithmetic overflow will arise as 9876543210 exceeds the TEX bounds. The same would hold for \xintAdd.

   To the contrary \xinttheiiexpr and others have no issues with things such as \xinttheiiexpr \x+\x\y\relax.

2. using \if...\fi constructs *inside* the package macro arguments requires suitably mastering TEXniques (\expandafter's and/or swapping techniques) to ensure that the *f*-expansion will indeed absorb the \else or closing \fi, else some error will arise in further processing. Therefore it is highly recommended to use the package provided conditionals such as \xintifEq, \xintifGt, \xintifSgn,... or, for LATEX users and when dealing with short integers the etoolbox[55] expandable conditionals (for small integers only) such as \ifnumequal, \ifnumgreater, ... Use of *non-expandable* things such as \ifthenelse is impossible inside the arguments of xint macros.

   One can use naive \if..\fi things inside an \xinttheexpr-ession and cousins, as long as the test is expandable, for example
   ```
   \xinttheiexpr\ifnum3>2 143\else 33\fi 0^2\relax→2044900=1430^2
   ```

3. after the definition \def\x {12}, one can not use -\x as input to one of the package macros: the *f*-expansion will act only on the minus sign, hence do nothing. The only way is to use the \xintOpp macro (or \xintiiOpp which is integer only) which obtains the opposite of a given number.

   Again, this is otherwise inside an \xinttheexpr-ession or \xintthefloatexpr-ession. There, the minus sign may prefix macros which will expand to numbers (or parentheses etc...)

4. With the definition
   ```
   \def\AplusBC #1#2#3{\xintAdd {#1}{\xintMul {#2}{#3}}}
   ```
   one obtains an expandable macro producing the expected result, not in two, but rather in three steps: a first expansion is consumed by the macro expanding to its definition. As the package

[54] except \xintXTrunc. [55] https://ctan.org/pkg/etoolbox

macros expand their arguments until no more is possible (regarding what comes first), this \AplusBC may be used inside them: \xintAdd {\AplusBC {1}{2}{3}}{4} does work and returns 11/1[0].

If, for some reason, it is important to create a macro expanding in two steps to its final value, one may either do:

```
\def\AplusBC #1#2#3{\romannumeral-`0\xintAdd {#1}{\xintMul {#2}{#3}}}
```

or use the *lowercase* form of \xintAdd:

```
\def\AplusBC #1#2#3{\romannumeral0\xintadd {#1}{\xintMul {#2}{#3}}}
```

and then \AplusBC will share the same properties as do the other xint `primitive' macros.

5. The \romannumeral0 and \romannumeral-`0 things above look like an invitation to hacker's territory; if it is not important that the macro expands in two steps only, there is no reason to follow these guidelines. Just chain arbitrarily the package macros, and the new ones will be completely expandable and usable one within the other.

   Since release 1.07 the \xintNewExpr macro automatizes the creation of such expandable macros:

   ```
   \xintNewExpr\AplusBC[3]{#1+#2*#3}
   ```

   creates the \AplusBC macro doing the above and expanding in two expansion steps.

6. In the expression parsers of xintexpr such as \xintexpr..\relax, \xintfloatexpr..\relax the contents are expanded completely from left to right until the ending \relax is found and swallowed, and spaces and even (to some extent) catcodes do not matter.

7. For all variants, prefixing with \xintthe allows to print the result or use it in other contexts. Shortcuts \xinttheexpr, \xintthefloatexpr, \xinttheiiexpr, ... are available.

9.4. Input formats for macros

Macros can have different types of arguments (we do not consider here the \xintexpr-parsers but only the macros of xintcore/xint/xintfrac). In a macro description, a margin annotation signals what is the argument type.

num *x*

1. TEX integers are handled inside a \numexpr..\relax hence may be count registers or variables. Beware that -(1+1) is not legal and raises an error, but 0-(1+1) is. Also 2\cnta with \cnta a \count isn't legal. Integers must be kept less than 2147483647 in absolute value, although the *scaling* operation (a*b)/c computes the intermediate product with twice as many bits.

   The slash / does a rounded division which is a fact of life of \numexpr which I have found very annoying in at least nine cases out of ten, not to say ninety-nine cases out of one hundred. Besides, it is at odds with TEX's \divide which does a truncated division (non-expandably).

   But to follow-suit / also does rounded integer division in \xintiiexpr..\relax, and the operator // does there the truncated division.

f

2. the strict format applies to macros handling big integers but only *f*-expanding their arguments. After this *f*-expansion the input should be a string of digits, optionally preceded by a unique minus sign. The first digit can be zero only if it is the only digit. A plus sign is not accepted. -0 is not legal in the strict format. Macros of xint with a double ii require this `strict' format for the inputs.

Num *f*

3. the extended integer format applies when the macro parses its arguments via \xintNum. The input may then have arbitrarily many leading minus and plus signs, followed by leading zeroes, and further digits. With xintfrac loaded, \xintNum is extended to accept fractions and its action is to truncate them to integers.

Frac f

4. the fraction input format applies to the arguments of xintfrac macros handling genuine fractions. It allows two types of inputs: general and restricted. The restricted type is parsed faster, but... is restricted.

   general: inputs of the shape A.BeC/D.EeF. Example:

```
 \noindent\xintRaw{+--0367.8920280e17/-++278.289287e-15}\newline
 \xintRaw{+--+1253.2782e++--3/---0087.123e---5}\par
-3678920280/278289287[31]
-12532782/87123[7]
```

   The input parser does not reduce fractions to smallest terms. Here are the rules of this general fraction format:

   - everything is optional, absent numbers are treated as zero, here are some extreme cases:

```
 \xintRaw{}, \xintRaw{.}, \xintRaw{./1.e}, \xintRaw{-.e}, \xintRaw{e/-1}
0/1[0], 0/1[0], 0/1[0], 0/1[0], 0/1[0]
```

   - AB and DE may start with pluses and minuses, then leading zeroes, then digits.
   - C and F will be given to \numexpr and can be anything recognized as such and not provoking arithmetic overflow (the lengths of B and E will also intervene to build the final exponent naturally which must obey the TEX bound).
   - the /, . (numerator and/or denominator) and e (numerator and/or denominator) are all optional components.
   - each of A, B, C, D, E and F may arise from *f*-expansion of a macro.
   - the whole thing may arise from *f*-expansion, however the /, ., and e should all come from this initial expansion. The e of scientific notation is mandatorily lowercased.

   restricted: inputs either of the shape A[N] or A/B[N], which represents the fraction A/B times 10^N. The whole thing or each of A, B, N (but then not / or [) may arise from *f*-expansion, A (after expansion) *must* have a unique optional minus sign and no leading zeroes, B (after expansion) if present *must* be a positive integer with no signs and no leading zeroes, [N] if present will be given to \numexpr. Any deviation from the rules above will result in errors.

Frac f

   Notice that *, + and - contrarily to the / (which is treated simply as a kind of delimiter) are not acceptable within arguments of this type (see subsection 9.6 for some exceptions to this.)

Generally speaking, there should be no spaces among the digits in the inputs (in arguments to the package macros). Although most would be harmless in most macros, there are some cases where spaces could break havoc.[56] So the best is to avoid them entirely.

This is entirely otherwise inside an \xintexpr-ession, where spaces are ignored (except when they occur inside arguments to some macros, thus escaping the \xintexpr parser). See the section 3.

[56] The \xintNum macro does not remove spaces between digits beyond the first non zero ones; however this should not really alter the subsequent functioning of the arithmetic macros, and besides, since xintcore 1.2 there is an initial parsing of the entire number, during which spaces will be gobbled. However I have not done a complete review of the legacy code to be certain of all possibilities after 1.2 release. One thing to be aware of is that \numexpr stops on spaces between digits (although it provokes an expansion to see if an infix operator follows); the exponent for \xintiiPow or the argument of the factorial \xintiiFac are only subjected to such a \numexpr (there are a few other macros with such input types in xint). If the input is given as, say 1 2\x where \x is a macro, the macro \x will not be expanded by the \numexpr, and this will surely cause problems afterwards. Perhaps a later xint will force \numexpr to expand beyond spaces, but I decided that was not really worth the effort. Another immediate cause of problems is an input of the type \xintiiAdd {<space>\x }{\y }, because the space will stop the initial expansion; this will most certainly cause an arithmetic overflow later when the \x will be expanded in a \numexpr. Thus in conclusion, damages due to spaces are unlikely if only explicit digits are involved in the inputs, or arguments are single macros with no preceding space.

There are also some slighly more obscure expansion types: in particular, the \xintApplyInline and \xintFor* macros from xinttools apply a special iterated *f*-expansion, which gobbles spaces, to the non-braced items (braced items are submitted to no expansion because the opening brace stops it) coming from their list argument; this is denoted by a special symbol in the margin. Some other macros such as \xintSum from xintfrac first do an *f*-expansion, then treat each found (braced or not) item (skipping spaces between such items) via the general fraction input parsing, this is signaled as here in the margin where the signification of the * is thus a bit different from the previous case.

*f

Frac
f⟶ * f

A few macros from xinttools do not expand, or expand only once their argument. This is also signaled in the margin with notations à la LaTeX3.

n, resp. o

9.5. Output formats of macros

We do not consider here the \xintexpr-parsers but only the macros from xintcore, xint and xintfrac. Macros of other components of the bundle may have their own output formats, for example for continuous fractions with xintcfrac. There are mainly three types of outputs:

- arithmetic macros from xintcore/xint deliver integers in the strict format as described in the previous section.
- arithmetic macros from xintfrac produce on output the strict fraction format A/B[N], which stands for (A/B)×10^N, where A and B are integers, B is positive, and N is a ``short'' integer. The output is not reduced to smallest terms. The A and B may end with zeroes (*i.e*, N does not represent all powers of ten). The denominator B is always strictly positive. There is no + sign. The - is always first if present (i.e. the denominator on output is always positive.) The output will be expressed as such a fraction even if the inputs are both integers and the mathematical result is an integer. The B=1 is not removed.[57]
- macros from xintfrac having Float in their names deliver a number in the scientific notation as described in the documentation of \xintFloat.

  The exception is \xintPFloat which does some customizable pretty printing of the result.

9.6. Count registers and variables

Inside \xintexpr..\relax and its variants, a count register or count control sequence is automatically unpacked using \number, with tacit multiplication: 1.23\counta is like 1.23*\number\counta. There is a subtle difference between count *registers* and count *variables*. In 1.23*\counta the unpacked \counta variable defines a complete operand thus 1.23*\counta 7 is a syntax error. But 1.23*\count0 just replaces \count0 by \number\count0 hence 1.23*\count0 7 is like 1.23*57 if \count0 contains the integer value 5.

Regarding now the package macros, there is first the case of arguments having to be short integers: this means that they are fed to a \numexpr...\relax, hence submitted to a *complete expansion* which must deliver an integer, and count registers and even algebraic expressions with them like \mycountA+\mycountB*17-\mycountC/12+\mycountD are admissible arguments (the slash stands here for the rounded integer division done by \numexpr). This applies in particular to the number of digits to truncate or round with, to the indices of a series partial sum, . . .

The macros allowing the extended format for long numbers or dealing with fractions will *to some extent* allow the direct use of count registers and even infix algebra inside their arguments: a count register \mycountA or \count 255 is admissible as numerator or also as denominator, with no need to be prefixed by \the or \number. It is possible to have as argument an algebraic expression as would be acceptable by a \numexpr...\relax, under this condition: *each of the numerator and*

[57] refer to the documentation of \xintPRaw for an alternative.

denominator is expressed with at most nine *tokens*.[58] [59] Important: a slash for rounded division in a \numexpr should be written with braces {/} to not be confused with the xintfrac delimiter between numerator and denominator (braces will be removed internally and the slash will count for one token). Example: \mycountA+\mycountB{/}17/1+\mycountA*\mycountB, or \count 0+\count 2{/}17/1+\count 0*\count 2.

```
\cnta 10 \cntb 35 \xintRaw {\cnta+\cntb{/}17/1+\cnta*\cntb}->12/351[0]
```

For longer algebraic expressions using count registers, there are two possibilities:

1. let the numerator and the denominator be presented as \the\numexpr...\relax,
2. or as \numexpr {...}\relax (the braces are removed during processing; they are not legal for \numexpr...\relax syntax.)

```
\cnta 100 \cntb 10 \cntc 1
\xintPRaw {\numexpr {\cnta*\cnta+\cntb*\cntb+\cntc*\cntc+
                     2*\cnta*\cntb+2*\cnta*\cntc+2*\cntb*\cntc}\relax/%
           \numexpr {\cnta*\cnta+\cntb*\cntb+\cntc*\cntc}\relax }
12321/10101
```

9.7. Dimension registers and variables

⟨*dimen*⟩ variables can be converted into (short) integers suitable for the xint macros by prefixing them with \number. This transforms a dimension into an explicit short integer which is its value in terms of the sp unit (1/65536 pt). When \number is applied to a ⟨*glue*⟩ variable, the stretch and shrink components are lost.

For LaTeX users: a length is a ⟨*glue*⟩ variable, prefixing a length macro defined by \newlength with \number will thus discard the plus and minus glue components and return the dimension component as described above, and usable in the xint bundle macros.

This conversion is done automatically inside an \xintexpr-essions, with tacit multiplication implied if prefixed by some (integral or decimal) number.

One may thus compute areas or volumes with no limitations, in units of sp^2 respectively sp^3, do arithmetic with them, compare them, etc..., and possibly express some final result back in another unit, with the suitable conversion factor and a rounding to a given number of decimal places.

A table of dimensions illustrates that the internal values used by TeX do not correspond always to the closest rounding. For example a millimeter exact value in terms of sp units is 72.27/10/2.54*65536=186467.981... and TeX uses internally 186467sp (TeX truncates to get an integral multiple of the sp unit; see at the end of this section the exact rules applied internally by TeX).

There is something quite amusing with the Didot point. According to the TeXBook, 1157 dd=1238 pt. The actual internal value of 1 dd in TeX is 70124 sp. We can use xintcfrac to display the list of centered convergents of the fraction 70124/65536:

```
\xintListWithSep{, }{\xintFtoCCv{70124/65536}}
```

1/1, 15/14, 61/57, 107/100, 1452/1357, 17531/16384, and we don't find 1238/1157 therein, but another approximant 1452/1357!

And indeed multiplying 70124/65536 by 1157, and respectively 1357, we find the approximations (wait for more, later):

```
``1157 dd''=1237.998474121093...pt
``1357 dd''=1451.999938964843...pt
```

and we seemingly discover that 1357 dd=1452 pt is *far more accurate* than the TeXBook formula 1157 dd=1238 pt ! The formula to compute N dd was

```
\xinttheexpr trunc(N\dimexpr 1dd\relax/\dimexpr 1pt\relax,12)\relax}
```

[58] The 1.2k and earlier versions manual claimed up to 8 tokens, but low-level TeX error arose if the \numexpr ...\relax occupied exactly 8 tokens *and* evaluated to zero. With 1.21 and later, up to 9 tokens are always safe and one may even drop the ending \relax. But well, all these explanations are somewhat silly because prefixing by \the or \number is always working with arbitrarily many tokens. [59] Attention! in the LaTeX context a \value{countername} will behave ok only if it is first in the input, if not it will not get expanded, and braces around the name will be removed and chaos will ensue inside a \numexpr. One should enclose the whole input in \the\numexpr...\relax in such cases.

Unit	definition	Exact value in sp units	TEX's value in sp units	Relative error
cm	0.01 m	236814336/127 = 1.864679811e6...	1864679	0%
mm	0.001 m	118407168/635 = 186467.981...	186467	-0.0005%
in	2.54 cm	118407168/25 = 4.73628672e6...	4736286	0%
pc	12 pt	786432 = 786432...	786432	0%
pt	1/72.27 in	65536 = 65536...	65536	0%
bp	1/72 in	1644544/25 = 65781.76...	65781	-0.0012%
3bp	1/24 in	4933632/25 = 197345.28...	197345	-0.0001%
12bp	1/6 in	19734528/25 = 789381.12...	789381	0%
72bp	1 in	118407168/25 = 4.73628672e6...	4736286	0%
dd	1238/1157 pt	81133568/1157 = 70124.086...	70124	-0.0001%
11dd	11*1238/1157 pt	892469248/1157 = 771364.95...	771364	-0.0001%
12dd	12*1238/1157 pt	973602816/1157 = 841489.037...	841489	0%
sp	1/65536 pt	1 = 1...	1	0%

TEX dimensions

What's the catch? The catch is that TEX *does not* compute 1157 dd like we just did:

```
1157 dd=\number\dimexpr 1157dd\relax/65536=1238.000000000000...pt
1357 dd=\number\dimexpr 1357dd\relax/65536=1452.001724243164...pt
```

We thus discover that TEX (or rather here, e-TEX, but one can check that this works the same in TEX82), uses 1238/1157 as a conversion factor (and necessarily intermediate computations simulate higher precision than a priori available with integers less than 2^{31} or rather 2^{30} for dimensions). Hence the 1452/1357 ratio is irrelevant, an artefact of the rounding (or rather, as we see, truncating) for one dd to be expressed as an integral number of sp's.

Let us now use \xintexpr to compute the value of the Didot point in millimeters, if the above rule is exactly verified:

```
\xinttheexpr trunc(1238/1157*25.4/72.27,12)\relax=0.376065027442...mm
```

This fits very well with the possible values of the Didot point as listed in the Wikipedia Article. The value 0.376065 mm is said to be *the traditional value in European printers' offices*. So the 1157 dd=1238 pt rule refers to this Didot point, or more precisely to the *conversion factor* to be used between this Didot and TEX points.

The actual value in millimeters of exactly one Didot point as implemented in TEX is

```
\xinttheexpr trunc(\dimexpr 1dd\relax/65536/72.27*25.4,12)\relax
=0.376064563929...mm
```

The difference of circa 5Å is arguably tiny!

By the way the *European printers' offices* (dixit Wikipedia) *Didot* is thus exactly

```
\xinttheexpr reduce(.376065/(25.4/72.27))\relax=543564351/508000000 pt
```

and the centered convergents of this fraction are 1/1, 15/14, 61/57, 107/100, 1238/1157, 11249/10513, 23736/22183, 296081/276709, 615898/575601, 11382245/10637527, 22148592/20699453, 1885709/176233151, 543564351/508000000. We do recover the 1238/1157 therein!

> Here is how TEX converts abc.xyz...<unit>. First the decimal is *rounded* to the nearest integral multiple of 1/65536, say X/65536. The <unit> is associated to a ratio N/D, which represents <unit>/pt. For the Didot point the ratio is indeed 1238/1157. TEX *truncates* the fraction XN/D to an integer M. The dimension is represented by M sp.

9.8. \ifcase, \ifnum, ... constructs

When using things such as \ifcase \xintSgn{\A} one has to make sure to leave a space after the closing brace for TEX to stop its scanning for a number: once TEX has finished expanding \xintSgn{

↪ \A} and has so far obtained either 1, 0, or -1, a space (or something `unexpandable') must stop it looking for more digits. Using \ifcase\xintSgn\A without the braces is very dangerous, because the blanks (including the end of line) following \A will be skipped and not serve to stop the number which \ifcase is looking for.

```
\begin{enumerate}[nosep]\def\A{1}
\item \ifcase \xintSgn\A 0\or OK\else ERROR\fi
\item \ifcase \xintSgn\A\space 0\or OK\else ERROR\fi
\item \ifcase \xintSgn{\A} 0\or OK\else ERROR\fi
\end{enumerate}
```

1. ERROR
2. OK
3. OK

In order to use successfully \if...\fi constructions either as arguments to the xint bundle expandable macros, or when building up a completely expandable macro of one's own, one needs some TEXnical expertise (see also item 2 on page 139).

It is thus much to be recommended to use the expandable branching macros, provided by xintfrac succh as \xintifSgn, \xintifZero, \xintifOne, \xintifNotZero, \xintifTrueAelseB, \xintifCmp, \xintifGt, \xintifLt, \xintifEq, \xintifInt... See their respective documentations. All these conditionals always have either two or three branches, and empty brace pairs {} for unused branches should not be forgotten.

If these tests are to be applied to standard TEX short integers, it is more efficient to use (under LATEX) the equivalent conditional tests from the etoolbox[60] package.

9.9. No variable declarations are needed

There is no notion of a *declaration of a variable*.

To do a computation and assign its result to some macro \z, the user will employ the \def, \edef, or \newcommand (in LATEX) as usual, keeping in mind that two expansion steps are needed, thus \edef is initially the main tool:

```
\def\x{1729728} \def\y{352827927} \edef\z{\xintiiMul {\x}{\y}}
\meaning\z
```

```
macro:->610296344513856
```

As an alternative to \edef the package provides \oodef which expands exactly twice the replacement text, and \fdef which applies *f*-expansion to the replacement text during the definition.

```
\def\x{1729728}\def\y{352827927}
\oodef\w {\xintiiMul\x\y} \fdef\z{\xintiiMul {\x}{\y}}
\meaning\w, \meaning\z
```

```
macro:->610296344513856, macro:->610296344513856
```

In practice \oodef is slower than \edef, except for computations ending in very big final replacement texts (thousands of digits). On the other hand \fdef appears to be slightly faster than \edef already in the case of expansions leading to only a few dozen digits.

xintexpr does provide an interface to declare and assign values to identifiers which can then be used in expressions: subsection 3.9.

9.10. Possible syntax errors to avoid

Here is a list of imaginable input errors. Some will cause compilation errors, others are more annoying as they may pass through unsignaled.

- using - to prefix some macro: -\xintiiSqr{35}/271.[61]

[60] https://ctan.org/pkg/etoolbox [61] to the contrary, this *is* allowed inside an \xintexpr-ession.

- using one pair of braces too many \xintIrr{{\xintiiPow {3}{13}}/243} (the computation goes through with no error signaled, but the result is completely wrong).
- things like \xintiiAdd { \x}{\y} as the space will cause \x to be expanded later, most probably within a \numexpr thus provoking possibly an arithmetic overflow.
- using [] and decimal points at the same time 1.5/3.5[2], or with a sign in the denominator 3/-5[7]. The scientific notation has no such restriction, the two inputs 1.5/-3.5e-2 and -1.5e2/3.5 are equivalent: \xintRaw{1.5/-3.5e-2}=-15/35[2], \xintRaw{-1.5e2/3.5}=-15/35[2].
- generally speaking, using in a context expecting an integer (possibly restricted to the TEX bound) a macro or expression which returns a fraction: \xinttheexpr 4/2\relax outputs 4/2, not 2. Use \xintNum {\xinttheexpr 4/2\relax} or \xinttheiexpr 4/2\relax (which rounds the result to the nearest integer, here, the result is already an integer) or \xinttheiiexpr 4/2\relax. Or, divide in your head 4 by 2 and insert the result directly in the TEX source.

9.11. Error messages

In situations such as division by zero, the TEX run will be interrupted with some error message. It conveys some short information on the cause of the problem,[62] then an optimistic statement about a possible recovery if the user (in interactive mode) simply hits the <return> key. In non-interactive (nonstopmode) the TEX run goes on uninterrupted and the error data will be found in the compilation log. Often, xint will fall-back to using a zero value. This is still an experimental feature.[63] [64]

The encouragements will be slightly better formatted if the run is with LATEX compared to Plain ε-TEX: Plain by default does not set the \newlinechar which allows to issue linebreaks in messages at chosen locations. In the examples here, xintsession is used, and it loads xint in a way activating the nicer \newlinechar formatted messages, even though it runs (a priori, but not necessarily) under Plain ε-TEX.

```
>>> 1/0;
Runaway argument?
! xint error: Division by zero: 1/0.
! Paragraph ended before \xint<...> is done, but will resume:
  hit <return> at the ? prompt to try fixing the error above
  which has been encountered before expansion was complete.
<to be read again>
                   \par
...
l.602 \xintsession
                  \endinput%^^M
?
@_1     0
>>> (-1)^3.2;
Runaway argument?
! xint error: Fractional power 32/1[-1] of negative -1[0].
! Paragraph ended before \xint<...> is done, but will resume:
  hit <return> at the ? prompt to try fixing the error above
  which has been encountered before expansion was complete.
<to be read again>
                   \par
```

CHANGED AT 1.4m!

[62] The wording of these messages has been last modified at 1.4m. [63] Customizable handlers, error traps, error flags are implemented in embryonic form but without user interface since 1.2l release. This is not ready yet. [64] The 1.4g new formatting implementation benefited from a May 2021 thread at the LATEX3 site where expandable error messages were discussed, with in particular contributions of @blefloch and @Skillmon.

```
...
l.602 \xintsession
                  \endinput%^^M
?
@_2     0
>>> cos 1);
Runaway argument?
! xint error: `cos1' unknown, say `Isome_var' or I use 0.
! Paragraph ended before \xint<...> is done, but will resume:
  hit <return> at the ? prompt to try fixing the error above
  which has been encountered before expansion was complete.
<to be read again>
                   \par
...
l.602 \xintsession
                  \endinput%^^M
?
Runaway argument?
! xint error: Extra ) removed. Hit <return>, fingers crossed.
! Paragraph ended before \xint<...> is done, but will resume:
  hit <return> at the ? prompt to try fixing the error above
  which has been encountered before expansion was complete.
<to be read again>
                   \par
...
l.602 \xintsession
                  \endinput%^^M
?
@_3     0
>>> 3=4;
Runaway argument?
! xint error: Expected an operator but got `='. Ignoring.
! Paragraph ended before \xint<...> is done, but will resume:
  hit <return> at the ? prompt to try fixing the error above
  which has been encountered before expansion was complete.
<to be read again>
                   \par
...
l.602 \xintsession
                  \endinput%^^M
?
@_4     12
>>> &bye
```

In the last example, tacit multiplication was applied as xintexpr was looking for an operator, got some invalid input and then a number.

Some constructs in xintexpr-essions use delimited macros and there is thus possibility in case of an ill-formed expression to end up beyond the \relax end-marker. Such a situation can also occur from \relax being swallowed by a non-terminated \numexpr:

```
\xintexpr 3 + \numexpr 5+4\relax followed by some LaTeX code...
```

The correct input is

```
\xintexpr 3 + \numexpr 5+4\relax\relax
```

But people in their right mind will have done

```
\xintexpr 3 + 5 + 4\relax
```

A few will have done the computation in their heads.

In such cases low-level errors will arise and may lead to very cryptic messages; but nothing unusual or especially traumatizing for the daring experienced TEX/LATEX user, whose has seen zillions of un-helpful error messages already in her daily practice of TEX/LATEX.

9.12. Package namespace, catcodes

This section reviews (probably with some omissions) important miscellany regarding control sequence names and catcode matters and is basically in its entirety a

TEX-hackers note:

- The bundle packages force the \space and \empty control sequences into having their default meanings as in Plain TEX or LATEX2e formats.
- Private macros (or internally used \count registers, and one private \toks) have names starting with \xint_ or \XINT_. Some, for legacy or technical reasons, have \xint or \XINT prefix with no underscore.
- All public macros have their names starting with \xint except for: the xintkernel provided \odef, \oodef, \fdef. If macros with these names already exist xinttools will not overwrite them. Their meanings are also available under the names \xintodef, \xintoodef, etc...
- For the xintfrac macros to be able to parse their inputs, standard catcodes in the argument are assumed for the digits (of course), the plus and minus signs, the dot, the letter e, the forward slash, the square brackets. Spaces should be avoided although they may go unnoticed sometimes.
- For the xintexpr expressions there is more leeway: the digit tokens must have their standard catcodes, the letters must have their standard catcodes for variable and function names to be recognized, but other characters may mostly have unusual (but not extreme like catcode zero or one) catcodes. Active characters will be expanded and should usually be prefixed with \string. But, if activated via Babel this is not needed.

  NEW WITH 1.4n

  A few syntax elements are implemented via delimited macros. So the comma, the equal sign and the closing parenthesis must have their normal catcodes for these syntax elements to work. \string won't do.

  Spaces are gobbled. The e of scientific notation may be E on input, xintfrac macros on the other hand will not recognize the E.
- \xintdefvar and \xintdeffunc as they use automatically \xintexprSafeCatcodes and \xintexprRestoreCatcodes to temporarily set catcodes to safe values.
- \xintexprSafeCatcodes and \xintexprRestoreCatcodes can be employed at user level too.
- At loading time the catcode configuration may be arbitrary as long as it satisfies the following requirements:
  - % has its normal category code,
  - \ has its normal category code,
  - Latin letters have their normal category code "letter",
  - Digits have their normal category code "other".

  Nothing more is assumed, for example { and } may have unusual catcodes at package loading time. This will be admittedly unusual especially in LATEX as \usepackage{xintexpr} would then have had to be replaced by something such as \usepackage<xintexpr>...
- Loading the packages causes no insertion of space tokens.
- The previous two items also apply to usage of \xintreloadxintlog and of \xintreloadxinttrig.

9.13. Origins of the package

2013/03/28. Package bigintcalc by Heiko Oberdiek already provides expandable arithmetic operations on "big integers", i.e. integers beyond the TEX bound $2^{31}-1$, so why another[65] one?

[65] this section was written before the xintfrac package; the author is not aware of another package allowing expandable computations with arbitrarily big fractions.

I got started on this in early March 2013, via a thread on the c.t.tex usenet group, where Ulrich Diez used the previously cited package together with a macro (\ReverseOrder) which I had contributed to another thread.[66] What I had learned in this other thread thanks to interaction with Ulrich Diez and GL on expandable manipulations of tokens motivated me to try my hands at addition and multiplication.

I wrote macros \bigMul and \bigAdd which I posted to the newsgroup; they appeared to work comparatively fast. These first versions did not use the ε-TeX \numexpr primitive, they worked one digit at a time, having previously stored carry-arithmetic in 1200 macros.

I noticed that the bigintcalc package used \numexpr if available, but (as far as I could tell) not to do computations many digits at a time. Using \numexpr for one digit at a time for \bigAdd and \bigMul slowed them a tiny bit but avoided cluttering TeX memory with the 1200 macros storing precomputed digit arithmetic. I wondered if some speed could be gained by using \numexpr to do four digits at a time for elementary multiplications (as the maximal admissible number for \numexpr has ten digits).

2013/04/14. This initial xint was followed by xintfrac which handled exactly fractions and decimal numbers.

2013/05/25. Later came xintexpr and at the same time xintfrac got extended to handle floating point numbers.

2013/11/22. Later, xinttools was detached.

2014/10/28. Release 1.1 significantly extended the xintexpr parsers.

2015/10/10. Release 1.2 rewrote the core integer routines which had remained essentially unmodified, apart from a slight improvement of division early 2014.

This 1.2 release also got its impulse from a fast ``reversing'' macro, which I wrote after my interest got awakened again as a result of correspondence with Bruno Le Floch during September 2015: this new reverse uses a TeXnique which *requires* the tokens to be digits. I wrote a routine which works (expandably) in quasi-linear time, but a less fancy O(N^2) variant which I developed concurrently proved to be faster all the way up to perhaps 7000 digits, thus I dropped the quasi-linear one. The less fancy variant has the advantage that xint can handle numbers with more than 19900 digits (but not much more than 19950). This is with the current common values of the input save stack and maximal expansion depth: 5000 and 10000 respectively.

[66] the \ReverseOrder could be avoided in that circumstance, but it does play a crucial rôle here.

10. Macros of the xintkernel package

.1 Introduction 150
.2 \odef, \oodef, \fdef 150
.3 \xintReverseOrder 150
.4 \xintLength 151
.5 \xintFirstItem 151
.6 \xintLastItem 151
.7 \xintFirstOne 151
.8 \xintLastOne 151
.9 \xintReplicate, \xintreplicate 151
.10 \xintGobble, \xintgobble 152
.11 (WIP) \xintUniformDeviate 152

10.1. Introduction

The xintkernel package contains mainly the common code base for handling the load-order of the bundle packages, the management of catcodes at loading time, definition of common constants and macro utilities which are used throughout the code etc ... it is automatically loaded by all packages of the bundle.

It provides a few macros possibly useful in other contexts.

10.2. \odef, \oodef, \fdef

code *code* *code*

\oodef\controlsequence {<stuff>} does

```
    \expandafter\expandafter\expandafter\def
    \expandafter\expandafter\expandafter\controlsequence
    \expandafter\expandafter\expandafter{<stuff>}
```

This works only for a single \controlsequence, with no parameter text, even without parameters. An alternative would be:

```
\def\oodef #1#{\def\oodefparametertext{#1}%
               \expandafter\expandafter\expandafter\expandafter
               \expandafter\expandafter\expandafter\def
               \expandafter\expandafter\expandafter\oodefparametertext
               \expandafter\expandafter\expandafter }
```

but it does not allow \global as prefix, and, besides, would have anyhow its use (almost) limited to parameter texts without macro parameter tokens (except if the expanded thing does not see them, or is designed to deal with them).

There is a similar macro \odef with only one expansion of the replacement text <stuff>, and \fdef which expands fully <stuff> using \romannumeral-`0.

They can be prefixed with \global. It appears than \fdef is generally a bit faster than \edef when expanding macros from the xint bundle, when the result has a few dozens of digits. \oodef needs thousands of digits it seems to become competitive.

xintkernel will not define these macros if the control sequence names already exist. It provides them always under the names \xintodef, \xintoodef and \xintfdef respectively.

10.3. \xintReverseOrder

code

n ★ \xintReverseOrder{⟨*list*⟩} does not do any expansion of its argument and just reverses the order of the tokens in the ⟨*list*⟩. Braces are removed once and the enclosed material, now unbraced, does not get reversed. Unprotected spaces (of any character code) are gobbled.

```
\xintReverseOrder{\xintDigitsOf\xintiiPow {2}{100}\to\Stuff}
gives: \Stuff\to1002\xintiiPow\xintDigitsOf
```

xinttools provides a variant \xintRevWithBraces which keeps brace pairs in the output, and *f*-expands its input first.

For inputs consisting only digit tokens, see \xintReverseDigits from xint.

10.4. \xintLength

code

n ⋆ \xintLength{⟨list⟩} counts how many tokens (or braced items) there are (possibly none). It does no expansion of its argument, so to use it to count things in the replacement text of a macro \x one should do \expandafter\xintLength\expandafter{\x}. Blanks between items are not counted. See also \xintNthElt{0} (from xinttools) which first f-expands its argument and then applies the same code.

```
\xintLength {\xintiiPow {2}{100}}=3
 ≠ \xintLen {\xintiiPow {2}{100}}=31
```

The maximal input size is limited by TEX main memory (it seems to be about half of the TEXLive main_memory setting from file texmf.cnf. Because TEX main memory is also where the used format is stored, as well as all additional defined macros, it will for example be higher if compiling with etex (PDFTEX in dvi mode) than with pdftex. Testing expansion inside an \edef with etex the author obtained with TEXLive 2025 a limit of 2492667 tokens with 1.4n.

10.5. \xintFirstItem

code

n ⋆ \xintFirstItem{⟨list⟩} returns the first item of its argument, one pair of braces removed. If the list has no items the output is empty.

It does no expansion. For this and the next similar ones, see xintsource.pdf for comments on limitations.

10.6. \xintLastItem

code

Added at 1.2i.

n ⋆ \xintLastItem{⟨list⟩} returns the last item of its argument, one pair of braces removed. If the list has no items the output is empty.

It does no expansion, which should be obtained via suitable \expandafter's. See also \xintNthElt{-1} from xinttools which obtains the same result (but with another code) after having however f-expanded its argument first.

10.7. \xintFirstOne

code

n ⋆ \xintFirstOne{⟨list⟩} returns the first item as a braced item. i.e. if it was braced the braces are kept, else the braces are added. It looks like using \xintFirstItem within braces, but the difference is when the input was empty. Then the output is empty.

It does no expansion, which should be obtained via suitable \expandafter's.

10.8. \xintLastOne

code

n ⋆ \xintLastOne{⟨list⟩} returns the last item as a braced item. i.e. if it was braced the braces are kept, else the braces are added. It looks like using \xintLastItem within braces, but the difference is when the input was empty. Then the output is empty.

It does no expansion, which should be obtained via suitable \expandafter's.

10.9. \xintReplicate, \xintreplicate

code code

num X n ⋆ \romannumeral\xintreplicate{x}{⟨stuff⟩} is simply copied over from LATEX3's \prg_replicate:nn with some minor changes.[67]

[67] I started with the code from Joseph WRIGHT available on an online site.

And \xintReplicate{x} integrates the \romannumeral prefix.

It does not do any expansion of its second argument but inserts it in the upcoming token stream precisely x times. Using it with a negative x raises no error and does nothing.[68]

10.10. \xintGobble, \xintgobble

code code

num x ★ \romannumeral\xintgobble{x} is a Gobbling macro written in the spirit of LaTeX3's \prg_replicate:n ↪ n (which I cloned as \xintreplicate.) It gobbles x tokens upstream, with x allowed to be as large as 531440. Don't use it with x<0.

And \xintGobble{x} integrates the \romannumeral.

\xintgobble looks as if it must be related to \xintTrim from xinttools, but the latter uses different code (using directly \xintgobble is not possible because one must make sure not to gobble more than the number of available items; and counting available items first is an overhead which \xintTrim avoids.) It is rather\xintKeep with a negative first argument which hands over to \xintgobble (because in that case it is needed to count anyhow beforehand the number of items, hence \xintgobble can then be used safely.)

I wrote an \xintcount in the same spirit as \xintreplicate and \xintgobble. But it needs to be counting hundreds of tokens to be worth its salt compared to \xintLength.

10.11. (WIP) \xintUniformDeviate

code

num x ★ \xintUniformDeviate{x} is based upon the engine primitive, which is \pdfuniformdeviate with PDFTeX, and \uniformdeviate with LuaTeX and XeTeX. A common name for that primitive is \xinttex ↪ uniformdeviate. But \xintUniformDeviate is, was will be explained designed to work around some features/limitations.

The argument is expanded in \numexpr and the macro itself needs two expansion steps. It produces like the engine primitive an integer (digit tokens) with minimal value 0 and maximal one x-1 if x is positive, or minimal value x+1 and maximal value 0 if x is negative. For the discussion next, x is supposed positive as this avoids having to insert absolute values in formulas.

The underlying engine primitive (thus, \xinttexuniformdeviate) accesses a random number generator (RNG) originally embedded into MetaPost and described in *The Art of Computer Programming, Vol. 2*. During discussions with Bruno Le Floch in May 2018, when he was adding to LaTeX3 interface for randomness, the author became aware of some limitations, some of them surprising, in the randomness of the numbers produced by this RNG. For example, the values produced by \pdfuniformde ↪ viate 201326592 and reduction modulo three are in the proportion 1:1:2, not 1:1:1.

Let's count how many 0's, 1's, and 2's we get from reducing modulo 3 the output of \xinttexunifo ↪ rmdeviate 201326592.

```
\xintsetrandomseed 87654321
\def\A{0}\def\B{0}\def\C{0}
\xintReplicate{504}{\ifcase\xinteval{(\xinttexuniformdeviate 201326592)/:3}
  \edef\A{\the\numexpr\A+1}\or
  \edef\B{\the\numexpr\B+1}\or
  \edef\C{\the\numexpr\C+1}\fi
}
We found \A{} 0's, \B{} 1's and \C{} 2's among 504 trials.
```

```
We found 124 0's, 147 1's and 233 2's among 504 trials.
```

In contrast, here is what happens if using \xintUniformDeviate:

```
\xintsetrandomseed 87654321
\def\A{0}\def\B{0}\def\C{0}
\xintReplicate{504}{\ifcase\xinteval{\xintUniformDeviate{201326592}/:3}
  \edef\A{\the\numexpr\A+1}\or
```

[68] This behaviour may change in future.

```
    \edef\B{\the\numexpr\B+1}\or
    \edef\C{\the\numexpr\C+1}\fi
  }
  We found \A{} 0's, \B{} 1's and \C{} 2's among 504 trials.
```
```
We found 161 0's, 174 1's and 169 2's among 504 trials.
```

TEX-hackers note: The RNG works with 28-bits integers. To output a supposedly uniform random integer in a given range 0..x-1, it first produces a supposedly uniform integer in the range $0..2^{28}-1$ (where $2^{28}=268435456$) then rescales (with rounding) to the target range. Note that with $x=2^{29}$ this means in particular that all produced ``random numbers'' in the 0..x-1 range will be even...

But even with $x=2^{28}$ there are some more serious defects of the RNG: two seeds sharing the same low k bits generate sequences of 28-bits integers which are identical one-to-one modulo 2^k! In particular after setting the seed, there are only 2 distinct sequences for the parity bits for the integers generated by \pdfuniformdeviate 268435456 (or rather with \xinttexuniformdeviate if with another engine than pdftex).

Let's define the non-uniformity of \pdfuniformdeviate x to be the maximum, taken over all y's from 0 to x-1, of $|x\times\mathrm{Prob}(\text{\pdfuniformdeviate x} = y) - 1|$. For a general x, the engine primitive guarantees only a $x/2^{28}$ relative non-uniformity for the outputs of \pdfuniformdeviate x.

\xintUniformDeviate improves this by a factor of 2^{28}=268435456: *the relative non-uniformity now is guaranteed to be bounded above by* $x/2^{56}$. With such a small non-uniformity, modulo phenomena as mentioned earlier are not observable in reasonable computing time.

The implementation of \xintUniformDeviate consumes exactly 5 calls to the engine primitive at each execution; the improved x/2^{56} non-uniformity could be obtained with only 2 calls, but paranoïa about the phenonemon of seeds with common bits has led me to accept the overhead of using the 7 high bits of 4 random 28-bits integers, rather than one single 28-bits integer, or two, or three.

Timings indicate that one \xintUniformDeviate has a time cost about 13 times the one for one call to the engine primitive (and not only 5, as the extra arithmetic expressions add overhead which is more costly than the primitive itself). Except if the code using the pseudo-random number is very short, this time penalty will prove in practice much less severe (and this is one important reason why we opted for obtaining 28bits via the 7 high bits of 4 successive pseudo random numbers from the engine primitive).

For example let's raise 100 times a random integer in the 0..99999999 range to the tenth power:[69]

```
  \xintsetrandomseed 12345678
  \xintresettimer
  \xintReplicate{100}{\edef\foo{\xintiiPow{\xintUniformDeviate{100000000}}{10}}}%
  \xinttheseconds s (if using \string\xintUniformDeviate)\newline
  \xintsetrandomseed 12345678
  \xintresettimer
  \xintReplicate{100}{\edef\foo{\xintiiPow{\xinttexuniformdeviate 100000000}{10}}}%
  \xinttheseconds s (if using \string\xinttexuniformdeviate)\par
```
```
0.01823s (if using \xintUniformDeviate)
0.01836s (if using \xinttexuniformdeviate)
```

The macros \xintRandomDigits or \xintiiRandRange, and their variants, as well as the supporting macros for random() generate random decimal digits eight by eight as if using \xintUniformDeviate{100000000}, but via a direct optimized call made possibly by the range being a power of 10.

[69] Timings done during dvi build on an Apple desktop with M4 Pro architecture.

11. Macros of the xintcore package

.1 Introduction 154
.2 \xintiNum 155
.3 \xintDouble 155
.4 \xintHalf 155
.5 \xintInc 155
.6 \xintDec 155
.7 \xintDSL 155
.8 \xintDSR 155
.9 \xintDSRr 155
.10 \xintFDg 155
.11 \xintLDg 156
.12 \xintiiSgn 156
.13 \xintiiOpp 156
.14 \xintiiAbs 156
.15 \xintiiAdd 156
.16 \xintiiCmp 156
.17 \xintiiSub 156
.18 \xintiiMul 156
.19 \xintiiSqr 156
.20 \xintiiPow 156
.21 \xintiiFac 157
.22 \xintiiDivision 157
.23 \xintiiQuo 157
.24 \xintiiRem 157
.25 \xintiiDivRound 157
.26 \xintiiDivTrunc 158
.27 \xintiiDivFloor 158
.28 \xintiiMod 158
.29 \xintNum 158

11.1. Introduction

Package xintcore is automatically loaded by xint.

xintcore provides for big integers the four basic arithmetic operations (addition, subtraction, multiplication, division), as well as powers and factorials.

In the descriptions of the macros {N} and {M} stand for (big) integers or macros *f*-expanding to such big integers in strict format as described in subsection 9.4.

All macros require strict integer format on input and produce strict integer format on output, except:

- \xintiNum which converts to strict integer format an input in *extended* integer format, i.e. admitting multiple leading plus or minus signs, then possibly leading zeroes, then digits,
- and \xintNum which is an alias for the former, which gets redefined by xintfrac to accept more generally also decimal numbers or fractions as input and which truncates them to integers.

The ii in the names of the macros such as \xintiiAdd serves to stress that they accept only strict integers as input (this is signaled by the margin annotation *f*), or macros *f*-expanding to such strict format (big) integers and that they produce strict integers as output.

Other macros, such as \xintDouble, lack the ii, but this is only a legacy of the history of the package and they have the same requirements for input and format of output as the ii-macros.

The letter x (with margin annotation $\overset{\text{num}}{x}$) stands for an argument which will be handled embedded in \numexpr..\relax. It will thus be completely expanded and must give an integer obeying the TEX bounds. See also subsection 9.6. This is the case for the argument of \xintiiFac or the exponent argument of \xintiiPow.

The ⋆'s in the margin are there to remind of the complete expandability, even *f*-expandability of the macros, as discussed in subsubsection 9.3.1.

Table 4 summarizes the maximal allowed sizes for the four operations. The first column is the tested macro (it is expanded in an \edef; if deeper nested, the maximal admissible input sizes may actually prove lower than stated). The second column gives the maximal N such that the macro does not raise an error on both inputs having N digits (for division, the used test divisors had N/4, N/2 and 3N/4 digits).

These maximal N's depend on the values of TEX parameters such as input stack size and expansion depth. The last column gives the TEX parameter cited in the error message when trying with N+1 digits.

The table was last updated in July 2025, using 1.4m 2022/06/10 and TeXLive 2025 default settings which are: input stack size at 10000, expansion depth at 10000, parameter stack size at 20000.

	Max length of inputs	Limiting factor
\xintiiAdd	26648	expansion depth=10000
\xintiiSub	26632	expansion depth=10000
\xintiiMul	13320	expansion depth=10000
\xintiiDivision	26609	expansion depth=10000

Table 4: Maximal sizes of inputs (using TeXLive 2025) for core arithmetic

code
11.2. \xintiNum

f ⋆ \xintiNum{N} removes chains of plus or minus signs, followed by zeroes.

```
  \xintiNum{+---++----+--00000000367941789479}
-367941789479
```

code
11.3. \xintDouble

f ⋆ \xintDouble{N} computes 2N.

code
11.4. \xintHalf

f ⋆ \xintHalf{N} computes N/2 truncated towards zero.

code
11.5. \xintInc

f ⋆ \xintInc{N} evaluates N+1.

code
11.6. \xintDec

f ⋆ \xintDec{N} evaluates N-1.

code
11.7. \xintDSL

f ⋆ \xintDSL{N} is decimal shift left, *i.e.* multiplication by ten.

code
11.8. \xintDSR

f ⋆ \xintDSR{N} is truncated decimal shift right, *i.e.* it is the truncation of N/10 towards zero.

code
11.9. \xintDSRr

f ⋆ \xintDSRr{N} is rounded decimal shift right, *i.e.* it is the rounding of N/10 away from zero. It is needed in xintcore for use by \xintiiDivRound.

code
11.10. \xintFDg

f ⋆ \xintFDg{N} outputs the first digit (most significant) of the number.

code
11.11. \xintLDg

f ⋆ \xintLDg{N} outputs the least significant digit. When the number is positive, this is the same as the remainder in the Euclidean division by ten.

code
11.12. \xintiiSgn

f ⋆ \xintiiSgn{N} returns 1 if the number is positive, 0 if it is zero and -1 if it is negative.

code
11.13. \xintiiOpp

f ⋆ \xintiiOpp{N} outputs the opposite -N of the number N.

Important note: an input such as -\foo is not legal, generally speaking, as argument to the macros of the xint bundle (except, naturally in \xintexpr-essions). The reason is that the minus sign stops the *f*-expansion done during parsing of the inputs. One must use the syntax \xintiiOpp{\foo} if one wants to pass -\foo as argument to other macros.

code
11.14. \xintiiAbs

f ⋆ \xintiiAbs{N} outputs the absolute value of the number.

code
11.15. \xintiiAdd

ff ⋆ \xintiiAdd{N}{M} computes the sum of the two (big) integers.

code
11.16. \xintiiCmp

ff ⋆ \xintiiCmp{N}{M} produces 1 if N>M, 0 if N=M, and -1 if N<M.

At 1.2l this macro was moved from package xint to xintcore.

code
11.17. \xintiiSub

ff ⋆ \xintiiSub{N}{M} computes the difference N-M.

code
11.18. \xintiiMul

ff ⋆ \xintiiMul{N}{M} computes the product of two (big) integers.

code
11.19. \xintiiSqr

f ⋆ \xintiiSqr{N} produces the square.

code
11.20. \xintiiPow

f^{num} ⋆ \xintiiPow{N}{x} computes N^x. For x=0, this is 1. For N=0 and x<0, or if |N|>1 and x<0, an error is raised. There will also be an error if x exceeds the maximal ε-TEX number 2147483647, but the real limit for exponents comes from either the computation time or the settings of some TEX memory parameters.

Generally speaking the computation will end in an error if the output goes beyond what addition could accept on input, i.e. (with TEXLive 2025 default settings) has more than about 26600 digits.

For example the maximal power of 2 which \xintiiPow is able to compute (with TeXLive 2025 default memory parameters) is 2^{88470} which has 26633 digits. I.e. \edef\z{\xintiiPow{2}{88470}} succeeds (if you are patient enough to wait) but \edef\z{\xintiiPow{2}{88471}} fails.

11.21. \xintiiFac

code

num x ★ \xintiiFac{x} computes the factorial.

The (theoretically) allowable range is $0 \leqslant x \leqslant 10000$.

However the maximal possible computation depends on the values of some memory parameters of the ε-TeX executable: with xintcore at 1.4m and using TeXLive 2025, the maximal (within an \edef) computable one is 7712! which has 26631 decimal digits. With the input stack size set at 10000, the limiting factor here is the expansion depth at 10000.

The factorial() function, or equivalently !() as post-fix operator is available in the three parsers:

```
  \xinttheiiexpr factorial(30)\relax\par
  \printnumber{\xinteval{200!}}\par
265252859812191058636308480000000
  788657867364790503552363213932185062295135977687173263294742533244359449963403342920304284
↪ 011984623904177212138919638830257642790242637105061926624952829931113462857270763317237396988
↪ 894392244562145166424025403329186413122742829485327752424240757390324032125740557956866022608
↪ 319041703240623517008587961789222227896237038973747200000000000000000000000000000000000000000
↪ 000000000
```

Within \xintfloateval, the macro \xintFloatFac from package xintfrac is used.

```
  \xintfloateval{200!}\par
7.886578673647905e374
  \xintfloateval{2000!}\par
3.316275092450633e5735
```

See its documentation for more.

11.22. \xintiiDivision

code

ff ★ \xintiiDivision{M}{N} produces {quotient}{remainder}, in the sense of (mathematical) Euclidean division: M = QN + R, $0 \leq R < |N|$. So the remainder is always non-negative and the formula M = QN + R always holds independently of the signs of N or M. Division by zero is an error (even if M vanishes) and returns {0}{0}.

11.23. \xintiiQuo

ff ★ \xintiiQuo{M}{N} computes the quotient from the Euclidean division.

11.24. \xintiiRem

ff ★ \xintiiRem{M}{N} computes the remainder from the Euclidean division.

11.25. \xintiiDivRound

ff ★ \xintiiDivRound{M}{N} returns the rounded value of the algebraic quotient M/N of two big integers. The rounding is ``away from zero.''

```
  \xintiiDivRound {100}{3}, \xintiiDivRound {101}{3}
33, 34
```

11.26. \xintiiDivTrunc

f f ⋆ \xintiiDivTrunc{M}{N} computes trunc(M/N). For positive arguments M, N > 0 it is the same as the Euclidean quotient \xintiiQuo.

```
  \xintiiQuo{1000}{57} (Euclidean), \xintiiDivTrunc{1000}{57} (truncated),
  \xintiiDivRound{1000}{57} (rounded)\newline
  \xintiiQuo{-1000}{57}, \xintiiDivTrunc{-1000}{57} (t), \xintiiDivRound{-1000}{57} (r)
  \newline
  \xintiiQuo{1000}{-57}, \xintiiDivTrunc{1000}{-57} (t), \xintiiDivRound{1000}{-57} (r)
  \newline
  \xintiiQuo{-1000}{-57}, \xintiiDivTrunc{-1000}{-57} (t), \xintiiDivRound{-1000}{-57} (r)
  \par
```

```
17 (Euclidean), 17 (truncated), 18 (rounded)
-18, -17 (t), -18 (r)
-17, -17 (t), -18 (r)
18, 17 (t), 18 (r)
```

11.27. \xintiiDivFloor

f f ⋆ \xintiiDivFloor{M}{N} computes floor(M/N). For positive divisor N > 0 and arbitrary dividend M it is the same as the Euclidean quotient \xintiiQuo.

```
  \xintiiQuo{1000}{57} (Euclidean), \xintiiDivFloor{1000}{57} (floored)\newline
  \xintiiQuo{-1000}{57}, \xintiiDivFloor{-1000}{57}\newline
  \xintiiQuo{1000}{-57}, \xintiiDivFloor{1000}{-57}\newline
  \xintiiQuo{-1000}{-57}, \xintiiDivFloor{-1000}{-57}\par
```

```
17 (Euclidean), 17 (floored)
-18, -18
-17, -18
18, 17
```

11.28. \xintiiMod

f f ⋆ \xintiiMod{M}{N} computes M - N * floor(M/N). For positive divisor N > 0 and arbitrary dividend M it is the same as the Euclidean remainder \xintiiRem.

Formerly, this macro computed M - N * trunc(M/N). The former meaning is retained as \xintiiModTrunc.

```
  \xintiiRem {1000}{57} (Euclidean), \xintiiMod {1000}{57} (floored),
  \xintiiModTrunc {1000}{57} (truncated)\newline
  \xintiiRem {-1000}{57}, \xintiiMod {-1000}{57}, \xintiiModTrunc {-1000}{57}\newline
  \xintiiRem {1000}{-57}, \xintiiMod {1000}{-57}, \xintiiModTrunc {1000}{-57}\newline
  \xintiiRem {-1000}{-57}, \xintiiMod {-1000}{-57}, \xintiiModTrunc {-1000}{-57}\par
```

```
31 (Euclidean), 31 (floored), 31 (truncated)
26, 26, -31
31, -26, 31
26, -31, -31
```

11.29. \xintNum

code

f ⋆ \xintNum is originally an alias for \xintiNum. But with xintfrac loaded its meaning is modified to accept more general inputs. It then becomes an alias to \xintTTrunc which truncates the general input to an integer in strict format.

12. Macros of the xint package

.1 Introduction 159
.2 \xintiLen 160
.3 \xintReverseDigits 160
.4 \xintDecSplit 160
.5 \xintDecSplitL, \xintDecSplitR 161
.6 \xintiiE 161
.7 \xintDSH 161
.8 \xintDSHr, \xintDSx 161
.9 \xintiiEq 161
.10 \xintiiNotEq 161
.11 \xintiiGeq 162
.12 \xintiiGt 162
.13 \xintiiLt 162
.14 \xintiiGtorEq 162
.15 \xintiiLtorEq 162
.16 \xintiiIsZero 162
.17 \xintiiIsNotZero 162
.18 \xintiiIsOne 162
.19 \xintiiOdd 162
.20 \xintiiEven 162
.21 \xintiiMON 162
.22 \xintiiMMON 163
.23 \xintiiifSgn 163
.24 \xintiiifZero 163
.25 \xintiiifNotZero 163
.26 \xintiiifOne 163
.27 \xintiiifCmp 163
.28 \xintiiifEq 163
.29 \xintiiifGt 163
.30 \xintiiifLt 163
.31 \xintiiifOdd 164
.32 \xintiiSum 164
.33 \xintiiPrd 164
.34 \xintiiSquareRoot 164
.35 \xintiiSqrt, \xintiiSqrtR 165
.36 \xintiiBinomial 165
.37 \xintiiPFactorial 166
.38 \xintiiMax 166
.39 \xintiiMin 166
.40 \xintiiMaxof 166
.41 \xintiiMinof 166
.42 \xintifTrueAelseB 167
.43 \xintifFalseAelseB 167
.44 \xintNOT 167
.45 \xintAND 167
.46 \xintOR 167
.47 \xintXOR 167
.48 \xintANDof 167
.49 \xintORof 167
.50 \xintXORof 167
.51 \xintiiGCD 168
.52 \xintiiLCM 168
.53 \xintiiGCDof 168
.54 \xintiiLCMof 168
.55 \xintLen 168
.56 (WIP) \xintRandomDigits 168
.57 (WIP) \xintXRandomDigits 169
.58 (WIP) \xintiiRandRange 169
.59 (WIP) \xintiiRandRangeAtoB 170

12.1. Introduction

This package loads automatically xintcore (and xintkernel) hence all macros described in section 11 are still available.

This is 2.0a2 of 2026/11/26.

Version 1.0 was released 2013/03/28. Since 1.1 2014/10/28 the core arithmetic macros have been moved to a separate package xintcore, which is automatically loaded by xint. Only the \xintiiSum, \xintiiPrd, \xintiiSquareRoot, \xintiiSqrt, \xintiiSqrtR, \xintiiPFactorial, \xintiiBinomial genuinely add to the arithmetic macros from xintcore. (\xintiiFac which computes factorials is already in xintcore.)

With the exception of \xintLen, of the "Boolean logic macros" (see next paragraphs) all macros require inputs being integers in strict format, see subsection 9.4.[70] The ii in the macro names is here as a reminder of that fact. The output is an integer in strict format, or a pair of two braced such integers for \xintiiSquareRoot, with the exception of \xintiiE which may produce strings of zero's if its first argument is zero.

Macros \xintDecSplit and \xintReverseDigits are non-arithmetic and have their own specific rules.

[70] of course for conditionals such as \xintiiifCmp this constraint applies only to the first two arguments.

For all macros described here for which it makes sense, package xintfrac defines a similar one without ii in its name. This will handle more general inputs: decimal, scientific numbers, fractions. The ii macros provided here by xint can be nested inside macros of xintfrac but the opposite does not apply, because the output format of the xintfrac macros, even for representing integers, is not understood by the ii macros. The "Boolean macros" \xintAND etc... are exceptions though, they work fine if served as inputs some xintfrac output, despite doing only *f*-expansion. Prior to 1.2o, these macros did apply the \xintNum or the more general xintfrac general parsing, but this overhead was deemed superfluous as it serves only to handle hand-written input and is not needed if the input is obtained as a nested chain of xintfrac macros for example.

Prior to release 1.2o, xint defined additional macros which applied \xintNum to their input arguments. All these macros were deprecated at 1.2o and have been removed at 1.3.

At 1.3d macros \xintiiGCD and \xintiiLCM from package xintgcd are also available from loading xint only. They are support macros for the (multi-arguments) functions gcd() and lcm() in \xintiiexpr.

See subsubsection 9.3.1 for the significance of the $^{\text{Num}}f$, f, $^{\text{num}}x$ and ⋆ margin annotations.

12.2. \xintiLen

code

$^{\text{Num}}f$ ⋆ \xintiLen{N} returns the length of the number, after its parsing via \xintiNum. The count does not include the sign.

```
\xintiLen{-12345678901234567890123456789}
```
```
29
```

Prior to 1.2o, the package defined only \xintLen, which is extended by xintfrac to fractions or decimal numbers, hence acquires a bit more overhead then.

12.3. \xintReverseDigits

code

3.60004pt, 8.39996pt, 12.0pt

f ⋆ \xintReverseDigits{N} will reverse the order of the digits of the number. \xintRev is the former denomination and is kept as an alias. Leading zeroes resulting from the operation are not removed. Contrarily to \xintReverseOrder this macro *f*-expands its argument; it is only usable with digit tokens. It does *not* apply \xintNum to its argument (so this must be done explicitely if the argument is an integer produced from some xintfrac macros). It does accept a leading minus sign which will be left upfront in the output.

```
\oodef\x{\xintReverseDigits
  {98765432109876543210987654321098765432109876543210}}\meaning\x\par
\noindent\oodef\x{\xintReverseDigits {\xintReverseDigits
  {98765432109876543210987654321098765432109876543210}}}\meaning\x\par
```
```
macro:->01234567890123456789012345678901234567890123456789
macro:->98765432109876543210987654321098765432109876543210
```

12.4. \xintDecSplit

code

$^{\text{num}}x$ *f* ⋆ \xintDecSplit{x}{N} cuts the N (a list of digits) into two pieces L and R: it outputs {L}{R} where the original N is the concatenation LR. These two pieces are decided according to x:

- for x>0, R coincides with the x least significant digits. If x equals or exceeds the length of N the first piece L will thus be *empty*,
- for x=0, R is empty, and L is all of N,
- for x<0, the first piece L consists of the |x| most significant digits and the second piece R gets the remaining ones. If x equals or exceeds the length of N the second piece R will thus be *empty*.

This macro provides public interface to some functionality which is primarily of internal interest. It operates only (after *f*-expansion) on ``strings'' of digits tokens: leading zeroes are allowed but a leading sign (even a minus sign) will provoke an error.

Breaking change with 1.2i: formerly N<0 was replaced by its absolute value. Now, a sign (positive or negative) will create an error.

12.5. \xintDecSplitL, \xintDecSplitR

code code

num x f ★ \xintDecSplitL{x}{N} returns the first piece (unbraced) from the \xintDecSplit output.

num x f ★ \xintDecSplitR{x}{N} returns the second piece (unbraced) from the \xintDecSplit output.

12.6. \xintiiE

code

f num x ★ \xintiiE{N}{x} serves to extend N with x zeroes. The parameter x must be non-negative. The same output would be obtained via \xintDSH{-x}{N}, except for N=0, as \xintDSH{-x}{N} multiplies N by 10^x hence produces 0 if N=0 whereas \xintiiE{0}{x} produces x+1 zeros.

```
  \xintiiE {0}{91}\par
```

```
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
```

12.7. \xintDSH

code

num x f ★ \xintDSH{x}{N} is parametrized decimal shift. When x is negative, it is like iterating \xintDSL |x| times (*i.e.* multiplication by 10^{-x}). When x positive, it is like iterating \xintDSR x times (and is more efficient), and for a non-negative N this is thus the same as the quotient from the Euclidean division by 10^x.

12.8. \xintDSHr, \xintDSx

code code

num x f ★ \xintDSHr{x}{N} expects x to be zero or positive and it returns then a value R which is correlated to the value Q returned by \xintDSH{x}{N} in the following manner:

- if N is positive or zero, Q and R are the quotient and remainder in the Euclidean division by 10^x (obtained in a more efficient manner than using \xintiiDivision),
- if N is negative let Q1 and R1 be the quotient and remainder in the Euclidean division by 10^x of the absolute value of N. If Q1 does not vanish, then Q=-Q1 and R=R1. If Q1 vanishes, then Q=0 and R=-R1.
- for x=0, Q=N and R=0.

So one has N = 10^x Q + R if Q turns out to be zero or positive, and N = 10^x Q - R if Q turns out to be negative, which is exactly the case when N is at most -10^x.

num x f ★ \xintDSx{x}{N} for x negative is exactly as \xintDSH{x}{N}, *i.e.* multiplication by 10^{-x}. For x zero or positive it returns the two numbers {Q}{R} described above, each one within braces. So Q is \xintDSH{x}{N}, and R is \xintDSHr{x}{N}, but computed simultaneously.

12.9. \xintiiEq

code

f f ★ \xintiiEq{N}{M} returns 1 if N=M, 0 otherwise.

12.10. \xintiiNotEq

code

f f ★ \xintiiNotEq{N}{M} returns 0 if N=M, 1 otherwise.

code
12.11. \xintiiGeq

f f ⋆ \xintiiGeq{N}{M} returns 1 if the *absolute value* of the first number is at least equal to the absolute value of the second number. If |N|<|M| it returns 0.
Important: the macro compares *absolute values*.

code
12.12. \xintiiGt

f f ⋆ \xintiiGt{N}{M} returns 1 if N>M, 0 otherwise.

code
12.13. \xintiiLt

f f ⋆ \xintiiLt{N}{M} returns 1 if N<M, 0 otherwise.

code
12.14. \xintiiGtorEq

f f ⋆ \xintiiGtorEq{N}{M} returns 1 if N⩾M, 0 otherwise. Extended by xintfrac to fractions.

code
12.15. \xintiiLtorEq

f f ⋆ \xintiiLtorEq{N}{M} returns 1 if N⩽M, 0 otherwise.

code
12.16. \xintiiIsZero

f ⋆ \xintiiIsZero{N} returns 1 if N=0, 0 otherwise.

code
12.17. \xintiiIsNotZero

f ⋆ \xintiiIsNotZero{N} returns 1 if N!=0, 0 otherwise.

code
12.18. \xintiiIsOne

f ⋆ \xintiiIsOne{N} returns 1 if N=1, 0 otherwise.

code
12.19. \xintiiOdd

f ⋆ \xintiiOdd{N} is 1 if the number is odd and 0 otherwise.

code
12.20. \xintiiEven

f ⋆ \xintiiEven{N} is 1 if the number is even and 0 otherwise.

code
12.21. \xintiiMON

f ⋆ \xintiiMON{N} computes (-1)^N.

```
  \xintiiMON {-280914019374101929}
-1
```

12.22. \xintiiMMON

code

f ⋆ \xintiiMMON{N} computes (-1)^{N-1}.

```
  \xintiiMMON {280914019374101929}
1
```

12.23. \xintiiifSgn

code

\xintiiifSgn

fnnn ⋆ {⟨N⟩}{⟨A⟩}{⟨B⟩}{⟨C⟩} executes either the ⟨A⟩, ⟨B⟩ or ⟨C⟩ code, depending on its first argument being respectively negative, zero, or positive.

12.24. \xintiiifZero

code

fnn ⋆ \xintiiifZero{⟨N⟩}{⟨IsZero⟩}{⟨IsNotZero⟩} expandably checks if the first mandatory argument N (a number, possibly a fraction if xintfrac is loaded, or a macro expanding to one such) is zero or not. It then either executes the first or the second branch.

Beware that both branches must be present.

12.25. \xintiiifNotZero

code

fnn ⋆ \xintiiifNotZero{⟨N⟩}{⟨IsNotZero⟩}{⟨IsZero⟩} expandably checks if the first mandatory argument N is not zero or is zero. It then either executes the first or the second branch.

Beware that both branches must be present.

12.26. \xintiiifOne

code

fnn ⋆ \xintiiifOne{⟨N⟩}{⟨IsOne⟩}{⟨IsNotOne⟩} expandably checks if the first mandatory argument N is one or not one. It then either executes the first or the second branch. Beware that both branches must be present.

12.27. \xintiiifCmp

code

ffnnn ⋆ \xintiiifCmp{⟨A⟩}{⟨B⟩}{⟨A<B⟩}{⟨A=B⟩}{⟨A>B⟩} compares its first two arguments and chooses accordingly the correct branch.

12.28. \xintiiifEq

code

ffnn ⋆ \xintiiifEq{⟨A⟩}{⟨B⟩}{⟨A=B⟩}{⟨not(A=B)⟩} checks equality of its two first arguments and executes the corresponding branch.

12.29. \xintiiifGt

code

ffnn ⋆ \xintiiifGt{⟨A⟩}{⟨B⟩}{⟨A>B⟩}{⟨not(A>B)⟩} checks if A > B and executes the corresponding branch.

12.30. \xintiiifLt

code

ffnn ⋆ \xintiiifLt{⟨A⟩}{⟨B⟩}{⟨A<B⟩}{⟨not(A<B)⟩} checks if A < B and executes the corresponding branch.

12.31. \xintiiifOdd

code

fnn ★ \xintiiifOdd{⟨A⟩}{⟨A odd⟩}{⟨A even⟩} checks if A is and odd integer and executes the corresponding branch.

12.32. \xintiiSum

code

*f ★ \xintiiSum{⟨braced things⟩} after expanding its argument expects to find a sequence of tokens (or braced material). Each is f-expanded, and the sum of all these numbers is returned.

```
  \xintiiSum{{123}{-98763450}{\xintiiFac{7}}{\xintiiMul{3347}{591}}}\newline
  \xintiiSum{1234567890}\newline
  \xintiiSum{1234}\newline
  \xintiiSum{}
-96780210
45
10
0
```

A sum with only one term returns that number: \xintiiSum {{-1234}}=-1234. Attention that \xintiiSum {-1234} is not legal input and would make the TEX run fail.

12.33. \xintiiPrd

code

*f ★ \xintiiPrd{⟨braced things⟩} after expanding its argument expects to find a sequence of (of braced items or unbraced single tokens). Each is expanded (with the usual meaning), and the product of all these numbers is returned.

```
  \xintiiPrd{{-9876}{\xintiiFac{7}}{\xintiiMul{3347}{591}}}\newline
  \xintiiPrd{123456789123456789}\newline
  \xintiiPrd {1234}\newline
  \xintiiPrd{}
-98458861798080
131681894400
24
1
```

Attention that \xintiiPrd {-1234} is not legal input and would make the TEX compilation fail.

```
  $2^{200}3^{100}7^{100}=\printnumber
         {\xintiiPrd {{\xintiiPow {2}{200}}{\xintiiPow {3}{100}}{\xintiiPow {7}{100}}}}$
```

$2^{200}3^{100}7^{100}$ = 267872793166157757576627951700754840232474026637401534897445961481542641296549900004440072407657271300001653120764065456211801435719940159033435392440282124389668222489278629880843827161331376

With xintexpr, the syntax is the natural one:

```
  $2^{200}3^{100}7^{100}=\printnumber{\xinttheiiexpr 2^200 * 3^100 * 7^100\relax}$
```

$2^{200}3^{100}7^{100}$ = 267872793166157757576627951700754840232474026637401534897445961481542641296549900004440072407657271300001653120764065456211801435719940159033435392440282124389668222489278629880843827161331376

12.34. \xintiiSquareRoot

code

f ★ \xintiiSquareRoot{N} returns two braced integers {M}{d} which satisfy d>0 and M^2-d=N with M the smallest (hence if N=k^2 is a perfect square then M=k+1, d=2k+1).

```
  \xintAssign\xintiiSquareRoot {17000000000000000000000000}\to\A\B
  \xintiiSub{\xintiiSqr\A}\B=\A\string^2-\B
```

17000000000000000000000000=4123105625618^2-2799177881924

A rational approximation to $\sqrt{N}$ is M - $\frac{d}{2M}$ which is a majorant and the error is at most 1/2M (if N is a perfect square k^2 this gives k+1/(2k+2), not k.)

Package xintfrac has \xintFloatSqrt for square roots of floating point numbers.

12.35. \xintiiSqrt, \xintiiSqrtR

code code

f ⋆ \xintiiSqrt{N} computes the largest integer whose square is at most equal to N. \xintiiSqrtR pro-
f ⋆ duces the rounded, not truncated, square root.

```
\begin{itemize}[nosep]
\item \xintiiSqrt  {30000000000000000000000000000000000}
\item \xintiiSqrtR {30000000000000000000000000000000000}
\item \xintiiSqrt  {\xintiiE {3}{100}}
\end{itemize}
```

- 1732050807568877293
- 1732050807568877294
- 173205080756887729352744634150587236694280525381038

12.36. \xintiiBinomial

code

num num
x x ⋆ \xintiiBinomial{x}{y} computes binomial coefficients.

If x<0 an out-of-range error is raised. Else, if y<0 or if x<y the macro evaluates to 0.

The allowable range is 0 ⩽ x ⩽ 99999999. But this theoretical range includes binomial coefficients with more than the roughly 19950 digits that the arithmetics of xint can handle. In such cases, the computation will end up in a low-level TEX error after a long time.

It turns out that $\binom{65000}{32500}$ has 19565 digits and $\binom{64000}{32000}$ has 19264 digits. The latter can be evaluated (this takes a long long time) but presumably not the former (I didn't try). Reasonable feasible evaluations are with binomial coefficients not exceeding about one thousand digits.

The binomial function is available in the xintexpr parsers.

```
\xinttheiiexpr seq(binomial(100,i), i=47..53)\relax
```

844134872830640395015079376 00, 932065588750498769495816811 00, 989130828878080326811887228 00, 100891344545564193334812497256, 989130828878080326811887228 00, 932065588750498769495816811 00, 844134872830640395015079376 00

See \xintFloatBinomial from package xintfrac for the float variant, used in \xintfloatexpr.

In order to evaluate binomial coefficients $\binom{x}{y}$ with x > 99999999, or even x ⩾ 2^{31}, but y is not too large, one may use an ad hoc function definition such as:

```
\xintdeffunc mybigbinomial(x,y):=`*`(x-y+1..[1]..x)//y!;%
%                              without [1], x would have been limited to < 2^31
\printnumber{\xinttheexpr mybigbinomial(98765432109876543210,10)\relax}
```

24338098741940755592729533173058146177070669479669793038510211146784065843698581878582323710
↪ 27360575372715482389633359878460739973726786576925067784100587971261422326652270975592667517
↪ 4871960261

To get this functionality in macro form, one can do:

```
\xintNewIIExpr\MyBigBinomial [2]{`*`(#1-#2+1..[1]..#1)//#2!}
\printnumber{\MyBigBinomial {98765432109876543210}{10}}
```

24338098741940755592729533173058146177070669479669793038510211146784065843698581878582323710
↪ 27360575372715482389633359878460739973726786576925067784100587971261422326652270975592667517
↪ 4871960261

As we used \xintNewIIExpr, this macro will only accept strict integers. Had we used \xintNewExpr the \MyBigBinomial would have accepted general fractions or decimal numbers, and computed the product at the numerator without truncating them to integers; but the factorial at the denominator would truncate its argument.

code
12.37. \xintiiPFactorial

num num x x ★ \xintiiPFactorial{a}{b} computes the partial factorial (a+1)(a+2)...b. For a=b the product is considered empty hence returns 1.

The allowed range is -100000000 ⩽ a, b ⩽ 99999999. The rule is to interpret the formula as the product of the j's such that a < j ⩽ b, hence in particular if a ⩾ b the product is empty and the macro evaluates to 1.

Only for 0 ⩽ a ⩽ b is the behaviour to be considered stable. For a > b or negative arguments, the definitive rules have not yet been fixed.

```
\xintiiPFactorial {100}{130}
```
```
69293021885203871012298422845822803287591970060789350400000000
```

This theoretical range allows computations whose result values would have more than the roughly 19950 digits that the arithmetics of xint can handle. In such cases, the computation will end up in a low-level TEX error after a long time.

The pfactorial function is available in the xintexpr parsers.

```
\xinttheiiexpr pfactorial(100,130)\relax
```
```
69293021885203871012298422845822803287591970060789350400000000
```

See \xintFloatPFactorial from package xintfrac for the float variant, used in \xintfloatexpr.

In case values are needed with b > 99999999, or even b ⩾ 2^{31}, but b - a is not too large, one may use an ad hoc function definition such as:

```
\xintdeffunc mybigpfac(a,b):=`*`(a+1..[1]..b);%
%                            without [1], b would have been limited to < 2^31
\printnumber{\xinttheexpr mybigpfac(98765432100,98765432120)\relax}
```
```
78000855017567528067298107313023778438653002029049647467208196028116499434050587656870489322
↪ 99630604482236853566403912561449912587404607844104078121472675461815442734098676283450069933
↪ 32294860057301699703400956657664000 0
```

code
12.38. \xintiiMax

f f ★ \xintiiMax{N}{M} returns the largest of the two in the sense of the order structure on the relative integers (*i.e.* the right-most number if they are put on a line with positive numbers on the right): \xintiiMax {-5}{-6}=-5.

code
12.39. \xintiiMin

f f ★ \xintiiMin{N}{M} returns the smallest of the two in the sense of the order structure on the relative integers (*i.e.* the left-most number if they are put on a line with positive numbers on the right): \xintiiMin {-5}{-6}=-6.

code
12.40. \xintiiMaxof

f ⟶ * f ★ \xintiiMaxof{{a}{b}{c}...} returns the maximum. The list argument may be a macro, it is *f*-expanded first.

code
12.41. \xintiiMinof

f ⟶ * f ★ \xintiiMinof{{a}{b}{c}...} returns the minimum. The list argument may be a macro, it is *f*-expanded first.

code
12.42. \xintifTrueAelseB

fnn ⋆ \xintifTrueAelseB{⟨*f*⟩}{⟨*true branch*⟩}{⟨*false branch*⟩} is a synonym for \xintiiifNotZero. \xintiiifnotzero is lowercase companion macro.

Note 1: as it does only *f*-expansion on its argument it fails with inputs such as --0. But with xintfrac loaded, it does work fine if nested with other xintfrac macros, because the output format of such macros is fine as input to \xintiiifNotZero. This remark applies to all other "Boolean logic" macros next.

Note 2: prior to 1.2o this macro was using \xintifNotZero which applies \xintNum to its argument (or gets redefined by xintfrac to handle general decimal numbers or fractions). Hence it would have worked with input such as --0. But it was decided at 1.2o that the overhead was not worth it. The same remark applies to the other "Boolean logic" type macros next.

code
12.43. \xintifFalseAelseB

fnn ⋆ \xintifFalseAelseB{⟨*f*⟩}{⟨*false branch*⟩}{⟨*true branch*⟩} is a synonym for \xintiiifZero. \xintiiifzero is lowercase companion macro.

code
12.44. \xintNOT

f ⋆ \xintNOT is a synonym for \xintiiIsZero.
\xintiiiszero serves as lowercase companion macro.

code
12.45. \xintAND

ff ⋆ \xintAND{f}{g} returns 1 if f!=0 and g!=0 and 0 otherwise.

code
12.46. \xintOR

ff ⋆ \xintOR{f}{g} returns 1 if f!=0 or g!=0 and 0 otherwise.

code
12.47. \xintXOR

ff ⋆ \xintXOR{f}{g} returns 1 if exactly one of f or g is true (i.e. non-zero), else 0.

code
12.48. \xintANDof

f→ ∗*f* ⋆ \xintANDof{{a}{b}{c}...} returns 1 if all are true (i.e. non zero) and 0 otherwise. The list argument may be a macro, it (or rather its first token) is *f*-expanded first to deliver its items.

code
12.49. \xintORof

f→ ∗*f* ⋆ \xintORof{{a}{b}{c}...} returns 1 if at least one is true (i.e. does not vanish), else it produces 0. The list argument may be a macro, it is *f*-expanded first.

code
12.50. \xintXORof

f→ ∗*f* ⋆ \xintXORof{{a}{b}{c}...} returns 1 if an odd number of them are true (i.e. do not vanish), else it produces 0. The list argument may be a macro, it is *f*-expanded first.

12.51. \xintiiGCD

code

f f ⋆ \xintiiGCD{N}{M} computes the greatest common divisor. It is positive, except when both N and M vanish, in which case the macro returns zero.

```
\xintiiGCD{10000}{1113}=1
\xintiiGCD{123456789012345}{9876543210321}=3
```

At 1.3d, this macro (which is used by the gcd() function in \xintiiexpr) was copied over to xint, thus removing a partial dependency of xintexpr on xintgcd.

At 1.4 xintgcd requires xint and the latter is thus the one providing the macro.

12.52. \xintiiLCM

code

f f ⋆ \xintiiLCM{N}{M} computes the least common multiple. It is positive, except if one of N or M vanish, in which case the macro returns zero.

```
\xintiiLCM{10000}{1113}=11130000
\xintiiLCM{123456789012345}{9876543210321}=406442103762636081733470915
```

At 1.3d, this macro (which is used by the lcm() function in \xintiiexpr) was copied over to xint, thus removing a partial dependency of xintexpr on xintgcd.

At 1.4 xintgcd requires xint and the latter is thus the one providing the macro.

12.53. \xintiiGCDof

code

f ⟶ ⋆*f* ⋆ \xintiiGCDof{{a}{b}{c}...} computes the greatest common divisor of the integers a, b, ... It is a support macro for the gcd() function of the \xintiiexpr parser.

It replaces the \xintGCDof which was formerly provided by xintgcd and is now available via xintfrac in a version handling also fractions.

12.54. \xintiiLCMof

code

f ⟶ ⋆*f* ⋆ \xintiiLCMof{{a}{b}{c}...} computes the least common multiple of the integers a, b, ... It is a support macro for the lcm() function of the \xintiiexpr parser.

It replaces the \xintLCMof which was formerly provided by xintgcd and is now available via xintfrac in a version handling also fractions.

12.55. \xintLen

code

Num *f* ⋆ \xintLen is originally an alias for \xintiLen. But with xintfrac loaded its meaning is modified to accept more general inputs.

12.56. (WIP) \xintRandomDigits

code

> All randomness related macros are Work-In-Progress: implementation and user interface may change. They work only if the TEX engine provides the \uniformdeviate or \pdfuniformdeviate primitive. See \xintUniformDeviate for additional information.

num *x* ⋆ \xintRandomDigits{N} expands in two steps to N random decimal digits. The argument must be non-negative and is limited by TEX memory parameters. On TEXLive 2018 with input save stack size at 5000 the maximal allowed N is at most 19984 (tested within a \write to an auxiliary file, the macro context may cause a reduced maximum).

```
\xintsetrandomseed 271828182
\xintRandomDigits{92}
```
```
60033782389146151207277993539344280578090871919638745398735577686436165769394958639376355806
```

TEX-hackers note: the digits are produced eight by eight by the same method which would result from \xintUniformDeviate{100000000} but with less overhead.

12.57. (WIP) \xintXRandomDigits

code

num X ☆ \xintXRandomDigits{N} expands under exhaustive expansion (\edef, \write, \csname ...) to N random decimal digits. The argument must be non-negative. For example:

```
\newwrite\out
\immediate\openout\out=\jobname-out.txt
\immediate\write\out{\xintXRandomDigits{4500000}}
\immediate\closeout\out
```

creates a 4500001 bytes file (it ends with a line feed character). Trying with 5000000 raises this error:

```
Runaway text?
588875947168511582764514135070217555354479805240439407753451354223283\ETC.
! TeX capacity exceeded, sorry [main memory size=5000000].
<inserted text> 666515098

l.15 ...ate\write\out{\xintXRandomDigits{5000000}}

No pages of output.
Transcript written on temp.log.
```

This can be lifted by increasing the TEX memory settings (installation dependent).

TEX-hackers note: the digits are produced eight by eight by the same method which would result from \xintUniformDeviate{100000000} but with less overhead.

12.58. (WIP) \xintiiRandRange

code

f ★ \xintiiRandRange{A} expands to a random (big) integer N such that 0<=N<A. It is a supporting macro for randrange(). As with Python's function of the same name, it is an error if A<=0.

```
\xintsetrandomseed 271828314
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx\newline
\xintiiRandRange{\xintNum{1e40}}\newline
\xintsetrandomseed 271828314
\xinttheiiexpr randrange(num(1e40))\relax\newline
% bare 1e40 not understood by \xintiiexpr
\xintsetrandomseed 271828314
\xinttheexpr randrange(1e40)\relax
```
```
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1408107837990425263001878034077495278697
1408107837990425263001878034077495278697
1408107837990425263001878034077495278697
```

Of course, keeping in mind that the set of seeds is of cardinality 2^{28}, randomness is a bit illusory here say with A=10^N, N>8, if we proceed immediately after having set the seed. If we add some entropy in any way, then it is slightly more credible; but I think that for each seed the period is something like 2^{27}(2^{55}-1)55,[71] so we expect at most about 2^{110}55 ``points in time'',

[71] Compare the result of exercise 3.2.2-30 in TAOCP, vol II.

and this is already small compared to the 10^40 from example above. Thus already we are very far from being intrinsically able to generate all numbers with forty digits as random numbers, and this makes the previous section about usage of \xintXRandomDigits to generate millions of digits a bit comical...

TEX-hackers note: the digits are produced eight by eight by the same method which would result from \xintUniformDeviate{100000000} but with less overhead.

12.59. (WIP) \xintiiRandRangeAtoB

code

f f ⋆ \xintiiRandRangeAtoB{A}{B} expands to a random (big) integer N such that A<=N<B. It is a supporting macro for randrange(). As with Python's function of the same name, it is an error if B<=A.

```
\xintsetrandomseed 271828314
12345678911111111111111111111\newline
\xintiiRandRangeAtoB{12345678911111111111111111111}{12345678922222222222222222222}%
\newline
\xintsetrandomseed 271828314
\def\test{%
\xinttheiiexpr
            randrange(12345678911111111111111111111,12345678922222222222222222222)
\relax}%
\romannumeral\xintreplicate{10}{\test\newline}%
12345678922222222222222222222
```

```
12345678911111111111111111111
12345678916037426188606389808
12345678916037426188606389808
12345678916060337223949101536
12345678912190033095886250034
12345678917323740152668511995
12345678915424847208552293485
12345678921595726610650510660
12345678911673261982088192858
12345678911339325803675947159
12345678917791540296982027151
12345678913602899909728811895
12345678922222222222222222222
```

TEX-hackers note: the digits are produced eight by eight by the same method which would result from \xintUniformDeviate{100000000} but with less overhead.

13. Macros of the xintfrac package

.1 Introduction 172
.2 \xintTeXFromSci 172
.3 \xintTeXFrac 173
.4 \xintTeXsignedFrac 173
.5 \xintTeXOver 174
.6 \xintTeXsignedOver 174
.7 \xintLen 174
.8 \xintNum 174
.9 \xintRaw 174
.10 \xintRawBraced 175
.11 \xintNumerator 175
.12 \xintDenominator 175
.13 \xintRawWithZeros 175
.14 \xintREZ 175
.15 \xintIrr 176
.16 \xintPIrr 176
.17 \xintJrr 176
.18 \xintPRaw 176
.19 \xintDecToStringREZ 177
.20 \xintDecToString 177
.21 \xintFracToSci 178
.22 \xintFracToDecimal 179
.23 \xintTrunc 179
.24 \xintXTrunc 180
.25 \xintTFrac 182
.26 \xintRound 183
.27 \xintFloor 183
.28 \xintCeil 183
.29 \xintiTrunc 183
.30 \xintTTrunc 184
.31 \xintiRound 184
.32 \xintiFloor 184
.33 \xintiCeil 185
.34 \xintE 185
.35 \xintCmp 185
.36 \xintEq 185
.37 \xintNotEq 185
.38 \xintGeq 185
.39 \xintGt 185
.40 \xintLt 185
.41 \xintGtorEq 185
.42 \xintLtorEq 185
.43 \xintIsZero 186
.44 \xintIsNotZero 186
.45 \xintIsOne 186
.46 \xintOdd 186
.47 \xintEven 186
.48 \xintifSgn 186
.49 \xintifZero 186
.50 \xintifNotZero 186
.51 \xintifOne 186
.52 \xintifOdd 186
.53 \xintifCmp 187
.54 \xintifEq 187
.55 \xintifGt 187
.56 \xintifLt 187
.57 \xintifInt 187
.58 \xintSgn 187
.59 \xintSignBit 187
.60 \xintOpp 187
.61 \xintAbs 187
.62 \xintAdd 187
.63 \xintSub 187
.64 \xintMul 188
.65 \xintDiv 188
.66 \xintDivFloor 188
.67 \xintMod 188
.68 \xintDivMod 188
.69 \xintDivTrunc 188
.70 \xintModTrunc 188
.71 \xintDivRound 189
.72 \xintSqr 189
.73 \xintPow 189
.74 \xintFac 189
.75 \xintBinomial 189
.76 \xintPFactorial 189
.77 \xintMax 190
.78 \xintMin 190
.79 \xintMaxof 190
.80 \xintMinof 190
.81 \xintSum 190
.82 \xintPrd 190
.83 \xintGCD 190
.84 \xintLCM 191
.85 \xintGCDof 191
.86 \xintLCMof 191
.87 \xintDigits, \xinttheDigits 191
.88 \xintSetDigits 192
.89 \xintFloat 192
.90 \xintFloatBraced 193
.91 \xintFloatToDecimal 194
.92 \xintPFloat 194
.93 \xintFloatAdd 198
.94 \xintFloatSub 198
.95 \xintFloatMul 198
.96 \xintFloatDiv 198
.97 \xintFloatPow 199
.98 \xintFloatPower 199
.99 \xintFloatSqrt 199
.100 \xintFloatFac 200
.101 \xintFloatBinomial 200
.102 \xintFloatPFactorial 201

13.1. Introduction

First version of this package was in release 1.03 (2013/04/14) of the xint bundle.

At release 1.3 (2018/02/28) the behaviour of \xintAdd (and of \xintSub) was modified: when adding a/b and c/d they will use always the least common multiple of the denominators. This helps limit the build-up of denominators, but the author still hesitates if the fraction should be reduced to smallest terms. The current method allows (for example when multiplying two polynomials) to keep a well-predictable denominator among various terms, even though some may be reducible.

xintfrac loads automatically xintcore and xint and inherits their macro definitions. Only these two are redefined: \xintNum and \xintLen. As explained in subsection 9.4 and subsection 9.5 the interchange format for the xintfrac macros, i.e. A/B[N], is not understood by the ii-named macros of xintcore/xint which expect the so-called strict integer format. Hence, to use such an ii-macro with an output from an xintfrac macro, an extra \xintNum wrapper is required. But macros already defined by xintfrac cover most use cases hence this should be a rarely needed.

Frac *f*

In the macro descriptions, the variable f and the margin indicator stand for the xintfrac input format for integers, scientific numbers, and fractions as described in subsection 9.4.

num *x*

As in the xint.sty documentation, x stands for something which internally will be handled in a \numexpr. It may thus be an expression as understood by \numexpr but its evaluation and intermediate steps must obey the TEX bound.

The output format for most macros is the A/B[N] format but naturally the float macros use the scientific notation on output. And some macros are special, for example \xintTrunc produces decimal numbers, \xintIrr produces an A/B with no [N], \xintiTrunc and \xintiRound produce integers without trailing [N] either, etc...

At 1.4g, old legacy typesetting macros \xintFrac, \xintSignedFrac, \xintFwOver and \xintSignedFwOver were renamed into \xintTeXFrac, \xintTeXsignedFrac, \xintTeXOver, \xintTeXsignedOver. The old names will raise errors and will be removed completely soon.

CHANGED AT 1.4m!

13.2. \xintTeXFromSci

code

☆ Experimental. This typesetting math-mode-only macro expects an input which is already in, or will expand to, decimal or scientific notation. A trailing /B is accepted and will be handled differently according to whether it follows some scientific exponent eN part or not.

It was formerly \xintTeXfromSci. Old name deprecated at 1.4l. Also it used to be *f*-expandable but is now only *x*-expandable. Use \expanded if needed.

This macro can be used as a typesetting wrapper for \xinteval or \xintfloateval output: it expects its input (after expansion) to have been already "prettified" for example via the removal of trailing zeros, usage of fixed point notation if scientific exponent is small, etc... It simply transforms the e<exponent> part, if actually present, into \cdot 10^{exponent}. A fractional part /B if found in the expansion of the input must be last and will be tranformed into \cdot B^{-1} if there was a scientific part, else the output will be using \frac{A}{B} (or the TEX equivalent in place of \frac) with A the numerator.

TEX-hackers note:

- I am hesitating whether the \frac{A}{B} branch choice should require A to be an integer, or will also, as currently, be done with A being a number in decimal notation. Please advise.
- The package does:

```
\ifdefined\frac
  \protected\def\xintTeXFromScifracmacro#1#2{\frac{#2}{#1}}%
\else
  \protected\def\xintTeXFromScifracmacro#1#2{{#2\over#1}}%
\fi
```

Customize as desired. Notice the interversion of arguments.

Example:

```
  $\xintTeXFromSci{\xintfloateval{1.1^10000/5}}$,
  $\xintTeXFromSci{\xinteval{1.1^10000/5}}$\par
```

$1.689980050240070 \cdot 10^{413}$, $8.449900251200348 \cdot 10^{413} \cdot 5^{-1}$

The above examples are in the case of a single numerical value. To handle more complex outputs from \xinteval or \xintfloateval one will need to proceed via a redefinition of \xintfloatexprPrintOne and/or \xintexprPrintOne like this:

```
  \[\def\xintfloatexprPrintOne[#1]#2{\xintTeXFromSci{\xintPFloat[#1]{#2}}}
    \xintfloateval[10]{2^100, 3^100, 13^100}\]
```

$$1.267650600 \cdot 10^{30}, 5.153775207 \cdot 10^{47}, 2.479335111 \cdot 10^{111}$$

```
  \[\def\xintexprPrintOne#1{\xintTeXFromSci{\xintFracToSci{#1}}}
    \xinteval{sqrt(2^101,60), 355/113, 6.02e23/1000}\]
```

$$1.592262918131443141159535896304 \cdot 10^{15}, \frac{355}{113}, 6.02 \cdot 10^{23} \cdot 1000^{-1}$$

This will however make then impossible, due to the added TEX mark-up in the output, the nesting of \xintfloateval/\xinteval inside one another. The core \xintexpr...\relax syntax remains usable and is anyhow the recommended way for such nesting as it is more efficient.

Some similar effect is also possible without \xintTeXFromSci, simply by a customization of \xintPFloatE like this:

```
  \begingroup
  \def\xintPFloatE#1.{\cdot10^{#1}.}%
  $\xintfloateval{1.1^10000/5}$, $\xinteval{1.1^10000/5}$
  \endgroup\newline
```

$1.689980050240070 \cdot 10^{413}$, $8.449900251200348 \cdot 10^{413}/5$

This method is simpler-minded but will leave the trailing /B's "as is", even if the numerator has no scientific exponent part. The presence of extra TEX mark-up in the output has the consequences on nesting which were mentioned above.

13.3. \xintTeXFrac

code

Frac *f* ★

This is a typesetting LATEX only macro, math mode only as it expands to \frac{A}{B}10^n for an input which ends up parsed into raw format A/B[n].

If the denominator B is 1, the output is A\cdot 10^n. If the exponent n is 0, the [\cdot]10^n part is omitted.

```
  $\xintTeXFrac {178.000/25600000}$, $\xintTeXFrac {178.000/1}$,
  $\xintTeXFrac {3.5/5.7}$\newline
```

$\frac{178000}{25600000}10^{-3}$, $178000 \cdot 10^{-3}$, $\frac{35}{57}$

The input, if in a fraction form, is not simplified in any way, except for transforming numerator and denominator into integers, separating a power of ten part. Macros such as \xintIrr, \xintPIrr, \xintREZ can be inserted to wrap the input and help simplify it. The minus sign ends up in the numerator.

CHANGED AT 1.4m! It is the new name since 1.4g of \xintFrac. The old name now raises a TEX error.

13.4. \xintTeXsignedFrac

code

Frac *f* ★

This is as \xintTeXFrac except that a negative fraction has the sign ending up in front, not in the numerator.

```
  $\xintTeXFrac {-355/113}=\xintTeXsignedFrac {-355/113}$\newline
```

$\frac{-355}{113} = -\frac{355}{113}$

CHANGED AT 1.4m! It is the new name since 1.4g of \xintSignedFrac. The old name now raises a TEX error.

code
13.5. \xintTeXOver

Frac
f ★ This does the same as \xintTeXFrac except that the \over primitive is used for the fraction (in case the denominator is not one; and a pair of braces contains the A\over B part).

```
  $\xintTeXOver {178.000/25600000}$, $\xintTeXOver {178.000/1}$,
  $\xintTeXOver {3.5/5.7}$\newline
```

$\frac{178000}{25600000}10^{-3}$, $178000\cdot 10^{-3}$, $\frac{35}{57}$

CHANGED AT 1.4m! It is the new name since 1.4g of \xintFwOver. The old name now raises a TEX error.

code
13.6. \xintTeXsignedOver

Frac
f ★ This is as \xintTeXOver except that a negative fraction has the sign put in front, not in the numerator.

```
  $\xintTeXOver{-355/113}=\xintTeXsignedOver{-355/113}$\newline
```

$\frac{-355}{113} = -\frac{355}{113}$

CHANGED AT 1.4m! It is the new name since 1.4g of \xintSignedFwOver. The old name now raises a TEX error.

code
13.7. \xintLen

Frac
f ★ The \xintLen macro from xint is extended to accept a fraction on input: the length of A/B[n] is the length of A plus the length of B plus the absolute value of n and minus one (an integer input as N is internally represented in a form equivalent to N/1[0] so the minus one means that the extended \xintLen behaves the same as the original for integers).

```
  \xintLen{201710/298219}=\xintLen{201710}+\xintLen{298219}-1\newline
  \xintLen{1234/1}=\xintLen{1234}=\xintLen{1234[0]}=\xintiLen{1234}\newline
  \xintLen{-1e3/5.425} (\xintRaw {-1e3/5.425})\par
```

```
11=6+6-1
4=4=4=4
10 (-1/5425[6])
```

The length is computed on the A/B[n] which would have been returned by \xintRaw, as illustrated by the last example above.

\xintLen is only for use with such (scientific) numbers or fractions. See also \xintNthElt from xinttools. See also \xintLength (which however does not expand its argument) from xintkernel for counting more general tokens (or rather braced items).

code
13.8. \xintNum

Frac
f ★ The \xintNum from xint is transformed into a synonym to \xintTTrunc.

Attention that for example \xintNum{1e100000} expands to the needed 100001 digits...

The original \xintNum from xintcore which does not understand the fraction slash or the scientific notation is still available under the name \xintiNum.

code
13.9. \xintRaw

Frac
f ★ This macro `prints' the fraction f as it is received by the package after its parsing and expansion, in a form A/B[N] equivalent to the internal representation: the denominator B is always strictly positive and is printed even if it has value 1.

```
  \xintRaw{\the\numexpr 571*987\relax.123e-10/\the\numexpr-201+59\relax e-7}
```

```
-563577123/142[-6]
```

No simplification is done, not even of common zeroes between numerator and denominator:

```
  \xintRaw {178000/25600000}
```

```
178000/25600000[0]
```

13.10. \xintRawBraced

code

Frac *f* ★ This macro expands and parses its input f as all xintfrac macros and produces as output {N}{A}{B} (with TEX braces) where \xintRaw would have returned A/B[N].

13.11. \xintNumerator

code

Frac *f* ★ The input data is parsed as if by \xintRaw into A/B[N] and then A is returned if N<=0, or A extended by N zeroes if N>0.

```
  \xintNumerator {178000/25600000[17]}\newline
  \xintNumerator {312.289001/20198.27}\newline
  \xintNumerator {178000e-3/256e5}\newline
  \xintNumerator {178.000/25600000}
```

```
17800000000000000000000
312289001
178000
178000
```

13.12. \xintDenominator

code

Frac *f* ★ The input data is parsed as if by \xintRaw into A/B[N] and then B is returned if N>0, or B extended by |N| zeroes if N<=0.

```
  \xintDenominator {178000/25600000[17]}\newline
  \xintDenominator {312.289001/20198.27}\newline
  \xintDenominator {178000e-3/256e5}\newline
  \xintDenominator {178.000/25600000}
```

```
25600000
20198270000
25600000000
25600000000
```

13.13. \xintRawWithZeros

code

Frac *f* ★ This macro parses the input and outputs A'/B', with A' as would be returned by \xintNumerator{f} and B' as would be returned by \xintDenominator{f}.

```
  \xintRawWithZeros{178000/25600000[17]}\newline
  \xintRawWithZeros{312.289001/20198.27}\newline
  \xintRawWithZeros{178000e-3/256e5}\newline
  \xintRawWithZeros{178.000/25600000}\newline
  \xintRawWithZeros{\the\numexpr 571*987\relax.123e-10/\the\numexpr-201+59\relax e-7}
```

```
1780000000000000000000/25600000
312289001/20198270000
178000/25600000000
178000/25600000000
-563577123/142000000
```

13.14. \xintREZ

code

Frac *f* ★ The input is first parsed into A/B[N] as by \xintRaw, then trailing zeroes of A and B are suppressed

and N is accordingly adjusted.

```
\xintREZ {178000/25600000[17]}
```

```
178/256[15]
```

13.15. \xintIrr

code

Frac f ★ This puts the fraction into its unique irreducible form:

```
\xintIrr {178.256/256.1780}, \xintIrr {178000/25600000[17]}
```

```
6856/9853, 695312500000000/1
```

The current implementation does not cleverly first factor powers of 2 and 5, and \xintIrr {2/3[100]} will execute the Euclidean division of 2·10^{100} by 3, which is a bit stupid as it could have known that the 100 trailing zeros can not bring any divisibility by 3.

Starting with release 1.08, \xintIrr does not remove the trailing /1 when the output is an integer. This was deemed better for various (questionable?) reasons, anyway the output format is since *always* A/B with B>0, even in cases where it turns out that B=1. Use \xintPRaw on top of \xintIrr if it is needed to get rid of such a trailing /1.

13.16. \xintPIrr

code

Frac f ★ This puts the fraction into irreducible form, *keeping as is the decimal part* [N] from raw internal A/B[N] format. (P stands here for *Partial*)

```
\xintPIrr {178.256/256.1780}, \xintPIrr {178000/25600000[17]}
```

```
3428/49265[1], 89/12800[17]
```

Notice that the output always has the ending [N], which is exactly the opposite of \xintIrr's behaviour. The interest of this macro is mainly in handling fractions which somehow acquired a big [N] (perhaps from input in scientific notation) and for which the reduced fraction would have a very large number of digits. This large number of digits can considerably slow-down computations done afterwards.

For example package polexpr uses \xintPIrr when differentiating a polynomial, or in setting up a Sturm chain for localization of the real roots of a polynomial. This is relevant to polynomials whose coefficients were input in decimal notation, as this automatically creates internally some [N]. Keeping and combining those [N]'s during computations significantly increases their speed.

13.17. \xintJrr

code

Frac f ★ This also puts the fraction into its unique irreducible form:

```
\xintJrr {178.256/256.178}
```

```
6856/9853
```

This is (supposedly, not tested for ages) faster than \xintIrr for fractions having some big common factor in the numerator and the denominator.

```
\xintJrr {\xintiiPow{\xintiiFac {15}}{3}/%
              \xintiiPrd{{\xintiiFac{10}}{\xintiiFac{30}}{\xintiiFac{5}}}}
```

```
1001/51705840
```

But to notice the difference one would need computations with much bigger numbers than in this example. As \xintIrr, \xintJrr does not remove the trailing /1 from a fraction reduced to an integer.

13.18. \xintPRaw

code

Frac f ★ PRaw stands for ``pretty raw''. It does like \xintRaw apart from removing the [N] part if N=0 and removing the B if B=1.

```
\xintPRaw {123e10/321e10}, \xintPRaw {123e9/321e10}, \xintPRaw {\xintIrr{861/123}}
123/321, 123/321[-1], 7
```

13.19. \xintDecToStringREZ

code

Frac *f* ★

\xintDecToStringREZ uses fixed point (decimal) notation for the output. The REZ means that it trims (REmoves) trailing Zeros. The name is a bit strange, because it its not limited to *decimal numbers* but accepts the same kind of inputs as most other xintfrac macros. The parsing of this input transforms it first into an internal format having a numerator A, a denominator B and a power of ten exponent N, standing for the fraction A/B times 10 to the power N. Then the following recipe applies:

- the zero value is printed as 0 (no decimal point).
- trailing zeros of A and B are removed and N is adjusted,
- if the new B is not 1, it will appear in the output as /B,
- fixed point decimal notation is used for AeN:
  - if N is non-negative, the output is an integer with N trailing zeros (and no decimal mark)
  - if N is negative a decimal point is used, and if AeN is less than one in absolute value, output will start with 0. (i.e. a decimal mark).

The following should be noted:

1. the fraction AeN/B or even A/B is not pre-reduced into lowest terms,
2. the macro does not check if B contains only powers of 2 and 5, so 1/2 is printed as 1/2, not as 0.5.

The definitive behaviour remains to be decided regarding these last two points.

```
\xintDecToStringREZ{0}, \xintDecToStringREZ{1/2}, \xintDecToStringREZ{0.5000}\newline
\xintDecToStringREZ{1.23456789e5}, \xintDecToStringREZ {1.23456789e-3}\newline
\xintDecToStringREZ{12345e-1}, \xintDecToStringREZ {12345e-2},
\xintDecToStringREZ{12345e-3}\newline
\xintDecToStringREZ{12345e-4}, \xintDecToStringREZ {12345e-5},
\xintDecToStringREZ{12345e-6}\newline
\xintDecToStringREZ{1.234567890000e12}, \xintDecToStringREZ{1.23456000e-5/10}\newline
\xintDecToStringREZ{70/14} % is not reduced to lowest terms
0, 1/2, 0.5
123456.789, 0.00123456789
1234.5, 123.45, 12.345
1.2345, 0.12345, 0.012345
1234567890000, 0.00000123456
70/14
```

See \xintFloatToDecimal for a variant which first rounds the input to some given number of significant digits.

13.20. \xintDecToString

code

Frac *f* ★

\xintDecToString uses fixed point notation for the output. Is behaviour remains somewhat undecided in so far as whether it should identify inputs which correspond to decimal numbers (i.e. fractions with only powers of two and five in their denominator, once reduced to lowest terms).

As with \xintDecToStringREZ, the name is a bit strange as inputs are in no way limited to decimal numbers but are of the most general type accepted by the xintfrac macros.

It is the same macro as \xintDecToStringREZ except that it does not remove trailing zeros, in fact \xintDecToStringREZ{f} is defined as \xintDecToString{\xintREZ{f}}.

```
\xintDecToString{0}, \xintDecToString{1/2}, \xintDecToString{0.5000}\newline
\xintDecToString{1.23456789e5}, \xintDecToString {1.23456789e-3}\newline
\xintDecToString{12345e-1}, \xintDecToString {12345e-2}, \xintDecToString{12345e-3}%
\newline
\xintDecToString{12345e-4}, \xintDecToString {12345e-5}, \xintDecToString{12345e-6}%
\newline
\xintDecToString{1.234567890000e12}, \xintDecToString{1.23456000e-5/10}\newline
\xintDecToString{70/14}
```

```
0, 1/2, 0.5000
123456.789, 0.00123456789
1234.5, 123.45, 12.345
1.2345, 0.12345, 0.012345
1234567890000, 0.0000123456000/10
70/14
```

Since 1.4e, \xintDecToString is the default for \xintiexprPrintOne, which governs the \xintieval output format: in this use case there is never a /B fractional part and the output is always either an integer (if \xintieval was used without optional argument) or a decimal string

```
\def\xintiexprPrintOne{\xintDecToString}
```

Any replacement of \xintDecToString as the expansion of \xintiexprPrintOne should obey the following blueprint:

☆
- to be expandable, but not necessarily *f*-expandable,
- to accept on input A or A[N].

13.21. \xintFracToSci

code

Frac *f* ★

\xintFracToSci was initially at 1.4 a private macro which served as default customization of \xintexprPrintOne and, despite being documented in the user manual, was not supposed to be used at user level (not being *f*-expandable it could not be nested within macros of xintfrac, and besides accepted a limited range of inputs).

It has been upgraded at 1.4l to behave like all other xintfrac macros.

Here is what it does:

- it first parses the input like any other xintfrac macro and convert it into the ``raw'' A/B[N] format,
- it then produces this output: A/B if N=0 (and /B is omitted if not 1), and for N not zero, the output numerator will be AeN written in scientific notation exactly like it would by \xintPFloat but without of course prior rounding to a given number of digits; the trailing zeros in the significand will be removed always (the \xintPFloatMinTrimmed configuration is ignored). Then this value in scientific notation (or in decimal fixed point notation if the scientific exponent is in the \xintPFloatNoSciEmin to \xintPFloatNoSciEmax range) will be attached to a trailing denominator /B (omitted if it is /1).

Please note:

- there is no reduction of the fraction A/B to lowest terms,
- trailing zeros in the integer denominator B are not moved and incorporated into the final scientific exponent,
- no attempt is made to check if B is a product of only 2's and 5's and thus could be integrated into some pure decimal notation for the numerator or at least its significand.

Changes of \xintPFloat at 1.4k have an impact here. In particular the zero value will give 0 whether the input was some 0, 0e-5, 0/3, 0.00, etc. . . , whereas at 1.4e it would have been 0.0 for cases triggering some \xintPFloat subroutine.

The general blueprint is still to be considered *unstable*.

The output routine of \xinteval is customizable via redefining \xintexprPrintOne whose current default is (equivalent to):

```
\def\xintexprPrintOne{\xintFracToSci}
```

13.22. \xintFracToDecimal

code

Frac *f* ★

\xintFracToDecimal is a variant of \xintFracToSci which differs from it in so far as it outputs a numerator using decimal notation, i.e. with as many zeros as are needed (and no more) and no scientific exponent. The denominator goes through ``as is'' except if it is 1, then it is omitted.

In other terms its behaviour is currently intermediate between \xintDecToString and \xintDecToStringREZ, as it does not remove trailing zeros of the denominator. Consider its behaviour as *unstable*.

It can be used to customize \xintexprPrintOne:

```
\def\xintexprPrintOne{\xintFracToDecimal}
```

It was initially at 1.4k a private macro which served as an alternative to \xintFracToSci default customization of \xintexprPrintOne and, despite being documented in the user manual, was not supposed to be used at user level (not being *f*-expandable it could not be nested within macros of xintfrac, and besides accepted a limited range of inputs).

It has been upgraded at 1.4l to behave like all other xintfrac macros.

13.23. \xintTrunc

code

num *x* Frac *f* ★

\xintTrunc{x}{f} returns the start of the decimal expansion of the fraction f, truncated to:

- if x>0, x digits after the decimal mark,
- if x=0, an integer,
- if x<0, an integer multiple of 10^{-x} (in scientific notation).

The output is the sole digit token 0 if and only if the input was exactly zero; else it contains always either a decimal mark (even if x=0) or a scientific part and it conserves the sign of f (even if the truncated value represents the zero value).

Truncation is done towards zero.

```
\begin{multicols}{2}
  \noindent\xintFor* #1 in {\xintSeq[-1]{7}{-14}}:{\xintTrunc{#1}{-11e12/7}\newline}%
  \xintTrunc{10}{1e-11}\newline \xintTrunc{10}{1/65536}\par
\end{multicols}
```

-1571428571428.5714285
-1571428571428.571428
-1571428571428.57142
-1571428571428.5714
-1571428571428.571
-1571428571428.57
-1571428571428.5
-1571428571428.
-157142857142e1
-15714285714e2
-1571428571e3
-157142857e4
-15714285e5
-1571428e6
-157142e7
-15714e8
-1571e9
-157e10
-15e11
-1e12
-0e13
-0e14
0.0000000000
0.0000152587

Warning: *it is not yet decided is the current behaviour is definitive.*

Currently xintfrac has no notion of a positive zero or a negative zero. Hence transitivity of \xintTrunc is broken for the case where the first truncation gives on output 0.00...0 or -0.00...0: a second truncation to less digits will then output 0, whereas if it had been applied directly to the initial input it would have produced 0.00...0 or respectively -0.00...0 (with less zeros after decimal mark).

If xintfrac distinguished zero, positive zero, and negative zero then it would be possible to maintain transitivity.

The problem would also be fixed, even without distinguishing a negative zero on input, if \xintTrunc always produced 0.00...0 (with no sign) when the mathematical result is zero, discarding the information on original input being positive, zero, or negative.

I have multiple times hesitated about what to do and must postpone again final decision.

13.24. \xintXTrunc

code

num Frac
x f ☆

\xintXTrunc{x}{f} is similar to \xintTrunc with the following important differences:

- it is completely expandable but not *f*-expandable, as is indicated by the hollow star in the margin,
- hence it can not be used as argument to the other package macros, but as it *f*-expands its {f} argument, it accepts arguments expressed with other xintfrac macros,
- it requires x>0,
- contrarily to \xintTrunc the number of digits on output is not limited to about 19950 and may go well beyond 100000 (this is mainly useful for outputting a decimal expansion to a file),
- when the mathematical result is zero, it always prints it as 0.00...0 or -0.00...0 with x zeros after the decimal mark.

Warning: transitivity is broken too (see discussion of \xintTrunc), due to the sign in the last item. Hence *the definitive policy is yet to be fixed.*

Transitivity is here in the sense of using a first \edef and then a second one, because it is not possible to nest \xintXTrunc directly as argument to itself. Besides, although the number of digits on output isn't limited, nevertheless x should be less than about 19970 when the number of digits of the input (assuming it is expressed as a decimal number) is even bigger: \xintXTrunc{30000}{\Z} after \edef\Z{\xintXTrunc{60000}{1/66049} raises an error in contrast with a direct \xintXTrunc{30000}{1/66049}. But \xintXTrunc{30000}{123.456789} works, because here the number of digits originally present is smaller than what is asked for, thus the routine only has to add trailing zeros, and this has no limitation (apart from TEX main memory).

\xintXTrunc will expand fully in an \edef or a \write (\message, \wlog, ...) or in an \xintexpr-ession, or as list argument to \xintFor*.

Here is an example session where the user checks that the decimal expansion of $1/66049 = 1/257^2$ has the maximal period length $257 * 256 = 65792$ (this period length must be a divisor of $\varphi(66049)$ and to check it is the maximal one it is enough to show that neither 32896 nor 256 are periods.)

```
$ rlwrap etex -jobname worksheet-66049
This is pdfTeX, Version 3.14159265-2.6-1.40.17 (TeX Live 2016) (preloaded format=etex)
 restricted \write18 enabled.
**xintfrac.sty
entering extended mode
(/usr/local/texlive/2016/texmf-dist/tex/generic/xint/xintfrac.sty
(/usr/local/texlive/2016/texmf-dist/tex/generic/xint/xint.sty
(/usr/local/texlive/2016/texmf-dist/tex/generic/xint/xintcore.sty
(/usr/local/texlive/2016/texmf-dist/tex/generic/xint/xintkernel.sty))))
*% we load xinttools for \xintKeep, etc... \xintXTrunc itself has no more

*% any dependency on xinttools.sty since 1.2i
```

```
*\input xinttools.sty
(/usr/local/texlive/2016/texmf-dist/tex/generic/xint/xinttools.sty)
*\def\m#1;{\message{#1}}

*\m \the\numexpr 257*257\relax;
66049
*\m \the\numexpr 257*256\relax;
65792
*% Thus 1/66049 will have a period length dividing 65792.

*% Let us first check it is indeed periodical.

*\edef\Z{\xintXTrunc{66000}{1/66049}}

*% Let's display the first decimal digits.

*\m \xintXTrunc{208}{\Z};

0.000015140274644582052718436312434707565595239897651743402625323623370527941376
856576178291874214598252812306015231116292449545034746930309315810988811337037
653863041075565110751109025117715635361625459885
*% let's now fetch the trailing digits

*\m \xintKeep{65792-66000}{\Z};% 208 trailing digits

000015140274644582052718436312434707565595239897651743402625323623370527941376856576178291874214598252812306015231116292449545034746930309315810988811337037653863041075565110751109025117715635361625459885842329179851322502990204242304955411891171705854744205059879786219322018501415615679268421929173795212645157383154930430438008145467758785144362518736089872670290239064936637950612424109373344032460748837983921028327453860013020636194340565337855228693848506411906311980499326257778316098654029584096655513330
*% now with 65792/2=32896. Problem: we can't do \xintXTrunc{32896+100}{\Z}

*% but only direct \xintXTrunc{32896+100}{1/66049}. Anyway we want to nest it

*% hence let's do it all with (slower) \xintKeep, \xintKeepUnbraced.

*\m \xintKeep {-100}{\xintKeepUnbraced{2+65792/2+100}{\Z}};

999984859725355417947281563687565292434404760102348256597374676376629472058623143423821708125785401 7
*% This confirms 32896 isn't a period length.
```

```
*% To conclude let's write the 66000 digits to the log.

*\wlog{\Z}

*% We want always more digits:

*\wlog{\xintXTrunc{150000}{1/66049}}

*\bye
```

The acute observer will have noticed that there is something funny when one compares the first digits with those after the middle-period:

```
0000151402746445820527184363124347075655952398976517434026253236233705279413768...
9999848597253554179472815636875652924344047601023482565973746763766294720586231...
```

Mathematical exercise: can you explain why the two indeed add to 9999...9999?

You can try your hands at this simpler one:

```
1/49=\xintTrunc{42+5}{1/49}...\newline
\xintTrim{2}{\xintTrunc{21}{1/49}}\newline
\xintKeep{-21}{\xintTrunc{42}{1/49}}
```

```
1/49=0.020408163265306122448979591836734693877551020 40...
020408163265306122448
979591836734693877551
```

This was again an example of the type 1/N with N the square of a prime. One can also find counter-examples within this class: 1/31^2 and 1/37^2 have an odd period length (465 and respectively 111) hence they can not exhibit the symmetry.

> Mathematical challenge: prove generally that if the period length of the decimal expansion of 1/p^r (with p a prime distinct from 2 and 5 and r a positive exponent) is even, then the previously observed symmetry about the two halves of the period adding to a string of nine's applies.

13.25. \xintTFrac

code

Frac
f ★

\xintTFrac{f} returns the fractional part, f=trunc(f)+frac(f). Thus if f<0, then -1<frac(f)<=0 and if f>0 one has 0<= frac(f)<1. The T stands for `Trunc', and there should exist also similar macros associated respectively with `Round', `Floor', and `Ceil', each type of rounding to an integer deserving arguably to be associated with a fractional ``modulo''. By sheer laziness, the package currently implements only the ``modulo'' associated with `Truncation'. Other types of modulo may be obtained more cumbersomely via a combination of the rounding with a subsequent subtraction from f.

Notice that the result is filtered through \xintREZ, and will thus be of the form A/B[N], where neither A nor B has trailing zeros. But the output fraction is not reduced to smallest terms.

The function call in expressions (\xintexpr, \xintfloatexpr) is frac. Inside \xintexpr..\relax, the function frac is mapped to \xintTFrac. Inside \xintfloatexpr..\relax, frac first applies \xintTFrac to its argument (which may be an exact fraction with more digits than the floating point precision) and only in a second stage makes the conversion to a floating point number with the precision as set by \xintDigits (default is 16).

```
\xintTFrac {1235/97}, \xintTFrac {-1235/97}\newline
\xintTFrac {1235.973}, \xintTFrac {-1235.973}\newline
\xintTFrac {1.122435727e5}\par
```

```
71/97[0], -71/97[0]
973/1[-3], -973/1[-3]
```

```
5727/1[-4]
```

code
13.26. \xintRound

num Frac x f ★ \xintRound{x}{f} returns the start of the decimal expansion of the fraction f, rounded to:

- if x>0, x digits after the decimal mark,
- if x=0, an integer,
- if x<0, an integer multiple of 10^{-x} (in scientific notation).

The output is the sole digit token 0 if and only if the input was exactly zero; else it contains always either a decimal mark (even if x=0) or a scientific part and it conserves the sign of f (even if the rounded value represents the zero value).

```
\begin{multicols}{2}
  \noindent\xintFor* #1 in {\xintSeq[-1]{7}{-14}}:{\xintRound{#1}{-11e12/7}\newline}%
  \xintRound{10}{1e-11}\newline \xintRound{10}{1/65536}\newline
\end{multicols}
```

```
-1571428571428.5714286
-1571428571428.571429
-1571428571428.57143
-1571428571428.5714
-1571428571428.571
-1571428571428.57
-1571428571428.6
-1571428571429.
-157142857143e1
-15714285714e2
-1571428571e3
-157142857e4
-15714286e5
-1571429e6
-157143e7
-15714e8
-1571e9
-157e10
-16e11
-2e12
-0e13
-0e14
0.0000000000
0.0000152588
```

Rounding is done with half-way numbers going towards infinity of the same sign.

code
13.27. \xintFloor

Frac f ★ \xintFloor {f} returns the largest relative integer N with N ⩽ f.

```
\xintFloor {-2.13}, \xintFloor {-2}, \xintFloor {2.13}
```

-3/1[0], -2/1[0], 2/1[0] Note the trailing [0], see \xintiFloor if it is not desired.

code
13.28. \xintCeil

Frac f ★ \xintCeil {f} returns the smallest relative integer N with N > f.

```
\xintCeil {-2.13}, \xintCeil {-2}, \xintCeil {2.13}
```

-2/1[0], -2/1[0], 3/1[0]

code
13.29. \xintiTrunc

num Frac x f ★ \xintiTrunc{x}{f} returns the integer equal to 10^x times what \xintTrunc{x}{f} would produce. Attention that leading zeros are automatically removed: the output is in strict integer format.

```
\begin{multicols}{2}
```

```
    \noindent\xintFor* #1 in {\xintSeq[-1]{7}{-14}}:{\xintiTrunc{#1}{-11e12/7}\newline}%
    \xintiTrunc{10}{1e-11}\newline \xintiTrunc{10}{1/65536}\par
  \end{multicols}
```

```
-15714285714285714285                   -15714285
-1571428571428571428                    -1571428
-157142857142857142                     -157142
-15714285714285714                      -15714
-1571428571428571                       -1571
-157142857142857                        -157
-15714285714285                         -15
-1571428571428                          -1
-157142857142                           0
-15714285714                            0
-1571428571                             0
-157142857                              152587
```

code
13.30. \xintTTrunc

Frac
f ★ \xintTTrunc{f} truncates to an integer (truncation towards zero). This is the same as \xintiTrunc {0}{f} and also the same as \xintNum.

code
13.31. \xintiRound

num Frac
x *f* ★ \xintiRound{x}{f} returns the integer equal to 10^x times what \xintRound{x}{f} would return. The output has no leading zeroes, it is always in strict integer format.

```
  \begin{multicols}{2}
    \noindent\xintFor* #1 in {\xintSeq[-1]{7}{-14}}:{\xintiRound{#1}{-11e12/7}\newline}%
    \xintiRound{10}{1e-11}\newline \xintiRound{10}{1/65536}\par
  \end{multicols}
```

```
-15714285714285714286                   -15714286
-1571428571428571429                    -1571429
-15714285714285714285                   -15714285
-1571428571428571428                    -1571428
-157142857142857143                     -157143
-15714285714285714                      -15714
-1571428571428571                       -1571
-157142857142857                        -157
-15714285714286                         -16
-1571428571429                          -2
-157142857143                           0
-15714285714                            0
-1571428571                             0
-157142857                              152588
```

code
13.32. \xintiFloor

Frac
f ★ \xintiFloor {f} does the same as \xintFloor but without the trailing /1[0].

```
  \xintiFloor {-2.13}, \xintiFloor {-2}, \xintiFloor {2.13}
```

```
-3, -2, 2
```

code
13.33. \xintiCeil

Frac
f ⋆ `\xintiCeil {f}` does the same as `\xintCeil` but its output is without the `/1[0]`.

```
  \xintiCeil {-2.13}, \xintiCeil {-2}, \xintiCeil {2.13}
-2, -2, 3
```

code
13.34. \xintE

Frac num
f *x* ⋆ `\xintE {f}{x}` multiplies the fraction `f` by 10^x. The *second* argument `x` must obey the TEX bounds. Example:

```
  \count 255 123456789 \xintE {10}{\count 255}
10/1[123456789] Don't feed this example to \xintNum!
```

code
13.35. \xintCmp

Frac Frac
f *f* ⋆ This compares two fractions `F` and `G` and produces `-1`, `0`, or `1` according to `F<G`, `F=G`, `F>G`.
For choosing branches according to the result of comparing `f` and `g`, see `\xintifCmp`.

code
13.36. \xintEq

Frac Frac
f *f* ⋆ `\xintEq{f}{g}` returns `1` if `f=g`, `0` otherwise.

code
13.37. \xintNotEq

Frac Frac
f *f* ⋆ `\xintNotEq{f}{g}` returns `0` if `f=g`, `1` otherwise.

code
13.38. \xintGeq

Frac Frac
f *f* ⋆ This compares the *absolute values* of two fractions. `\xintGeq{f}{g}` outputs `1` if `|f| ⩾ |g|` and `0` if not.
Important: the macro compares *absolute values*.

code
13.39. \xintGt

Frac Frac
f *f* ⋆ `\xintGt{f}{g}` returns `1` if `f>g`, `0` otherwise.

code
13.40. \xintLt

Frac Frac
f *f* ⋆ `\xintLt{f}{g}` returns `1` if `f<g`, `0` otherwise.

code
13.41. \xintGtorEq

Frac Frac
f *f* ⋆ `\xintGtorEq{f}{g}` returns `1` if `f⩾g`, `0` otherwise. Extended by xintfrac to fractions.

code
13.42. \xintLtorEq

Frac Frac
f *f* ⋆ `\xintLtorEq{f}{g}` returns `1` if `f⩽g`, `0` otherwise.

code
13.43. \xintIsZero

f ★ \xintIsZero{f} returns 1 if f=0, 0 otherwise.

code
13.44. \xintIsNotZero

f ★ \xintIsNotZero{f} returns 1 if f!=0, 0 otherwise.

code
13.45. \xintIsOne

f ★ \xintIsOne{f} returns 1 if f=1, 0 otherwise.

code
13.46. \xintOdd

f ★ \xintOdd{f} returns 1 if the integer obtained by truncation is odd, and 0 otherwise.

code
13.47. \xintEven

f ★ \xintEven{f} returns 1 if the integer obtained by truncation is even, and 0 otherwise.

code
13.48. \xintifSgn

code
Frac
f nnn ★ \xintifSgn {⟨*f*⟩}{⟨*A*⟩}{⟨*B*⟩}{⟨*C*⟩} executes either the ⟨*A*⟩, ⟨*B*⟩ or ⟨*C*⟩ code, depending on its first argument being respectively negative, zero, or positive.

code
13.49. \xintifZero

Frac
f nn ★ \xintifZero{⟨*f*⟩}{⟨*IsZero*⟩}{⟨*IsNotZero*⟩} expandably checks if the first mandatory argument N (a number, possibly a fraction if xintfrac is loaded, or a macro expanding to one such) is zero or not. It then either executes the first or the second branch.

Beware that both branches must be present.

code
13.50. \xintifNotZero

Frac
f nn ★ \xintifNotZero{⟨*N*⟩}{⟨*IsNotZero*⟩}{⟨*IsZero*⟩} expandably checks if the first mandatory argument f is not zero or is zero. It then either executes the first or the second branch.

Beware that both branches must be present.

code
13.51. \xintifOne

Frac
f nn ★ \xintifOne{⟨*N*⟩}{⟨*IsOne*⟩}{⟨*IsNotOne*⟩} expandably checks if the first mandatory argument f is one or not one. It then either executes the first or the second branch. Beware that both branches must be present.

code
13.52. \xintifOdd

Frac
f nn ★ \xintifOdd{⟨*N*⟩}{⟨*odd*⟩}{⟨*not odd*⟩} expandably checks if the first mandatory argument f, after truncation to an integer, is odd or even. It then executes accordingly the first or the second branch. Beware that both branches must be present.

code
13.53. \xintifCmp

rac Frac
f f nnn ★ \xintifCmp{⟨f⟩}{⟨g⟩}{⟨if f<g⟩}{⟨if f=g⟩}{⟨if f>g⟩} compares its first two arguments and chooses accordingly the correct branch.

code
13.54. \xintifEq

Frac Frac
f f nn ★ \xintifEq{⟨f⟩}{⟨g⟩}{⟨YES⟩}{⟨NO⟩} checks equality of its two first arguments and executes accordingly the YES or the NO branch.

code
13.55. \xintifGt

Frac Frac
f f nn ★ \xintifGt{⟨f⟩}{⟨g⟩}{⟨YES⟩}{⟨NO⟩} checks if f > g and in that case executes the YES branch.

code
13.56. \xintifLt

Frac Frac
f f nn ★ \xintifLt{⟨f⟩}{⟨g⟩}{⟨YES⟩}{⟨NO⟩} checks if f < g and in that case executes the YES branch.

code
13.57. \xintifInt

Frac
f nn ★ \xintifInt{f}{YES branch}{NO branch} expandably chooses the YES branch if f reveals itself after expansion and simplification to be an integer.

code
13.58. \xintSgn

Frac
f ★ The sign of a fraction.

code
13.59. \xintSignBit

Frac
f ★ Expands to 1 for negative input, to 0 else.
Added at 1.4l.

code
13.60. \xintOpp

Frac
f ★ The opposite of a fraction. Note that \xintOpp {3} produces -3/1[0] whereas \xintiiOpp {3} produces -3.

code
13.61. \xintAbs

Frac
f ★ The absolute value. Note that \xintAbs {-2}=2/1[0] where \xintiiAbs {-2} outputs =2.

code
13.62. \xintAdd

Frac Frac
f f ★ Computes the addition of two fractions.
Since 1.3 always uses the least common multiple of the denominators.

code
13.63. \xintSub

Frac Frac
f f ★ Computes the difference of two fractions (\xintSub{F}{G} computes F-G).
Since 1.3 always uses the least common multiple of the denominators.

code
13.64. \xintMul

Frac Frac
f f ★ Computes the product of two fractions.
Output is not reduced to smallest terms.

code
13.65. \xintDiv

Frac Frac
f f ★ Computes the quotient of two fractions. (\xintDiv{F}{G} computes F/G).
Output is not reduced to smallest terms.

code
13.66. \xintDivFloor

Frac Frac
f f ★ Computes the quotient of two arguments then apply floor function to get an integer (in strict format). This macro was defined at 1.1 (but was left not documented until 1.3a...) and changed at 1.2p, formerly it appended /1[0] to output.

```
\xintDivFloor{-170/3}{23/2}
-5
```

code
13.67. \xintMod

Frac Frac
f f ★ Computes the remainder associated to the floored division \xintDivFloor. Prior to 1.2p the meaning was the one of \xintModTrunc. Was left undocumented until 1.3a.

```
\xintMod{-170/3}{23/2}
5/6[0]
```

Modified at 1.3 to use a l.c.m. for the denominator of the result.

code
13.68. \xintDivMod

Frac Frac
f f ★ Computes both the floored division and the remainder \xintDivFloor. New at 1.2p and documented at 1.3a.

```
\oodef\foo{\xintDivMod{-170/3}{23/2}}\meaning\foo
macro:->{-5}{5/6[0]}
```

code
13.69. \xintDivTrunc

Frac Frac
f f ★ Computes the quotient of two arguments then truncates to an integer (in strict format).

```
\xintDivTrunc{-170/3}{23/2}
-4
```

code
13.70. \xintModTrunc

Frac Frac
f f ★ Computes the remainder associated with the truncated division of two arguments. Prior to 1.2p it was named \xintMod, but the latter then got associated with floored division.

```
\xintModTrunc{-170/3}{23/2}
-64/6[0]
```

Modified at 1.3 to use a l.c.m. for the denominator of the result.

code
13.71. \xintDivRound

Frac Frac
f f ★ Computes the quotient of the two arguments then rounds to an integer (in strict format).

```
\xintDivRound{-170/3}{23/2}
-5
```

code
13.72. \xintSqr

Frac
f ★ Computes the square of one fraction.

code
13.73. \xintPow

Frac Num
f f ★ \xintPow{f}{x}: computes f^x with f a fraction and the exponent x possibly also, but if only xintfrac is loaded it will be truncated to an integer.

At 1.4e the behaviour of the macro is enhanced if xintexpr is loaded, as it then becomes the support macro for powers a^b, a**b (and the pow() function) in \xinteval: it now handles also non-integer exponents. Also, if the exponent is an integer, it checks a priori if an exact evaluation would produce more than about 10000 digits and then does in its place a floating point evaluation.

The check whether the exponent is integer is not on the mathematical value but on the format (for reasons of efficiency). So 4/2 will not be recognized as integer and it will thus trigger usage of the floating point evaluations; however 2.0 will be recognized as an integer, as of course 2.

Still under the condition that xintexpr is loaded, if the exponent is considered an integer it is then checked if

1. it is less than 10000 (in absolute value),
2. and the output will contain less than 10000 digits separately for numerator and denominator.

If both conditions true, then the power is computed *exactly*. Else it is computed as by \xintFloatPower (but the output uses raw A[N] format not scientific notation). This remains distinct from the native integer-power routine of \xintfloateval at 2.0a2. Use \xintiiPow (on integers only, not fractions) for exact powers with larger exponents.

Before version 2.0 of xintexpr, half-integers were also identified and then the power was computed by combining suitably \xintFloatPower with \xintFloatSqrt. This has been removed.

CHANGED AT 2.0a1!

Within an \xintiiexpr..\relax the infix operators ^ and ** are mapped to \xintiiPow and powers are always computed exactly even if they would produce more than 10000 digits and melt your CPU; within an \xintexpr-ession ^ and ** and the pow() function are mapped to \xintPow as described here.

code
13.74. \xintFac

Num
f ★ This is a convenience variant of \xintiiFac which applies \xintNum to its argument. Notice however that the output will have a trailing [0] according to the xintfrac format for integers.

code
13.75. \xintBinomial

Num Num
f f ★ This is a convenience variant of \xintiiBinomial which applies \xintNum to its arguments. Notice however that the output will have a trailing [0] according to the xintfrac format for integers.

code
13.76. \xintPFactorial

Num Num
f f ★ This is a convenience variant of \xintiiPFactorial which applies \xintNum to its arguments. Notice however that the output will have a trailing [0] according to the xintfrac format for integers.

code
13.77. \xintMax

Frac Frac
f f ★ The maximum of two fractions. Beware that \xintMax {2}{3} produces 3/1[0]. The original, for use with integers only with no need of normalization, is available as \xintiiMax: \xintiiMax {2}{3}
f f ★↪ =3.

```
  \xintMax {2.5}{7.2}
72/1[-1]
```

code
13.78. \xintMin

Frac Frac
f f ★ The minimum of two fractions. Beware that \xintMin {2}{3} produces 2/1[0]. The original, for use with integers only with no need of normalization, is available as \xintiiMin: \xintiiMin {2}{3}
f f ★↪ =2.

```
  \xintMin {2.5}{7.2}
25/1[-1]
```

code
13.79. \xintMaxof

Frac
f→ * f ★ The maximum of any number of fractions, each within braces, and the whole thing within braces.

```
  \xintMaxof {{1.23}{1.2299}{1.2301}} and \xintMaxof {{-1.23}{-1.2299}{-1.2301}}
12301/1[-4] and -12299/1[-4]
```

code
13.80. \xintMinof

Frac
f→ * f ★ The minimum of any number of fractions, each within braces, and the whole thing within braces.

```
  \xintMinof {{1.23}{1.2299}{1.2301}} and \xintMinof {{-1.23}{-1.2299}{-1.2301}}
12299/1[-4] and -12301/1[-4]
```

code
13.81. \xintSum

Frac
f→ * f ★ This computes the sum of fractions. The output will now always be in the form A/B[n]. The original, for big integers only (in strict format), is available as \xintiiSum.

```
  \xintSum {{1282/2196921}{-281710/291927}{4028/28612}}
-5037928302100692/6116678670072468[0]
  No simplification attempted.
```

code
13.82. \xintPrd

Frac
f→ * f ★ TThis computes the product of fractions. The output will now always be in the form A/B[n]. The original, for big integers only (in strict format), is available as \xintiiPrd.

```
  \xintPrd {{1282/2196921}{-281710/291927}{4028/28612}}
-1454721142160/18350036010217404[0]
  No simplification attempted.
  $\xintIsOne {21921379213/21921379213}\neq
   \xintIsOne {1.000000000000000000000000000000001}$
1 ≠ 0
```

code
13.83. \xintGCD

Frac Frac
f f ★ The greatest common divisor of its two arguments, which are possibly *fractions*.

Prior to 1.4 a macro of the same name existed in xintgcd. But it truncated its two arguments to integers via \xintNum.

See \xintiiGCD for the integer only variant.

13.84. \xintLCM

code

Frac Frac
f *f* ★

The least common multiple of its two arguments, which are possibly *fractions*.

Prior to 1.4 a macro of the same name existed in xintgcd. But it truncated its two arguments to integers via \xintNum.

See \xintiiLCM for the integer only variant.

13.85. \xintGCDof

code

Frac
f→ * *f* ★

\xintGCDof{{a}{b}{c}...} computes the greatest common divisor of a, b, ... The arguments are allowed to be *fractions*: the macro produces the non-negative generator of the fractional ideal they generate. The list argument may be a macro as it is *f*-expanded first. If all arguments vanish, then also the output.

Prior to 1.4 a macro of the same name existed in xintgcd. But it truncated all its arguments to integers via \xintNum and then proceeded with integer only computations.

See \xintiiGCDof for the integer only variant.

13.86. \xintLCMof

code

Frac
f→ * *f* ★

\xintLCMof{{a}{b}{c}...} computes the least common multiple of a, b, ... The arguments are allowed to be *fractions*: the macro produces the non-negative generator of the intersection of the corresponding fractional ideals. The list argument may be a macro, it is *f*-expanded first. If one of the item vanishes, then also the output.

Prior to 1.4 a macro of the same name existed in xintgcd. But it truncated all its arguments to integers via \xintNum.

See \xintiiLCMof for the integer only variant.

13.87. \xintDigits, \xinttheDigits

code

The syntax \xintDigits := N; or (recommended) \xintDigits := N\relax assigns the value of N to the number of digits to be used by floating point operations (this uses internally a \mathchardef assignement, and N stands for (or expands to) a legal TEX number). The default is 16. The maximal value is 32767.

Accepted syntax includes also \xintDigits = N; or \xintDigits = N\relax, i.e. the colon before the equality sign is optional.

Spaces do not matter as long as they do not occur in-between digits:

```
\xintDigits := 24\relax\xinttheDigits, %
\xintDigits:=36  \relax\xinttheDigits, %
\xintDigits:= 16 \relax and \xinttheDigits.
```

24, 36, and 16.

★ As shown above \xinttheDigits expands to the stored value.

An ending active semi-colon ; is *not* compatible: it can and will cause low-level TEX errors. This is why the alternative syntax

```
\xintDigits:= N\relax
```

is recommended (with or without the semi-colon). This is hopefully the syntax now in use in most examples from the documentation.

Actually, any non-expanding token can be used in place of the \relax. This non-expanding ending token (for example a full stop) will get removed from the token stream.

```
\xintDigits  = 24\def\xinttheDigits, % only for showing it works! don't do that!
\xintDigits := 36.\xinttheDigits,    % one can use a dot in place of semi-colon
\xintDigits  = 16\relax and \xinttheDigits.\par % with \relax, even better
```

24, 36, and 16.

13.88. \xintSetDigits

code

num x

To be used as \xintSetDigits{⟨*expression*⟩} where the expression will be fed to \numexpr. It is a shortcut for doing \xintDigits := \numexpr⟨*expression*⟩\relax \relax.

```
\xintSetDigits{1+2+3+4+5}The value is now \xinttheDigits.
\xintSetDigits{2*8}The value is now \xinttheDigits.\par
```

The value is now 15. The value is now 16.

13.89. \xintFloat

code

[num x] Frac f ★

The macro \xintFloat [P]{f} has an optional argument P which replaces the current value of \xinttheDigits. The fraction f is then printed in scientific notation with a rounding to P digits.

That is, on output: the first digit is from 1 to 9, it is possibly prefixed by a minus sign and is followed by a dot and P-1 digits, then a lower case e and an exponent N. The trailing zeroes are not trimmed.

There is one exception to the general description: the zero value, which gets output as 0.0e0. This was changed at 1.4k, until then it was using 0.e0 as output. Customize via

code

\xintFloatZero whose default definition is:

```
\def\xintFloatZero{0.0e0}
```

Starting with 1.2k, when the input is a fraction AeN/BeM the output always is the *correct rounding* to P digits. Formerly, this was guaranteed only when A and B had at most P+2 digits, or when B was 1 and A was arbitrary, but in other cases it was only guaranteed that the difference between the original fraction and the rounding was at most 0.6 unit in the last place (of the output), hence the output could differ in the last digit (and earlier ones in case of chains of zeros or nines) from the correct rounding.

Also: for releases 1.2j and earlier, in the special case when A/B ended up being rounded up to the next power of ten, the output was with a mantissa of the shape 10.0...0eN. However, this worked only for B=1 or when both A and B had at most P+2 digits, because the detection of the rounding-up to next power of ten was done not on original A/B but on an approximation A'/B', and it could happen that A'/B' was itself being rounded *down* to a power of ten which however was a rounding *up* of original A/B. With the 1.2j refactoring which achieves correct rounding in all cases, it was decided not to add to the code the extra overhead of detecting with 100% fiability the rounding up to next power of ten (such overhead would necessitate alterations of the algorithm and as a result we would end up with a slightly less efficient one; it would make sense in a model where inputs have their intrinsic precisions which is obeyed by the implementation of the basic operations, but currently the design decision for the floating point macros is that when the target precision is P the inputs are rounded first to P digits before further processing.)

```
{\def\x{99999999999999994999999999999999/99999999999999999999999999999999}%
\xintFor #1 in {13, 14, 15, 16, 17, 18, 19, 47, 48, 49, 50, 79, 80, 81}
\do{#1: \xintFloat[#1]{\x}\xintifForLast{\par}{\newline}}}%
```

13: 1.000000000000e0
14: 1.0000000000000e0

```
15: 1.00000000000000e0
16: 9.999999999999999e-1
17: 9.9999999999999995e-1
18: 9.9999999999999950e-1
19: 9.9999999999999500e-1
47: 9.99999999999999950000000000000000000000000000e-1
48: 9.99999999999999949999999999999999999999999999e-1
49: 9.99999999999999949999999999999999999999999995e-1
50: 9.99999999999999949999999999999999999999999950e-1
79: 9.99999999999999949999999999999999999999999995000000000000000000000000000e-1
80: 9.99999999999999949999999999999999999999999994999999999999999999999999999e-1
81: 9.99999999999999949999999999999999999999999994999999999999999999999999995e-1
```

As an aside, which is illustrated by the above, rounding is not transitive in the number of kept digits.

```
{\def\x{13789378917328973917317/13890138013801398}%
\xintFor* #1 in {\xintSeq{4}{20}}
\do{#1: \xintFloat[#1]{\x}\newline}}%
\xintFloat{5/9999999999999999}\newline
\xintFloat[32]{5/9999999999999999}\newline
\xintFloat[48]{5/9999999999999999}\par
```

```
4: 9.927e6
5: 9.9275e6
6: 9.92746e6
7: 9.927460e6
8: 9.9274600e6
9: 9.92745997e6
10: 9.927459975e6
11: 9.9274599746e6
12: 9.92745997457e6
13: 9.927459974572e6
14: 9.9274599745717e6
15: 9.92745997457166e6
16: 9.927459974571665e6
17: 9.9274599745716647e6
18: 9.92745997457166465e6
19: 9.927459974571664655e6
20: 9.92745997457166465 45e6
5.000000000000001e-16
5.00000000000000050000000000000001e-16
5.00000000000000050000000000000005000000000000001e-16
```

code

13.90. \xintFloatBraced

[num x] Frac f ★

The experimental macro \xintFloatBraced[P]{f} does like \xintFloat but its output consists of three TEX-braced groups

{⟨*sign bit*⟩}{⟨*scientific exponent*⟩}{⟨*full width mantissa with decimal point*⟩}

It is provided for users knowing how to pick one or the other of these constituents from usage of auxiliary macros. Or one can use \xintAssign:

```
\begingroup
\xintAssign\xintFloatBraced[7]{-3.1234e-14}\to\A\B\C
\string\A\ has meaning \meaning\A\newline
\string\B\ has meaning \meaning\B\newline
```

```
  \string\C\ has meaning \meaning\C\par
  \endgroup
```
```
\A has meaning macro:->1
\B has meaning macro:->-14
\C has meaning macro:->3.123400
```
Some aspects are undecided:

- should the first item be rather -1, 0, or 1? or -, nothing, nothing?
- in case of zero value the output ignores \xintFloatZero, it uses a zero exponent and full width fractional mantissa 0.000000000000000 (here no [P] and \xinttheDigits has value 16), should it be otherwise?
- should the mantissa be without the decimal separator ? should it incorporate the sign ?
- in case the mantissa is without separator, should the exponent be biased to match it?

13.91. \xintFloatToDecimal

code

[num x] Frac f ★

\xintFloatToDecimal [P]{f} does float rounding on input like \xintFloat then outputs the number using decimal notation, i.e. with as many zeros as are needed (and no more) and no scientific exponent.

In other terms it behaves (and is essentially defined) as:

```
  \xintDecToStringREZ{\xintFloat[optional P]{<input>}}
```
Examples:
```
  \xintFloatToDecimal{6.02e23}\newline
  \xintFloatToDecimal{6.02000000000000e23}\newline
  \xintFloatToDecimal[20]{1/7e10}\newline
  \xintFloatToDecimal[30]{1/7e10}
```
```
602000000000000000000000
602000000000000000000000
0.00000000001428571428571428571
0.00000000001428571428571428571428571428571
```
See \xintDecToString.

13.92. \xintPFloat

code

[num x] Frac f ★

\xintPFloat [P]{f} is like \xintFloat but ``pretty-prints'' the output.

This macro was initially added at 1.1 as a (very primitive) "prettifying printer" for floating point number, and was then somewhat influenced by Maple, for example the zero value was printed as "0.". Then at 1.4e there was breaking change and the rules became somewhat similar to observed Python behaviour: mantissas trimmed of trailing zeros (whether or not scientific notation was used in the output) and integers printed with a trailing ".0", in particular the zero value was printed as "0.0".

1.4k brought some breaking changes, which are reversible via customizing macros:

- Integers (when scientific notation is dropped according to criteria mentioned next) without a ".0" suffix.
- Same for the zero value, now "0".
- Significands are trimmed of trailing zeros only if that removes at least 4 zeros. The rationale is that automatic removal of trailing zeros (which was influenced at 1.4e from practice with Python in interactive mode) proves annoying visually with aligned values in tables, as this creates voids, so we want to do this only when really the presence of trailing zeros is not some kind of numerical fluke.

These changes impact the \xintfloateval output as \xintfloatexprPrintOne defaults to using \xintPFloat.



In this documentation ``trailing zeros'' refers not to how the input looked like, but to the corresponding mantissa of width P or \xinttheDigits.

The default rules are thus now:

1. The input is float-rounded to either Digits or the optional argument.
2. zero is printed as 0.
3. x.yz...eN is printed in decimal fixed point if -4 ≤ N ≤ +5 else it is printed in scientific notation.
4. Trailing zeros of the mantissa are trimmed if, and only if there are at least 4 of them.
5. In case of fixed point output format, and the value is an integer, the integer is printed with no decimal mark.
6. In case of scientific notation output format, and the mantissa has only one digit, no decimal mark is used.

\xintDigits at 16

- 0 ⟶ 0
- 1.2340000e-7 ⟶ 1.234e-7
- 1.2340000e-6 ⟶ 1.234e-6
- 1.2340000e-5 ⟶ 1.234e-5
- 1.2340000e-4 ⟶ 0.0001234
- 1.2340000e-3 ⟶ 0.001234
- 1.2340000e-2 ⟶ 0.01234
- 1.2340000e-1 ⟶ 0.1234
- 1.2340000e0 ⟶ 1.234
- 1.2340000e1 ⟶ 12.34
- 1.2340000e2 ⟶ 123.4
- 1.2340000e3 ⟶ 1234
- 1.2340000e4 ⟶ 12340
- 1.2340000e5 ⟶ 123400
- 1.2340000e6 ⟶ 1.234e6
- 1.2340000e7 ⟶ 1.234e7
- 1e-7/7 ⟶ 1.428571428571429e-8
- 1e-6/7 ⟶ 1.428571428571429e-7
- 1e-5/7 ⟶ 1.428571428571429e-6
- 1e-4/7 ⟶ 1.428571428571429e-5
- 1e-3/7 ⟶ 0.0001428571428571429
- 1e-2/7 ⟶ 0.001428571428571429
- 1e-1/7 ⟶ 0.01428571428571429
- 1e0/7 ⟶ 0.1428571428571429
- 1e1/7 ⟶ 1.428571428571429
- 1e2/7 ⟶ 14.28571428571429
- 1e3/7 ⟶ 142.8571428571429
- 1e4/7 ⟶ 1428.571428571429
- 1e5/7 ⟶ 14285.71428571429
- 1e6/7 ⟶ 142857.1428571429
- 1e7/7 ⟶ 1.428571428571429e6

13.92.1. Customizing macros of \xintPFloat

code

A number of macros allow to customize the behaviour of \xintPFloat:

- \xintPFloatE allows to modify the separator of the scientific notation. Here is its default:[72]

```
\def\xintPFloatE{e}
```

- \xintPFloatZero says how to print the zero value. The default:

```
\def\xintPFloatZero{0}
```

- \xintPFloatIntSuffix is postfixed to integer values (when scientific notation is not used). Its default at 1.4k is to add nothing. It replaces the formerly hard-coded ".0" from 1.4e (prior to that trailing zeros from the full significand of P or \xinttheDigits digits were not trimmed).

[72] For TEXperts: it is allowed to define \xintPFloatE as a macro which grabs the exponent as an argument delimited by a dot, and produces *f*-expandably an output also delimited by a dot (which gets removed via further internal processing).

```
\def\xintPFloatIntSuffix{}
```

- \xintPFloatLengthOneSuffix is postfixed to trimmed mantissas having only one digit, when scientific notation is used. Its default at 1.4k is to add nothing. It replaces formerly hard-coded ".0".

```
\def\xintPFloatLengthOneSuffix{}
```

- \xintPFloatNoSciEmax is the maximal scientific exponent which will trigger use of decimal fixed point notation and \xintPFloatNoSciEmin is the minimal one. Their defaults at 1.4k are the same as the formerly hard-coded behaviour from 1.4e:

```
\def\xintPFloatNoSciEmax{5}
\def\xintPFloatNoSciEmin{-4}
```

For example (with the package default width of 16 digits for mantissas of floating point numbers):

```
\begingroup
\def\xintPFloatNoSciEmin{-20}
\xintPFloat{1e-19/7}\newline
\xintPFloat{1e-20/7}\par
\def\xintPFloatNoSciEmax{19}
\xintPFloat{1e20/7}\newline
\xintPFloat{1e21/7}\par
\endgroup
```

```
0.00000000000000000001428571428571429
1.428571428571429e-21
14285714285714290000
1.428571428571429e20
```

- \xintPFloatMinTrimmed is the minimal number of trailing zeros which have to be present to activate actual trimming. The default definition is:

```
\def\xintPFloatMinTrimmed{4}
```

Defining it to expand to -1 or 0 will enable the trimming of trailing zeros always, and setting it to a value at least equal to P (or \xinttheDigits if no [P]) will prevent it altogether.

This setting is ignored for the case of an integer value, if the criteria for using fixed point notation are met, and for the case of a one-digit mantissa in scientific notation.

To mimick approximately the Python behaviour in interactive sessions, one can use the following configuration:

```
\def\xintPFloatZero{0.0}%
\def\xintPFloatIntSuffix{.0}%
\def\xintPFloatLengthOneSuffix{.0}%
\def\xintPFloatNoSciEmax{15}%
\def\xintPFloatNoSciEmin{-4}%
\def\xintPFloatMinTrimmed{-1}%
```

```
0. ⟶ 0.0
1234. ⟶ 1234.0
6e100 ⟶ 6.0e100
1234567812345678.12345678 ⟶ 1234567812345678.0
12345678123456781.2345678 ⟶ 1.2345678123456781e16
12345678.12340 ⟶ 12345678.1234
12345678.123400 ⟶ 12345678.1234
0.12345678123456781 ⟶ 0.12345678123456781
```

```
0.00012345678123456785 → 0.0001234567812345679
0.000012345678123456785 → 1.234567812345679e-5
```

The above using the default \xintDigits setting of 16 digits. This can not naturally match exactly CPython which uses internally radix 2 not 10, and has (by default) mantissas with 53=1+52 bits.

Same, but playing with xintsession in its &fp mode:

```
>>> &fp
fp mode (16 digits)
>>> \\def\xintPFloatZero{0.0}
(executing \\def\xintPFloatZero {0.0} in background)
)
Runaway argument?
def\xintPFloatZero {0.0}\message {
}\xs_fetch_aa \endinput
! File ended while scanning use of \\.
<inserted text>
                \par
<*> xintsession^^M

? S
OK, entering \scrollmode...

*\xintsession

  You are back to the xintexpr interactive session!
  (current mode: fp (Digits=16), with Digits=16)

  ">>> " means central computing is waiting for input
  "... " means that multi-line input continues.  Use `;' to terminate it.

  Say `&bye' at any time to terminate the session and the TeX run.
>>> \def\xintPFloatZero{0.0}
(executing \def \xintPFloatZero {0.0} in background)

>>> \def\xintPFloatIntSuffix{.0}\def\xintPFloatLengthOneSuffix{.0}

(executing \def \xintPFloatIntSuffix {.0}\def \xintPFloatLengthOneSuffix {.0} i
n background)

>>> \def\xintPFloatNoSciEmax{15}\def\xintPFloatNoSciEmin{-4}

(executing \def \xintPFloatNoSciEmax {15}\def \xintPFloatNoSciEmin {-4} in back
ground)

>>> \def\xintPFloatMinTrimmed{-1}
(executing \def \xintPFloatMinTrimmed {-1} in background)

>>> 0., 1234., 6e100;
@_1     0.0, 1234.0, 6.0e100
>>> 1234567812345678.12345678;
@_2     1234567812345678.0
>>> 12345678123456781.2345678;
```

```
@_3     1.234567812345678e16
>>> 12345678.12340;
@_4     12345678.1234
>>> 12345678.123400;
@_5     12345678.1234
>>> 0.1234567812345678;
@_6     0.1234567812345678
>>> 0.00012345678123456785;
@_7     0.0001234567812345679
>>> 0.000012345678123456785;
@_8     1.234567812345679e-5
>>> &bye
```

This is with version 0.4alpha (2021-11-01) of xintsession. Probably some ``magic'' shortcuts will be added in future to its interface for this kind of tasks, in place of the \def.

code

13.93. \xintFloatAdd

num Frac Frac
[x] f f ★

\xintFloatAdd [P]{f}{g} first replaces f and g with their float approximations f' and g' to P significant places or to the precision from \xintDigits. It then produces the sum f'+g', correctly rounded to nearest with the same number of significant places.

code

13.94. \xintFloatSub

num Frac Frac
[x] f f ★

\xintFloatSub [P]{f}{g} first replaces f and g with their float approximations f' and g' to P significant places or to the precision from \xintDigits. It then produces the difference f'-g' correctly rounded to nearest P-float.

code

13.95. \xintFloatMul

num Frac Frac
[x] f f ★

\xintFloatMul [P]{f}{g} first replaces f and g with their float approximations f' and g' to P (or \xinttheDigits) significant places. It then correctly rounds the product f'*g' to nearest P-float.

See subsection 9.2 for more.

NEW WITH 2.0a1

It is high time the author improved this algorithm to avoid computing an exact product of up to 2P digits, only to discard roughly half of them when rounding to P digits!

This issue has been partly addressed for multiplication in \xintfloateval (aka \calxeval), which uses specialized truncated-product algorithms at certain precisions. For larger precisions, however, the full product is still computed before rounding. This will need some improvements in the future.

code

13.96. \xintFloatDiv

num Frac Frac
[x] f f ★

\xintFloatDiv [P]{f}{g} first replaces f and g with their float approximations f' and g' to P (or \xinttheDigits) significant places. It then correctly rounds the fraction f'/g' to nearest P-float.

See subsection 9.2 for more.

Notice in the special situation with f and g integers that \xintFloatDiv [P]{f}{g} will *not necessarily* give the correct rounding of the exact fraction f/g. Indeed the macro arguments are

each first individually rounded to P digits of precision. The correct syntax to get the correctly rounded integer fraction f/g is \xintFloat[P]{f/g}.

code
13.97. \xintFloatPow

[num x] Frac f num x ★

\xintFloatPow [P]{f}{x} uses either the optional argument P or in its absence the value of \xinttheDigits. It computes a floating approximation to f^x.

The exponent x will be handed over to a \numexpr, hence count registers are accepted on input for this x. And the absolute value |x| must obey the TEX bound.

The argument f is first rounded to P significant figures to give f'. The output Z is such that the exact f'^x differs from Z by an absolute error less than 0.52 ulp(Z).

```
\xintFloatPow {3.1415}{1234567890}
1.612206591961208e613749456
```

code
13.98. \xintFloatPower

[num x] Frac f Num f ★

\xintFloatPower[P]{f}{g} computes a floating point value f^g where the exponent g is not constrained to be at most the TEX bound 2147483647. It may even be a fraction A/B but will be truncated to an integer. The exponent of the *output* however *must* at any rate obey the TEX bound.

The argument f is first rounded to P significant figures to give f'. The output Z is such that the exact f'^g differs from Z by an absolute error less than 0.52 ulp(Z).

Before 2.0 this macro was used in \xintfloateval for f**g (or pow()) when g was recognized as an integer. At 2.0a2 a native calx routine handles such exponents within the \numexpr range, at the arithmetic precision (independently of the logarithmic library precision). As with this macro, the base is first rounded to that precision. Other exponents, and exceptionally powers near the exponent range limits, use the logarithmic route.

CHANGED AT 2.0a2!

The internal implementation remains in use by \xinteval for *integer* exponents when an exact computation would produce more than about 10000 digits in the numerator or denominator, or when the absolute value of the exponent is at least 10000. See \xintPow. It is explained there that the recognition of an integer exponent by \xinteval is based on its input format, not on its mathematical value.

The integer exponent g may have more than P (or \xinttheDigits) digits, it is handled exactly. And as said above its absolute value may exceed the TEX bound.

Compare the following example (where the exponent is larger than the TEX bound)

```
\xintFloatPower {3.1415}{3703703670}
4.190463649314189e1841248368
```

with Python's built-in floating-point arithmetic:

```
>>> 3.1415**3703703670
Traceback (most recent call last):
  File "<python-input-3>", line 1, in <module>
    3.1415**3703703670
    ~~~~~~^^~~~~~~~~~~
OverflowError: (34, 'Result too large')
```

code
13.99. \xintFloatSqrt

[num x] Frac f ★

\xintFloatSqrt[P]{f} computes a floating point approximation of $\sqrt{f}$, either using the optional precision P or the value of \xinttheDigits.

More precisely since 1.2f the macro achieves so-called *correct rounding*: the produced value is the rounding to P significant places of the abstract exact value, *if the input has itself at most P digits* (and an arbitrary exponent).

```
  \xintFloatSqrt [89]{10}\newline
  \xintFloatSqrt [89]{100}\newline
  \xintFloatSqrt [89]{123456789}\par
3.16227766016837933199889354443271853371955513932521682685750485279259443863923822134424481e0
1.00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000e1
1.11111110605555554405416661433534692458784098601343510714585706752514714794963667365791364e4
  And now some tests to check that correct rounding applies correctly (sic):
  The argument has 16 digits, hence escapes initial rounding:\newline
  \xintFloatSqrt {5625000075000001}\newline
  This one gets rounded hence same value is computed:\newline
  \xintFloatSqrt {5625000075000001.4}\newline
  but actual value is more like:\newline
  \xintFloatSqrt [24]{5625000075000001.4}\newline
  \xintFloatSqrt [32]{5625000075000001.4}\newline
  The argument has 48 digits, hence escapes initial rounding:\newline
  \xintFloatSqrt [48]{562500000000000000000007500000000000000000000001}\newline
  \xintFloatSqrt [64]{562500000000000000000007500000000000000000000001}\newline
  \xintFloatSqrt [80]{562500000000000000000007500000000000000000000001}\newline
The argument has 16 digits, hence escapes initial rounding:
7.500000050000000e7
This one gets rounded hence same value is computed:
7.500000050000000e7
but actual value is more like:
7.500000050000000766666666e7
7.500000050000000766666661555555 6e7
The argument has 48 digits, hence escapes initial rounding:
7.500000000000000000000005000000000000000000000000e23
7.500000000000000000000005000000000000000000000005000000000000000e23
7.500000000000000000000005000000000000000000000049999999999999999999999966666667e23
 (we observe in passing illustrations that rounding to nearest is not transitive.)
```

code

13.100. \xintFloatFac

[num x] Num f ★

\xintFloatFac[P]{f} returns the factorial with either \xinttheDigits or P digits of precision.

The exact theoretical value differs from the calculated one Y by an absolute error strictly less than 0.6 ulp(Y).

```
  $1000!\approx{}$\xintFloatFac [30]{1000}
```

1000! ≈ 4.02387260077093773543702433923e2567 The computation proceeds via doing explicitely the product.[73]

The maximal allowed argument is 99999999, but already 100000! currently takes, for 16 digits of precision, a few seconds on my laptop (it returns 2.824229407960348e456573).

The factorial function is available in \xintfloatexpr:

```
  \xintthefloatexpr factorial(1000)\relax % same as 1000!
4.023872600770938e2567
```

code

13.101. \xintFloatBinomial

[num x] Num Num f f ★

\xintFloatBinomial[P]{x}{y} computes binomial coefficients with either \xinttheDigits or P digits of precision.

[73] An approach based upon the Stirling formula could not be done at time of implementation because of lack of exponential and logarithm. This is now supported. So perhaps at some point in future Gamma function will be implemented.

When x<0 an out-of-range error is raised. Else if y<0 or if x<y the macro evaluates to 0.0e0. The exact theoretical value differs from the calculated one Y by an absolute error strictly less than 0.6 ulp(Y).

```
${3000\choose 1500}\approx{}$\xintFloatBinomial [24]{3000}{1500}
```

$\binom{3000}{1500} \approx 1.79196793754756005073269e901$

The associated function in \xintfloatexpr is binomial():

```
\xintthefloatexpr binomial(3000,1500)\relax
```

1.791967937547560e901

The computation is based on the formula (x-y+1)...x/y! (here one arranges y<=x-y naturally).

13.102. \xintFloatPFactorial

code

[num x] Num f Num f ★

\xintFloatPFactorial[P]{x}{y} computes the product (x+1)...y.

The arguments must be integers (they are expanded inside \numexpr) and the allowed range is $-100000000 \leqslant x, y \leqslant 99999999$. If $x \geqslant y$ the product is considered empty hence returns one (as a floating point value). See also \xintiiPFactorial.

The exact theoretical value differs from the calculated one Y by an absolute error strictly less than 0.6 ulp(Y).

The associated function in \xintfloatexpr is pfactorial():

```
\xintthefloatexpr pfactorial(2500,5000)\relax
```

2.595989017947957e8914

14. Macros of the xintbinhex package

.1 Introduction 202
.2 xintexpr-essions 202
.3 \xintHexToDec 203
.4 \xintHexToOct 203
.5 \xintHexToBin 203
.6 \xintCHexToBin 203
.7 \xintDecToHex 204
.8 \xintDecToOct 204
.9 \xintDecToBin 204
.10 \xintOctToHex 205
.11 \xintOctToDec 205
.12 \xintOctToBin 205
.13 \xintCOctToBin 205
.14 \xintBinToHex 205
.15 \xintBinToDec 206
.16 \xintBinToOct 206
.17 \xintLowerHex 206
.18 \xintUpperHex 206
.19 Maximal sizes of inputs 207

14.1. Introduction

This package provides expandable conversions of (big) integers between the hexadecimal, decimal, octal (since 1.4n) and binary bases.

First version of this package was in the 1.08 (2013/06/07) release of xint. Its routines remained unmodified until their complete rewrite at release 1.2m (2017/07/31). Macros became faster, but the inputs got limited to a few thousand digits, whereas the 1.08 versions could handle (slowly...) tens of thousands of digits. At 1.4n some internals were refactored to use the

NEW WITH 1.4n

\expanded primitive (which was not available in 2017). The maximal sizes got increased, see subsection 14.19. More significant probably, the octal radix was added to the ones covered.

NEW WITH 1.4p

1.4p added \xintLowerHex and \xintUpperHex.

For each provided conversion macro, its argument is first *f*-expanded. This expansion is supposed to give a sequence of digits, with perhaps a (unique) leading minus sign, which gets prepended to output (note that \xintDecToHex{-0} thus expands to -0).

Let's insist that inputs can not start (after expansion) with a 0b, 0o, 0x, #x, ", ', or similar prefix notation: they must consist only of digits as fitting to the binary, octal, decimal, or hexadecimal radix. Situation is different if using xintexpr-essions, see subsection 14.2 next.

Low-level unrecoverable errors will occur if for example an octal input contains the decimal digit 8.

Hexadecimal digits in input must be uppercased. Category codes for them may be indifferently *letter* or *other*. In output they are of category *letter* (and uppercased).

Leading zeroes in the input are allowed, and depending on the macro may show up or not in the output. Note in particular:

- Inputs with no leading zeros give outputs with no leading zeros.
- All rules have (deliberate) exceptions, check the docs of \xintCHexToBin and \xintCOctToBin which are variants of \xintHexToBin and \xintOctToBin.
- Outputs (if non vanishing) from \xintDecToHex or \xintDecToOct have no leading zeros whether or not the inputs had some.
- \xintBinToHex and \xintBinToOct always use the minimal number of hexadecimal resp. octal digits as needed to represent the original binary digits, inclusive of their leading zeroes. For example \xintBinToHex{0000001} outputs 01 and \xintBinToOct{0000001} outputs 001.

14.2. xintexpr-essions

NEW WITH 1.4n

Inside xintexpr-essions, hexadecimal can be input using either " or 0x prefixes, octal using either ' or 0o, and binary using 0b. Prior to 1.4n only " was implemented and it was needed to load xintbinhex additionally to xintexpr. This is now done automatically.

Hexadecimal letters must be uppercased. In both of \xinteval and \xintfloateval a ``fractional part'' after the full stop as separator is allowed for all three bases. Here is an example using the three non-decimal bases:

```
\xinteval{subsm({x, y, x==y}, x="FF.FF * '777.777 * 0b11111.11111;
                                y=(16^2-16^-2)(8^3-8^-3)(2^5-2^-5);)}
```

```
4.19012807836890220642089843750e6, 17574670959615/4194304, 1
```

The p postfix notation from some programming languages, which stands for an extra power of two, is however not implemented so far.

NEW WITH 1.4n

With \xintiieval, which handles only integers, there is an optional parameter [h], [o], or [b] for automatic conversion of the output (this works also with comma separated inputs and even nested bracketed inputs). Here is an example illustrating the new more condensed syntax:

```
\xintiieval[b]{0b1011100101001110 * 0b111100011}
```

```
10101110110011110001010101
```

Compare with how one would have had to input it prior to 1.4n:

```
\xintDecToBin{\xintiieval{\xintBinToDec{1011100101001110}*\xintBinToDec{111100011}}}
```

```
10101110110011110001010101
```

Note that if while parsing an octal number the digits 8 or 9 is encountered, this does not trigger un error but terminates the octal input fetched so far and inserts a tacit multiplication. Same for binary input if encountering a digit other than 0 or 1.

14.3. \xintHexToDec

code

f ★ Converts from hexadecimal to decimal.

```
\xintHexToDec{11A9397C66949A97051F7D0A817914E3E0B17C41B11C48BAEF2B5760BB38D272F46DCE46C6032936BF37DAC918814C63}
->27182818284590452353602874713526624977572470936999595749669676277240766303535475945713821785251664274274663919320003
```

14.4. \xintHexToOct

code

f ★ Converts from hexadecimal to octal.

NEW WITH 1.4n

```
\xintHexToOct{11A9397C66949A97051F7D0A817914E3E0B17C41B11C48BAEF2B5760BB38D272F46DCE46C6032936BF37DAC918814C63}
->432447137063224465134050767641240274424707602613704066107044272736255273013547064471364333471066140145115537467665444304024614 3
```

14.5. \xintHexToBin

code

f ★ Converts from hexadecimal to binary. Up to three leading zeroes of the binary output are trimmed.

```
\xintHexToBin{11A9397C66949A97051F7D0A817914E3E0B17C41B11C48BAEF2B5760BB38D272F46DCE46C6032936BF37DAC918814C63}
->1000110101001001110010111110001100110100101001001101010010111000001010001111101111101000010101000000101111001000101001110001111100000101100010111110001000001101100010001110001001000101011100101000111011010111011110010101101010101110110000010111011001110001101001001110010111101000110110111001100010001101100000001100101001001101101011111100110111111011010110010010001100010000001001100011
```

14.6. \xintCHexToBin

code

f ★ Converts from hexadecimal to binary. Same as \xintHexToBin, but an input with N hexadecimal digits will give an output with exactly 4N binary digits, leading zeroes are not trimmed.

```
\xintCHexToBin{11A9397C66949A97051F7D0A817914E3E0B17C41B11C48BAEF2B5760BB38D272F46DCE46C60
↪ 32936BF37DAC918814C63}
->0001000110101001001110010111110001100110100101001001101010010111000001010001111101111101000
↪ 0010101000000101111001000101001110001111100000101100010111110001000001101100010001110001001
↪ 0010111010111011110010101101010111011000001011101100111000110100100111001011110100011011011
↪ 0011100100011011000110000000110010100100110110101111110011011111011010110010010001100010000
↪ 010100110001100011
```

This can be combined with `\xintBinToHex` for round-trips preserving leading zeroes for `4N` binary digits numbers, whereas using `\xintHexToBin` gives reproducing round-trips only for `4N` binary numbers numbers not starting with `0000`.

```
\xintBinToHex{0001111}\par
0F
```

Chaining, we end up with 4N-3 digits, as three binary zeroes are trimmed:

```
\xintHexToBin{\xintBinToHex{0001111}}\par
01111
```

But the next will always reproduce the initial input zero-filled to length 4N:

```
\xintCHexToBin{\xintBinToHex{0001111}}\par
00001111
```

Another example (visible space characters manually inserted):

$$0000000011111101001010001 \xrightarrow{\texttt{\textbackslash xintBinToHex}} 00FA51 \xrightarrow{\texttt{\textbackslash xintHexToBin}} ␣␣␣0000111111010010100001$$
$$0000000011111101001010001 \xrightarrow{\texttt{\textbackslash xintBinToHex}} 00FA51 \xrightarrow{\texttt{\textbackslash xintCHexToBin}} 0000000011111101001010001$$

code

14.7. \xintDecToHex

f ★ Converts from decimal to hexadecimal.

```
\xintDecToHex{271828182845904523536028747135266249775724709369995957496696762772407663035
↪ 5475945713821785251664274274663919320003}
->11A9397C66949A97051F7D0A817914E3E0B17C41B11C48BAEF2B5760BB38D272F46DCE46C6032936BF37DAC918
↪ 814C63
```

code

14.8. \xintDecToOct

f ★ Converts from decimal to octal.

NEW WITH 1.4n

```
\xintDecToOct{271828182845904523536028747135266249775724709369995957496696762772407663035
↪ 5475945713821785251664274274663919320003}
->43244713706322446513405076764124027442470760261370406610704427273625527301354706447136433
↪ 4710661401451155374676654443040246143
```

code

14.9. \xintDecToBin

f ★ Converts from decimal to binary.

```
\xintDecToBin{271828182845904523536028747135266249775724709369995957496696762772407663035
↪ 5475945713821785251664274274663919320003}
->1000110101001001110010111110001100110100101001001101010010111000001010001111101111101000
↪ 0101000000101111001000101001110001111100000101100010111110001000001101100010001110001001
↪ 0111010111011110010101101010111011000001011101100111000110100100111001011110100011011011
↪ 1100100011011000110000000110010100100110110101111110011011111011010110010010001100010000
↪ 100110001100011
```

code
14.10. \xintOctToHex

f ⋆ Converts from octal to hexadecimal.

NEW WITH 1.4n
```
\xintOctToHex{43244713706322446513405076764124027442470760261370406610704427273625527301354706447136433347106614014511553746766544430402461435}
->11A9397C66949A97051F7D0A817914E3E0B17C41B11C48BAEF2B5760BB38D272F46DCE46C6032936BF37DAC918814C63
```

code
14.11. \xintOctToDec

f ⋆ Converts from octal to decimal.

NEW WITH 1.4n
```
\xintOctToDec{43244713706322446513405076764124027442470760261370406610704427273625527301354706447136433347106614014511553746766544430402461435}
->27182818284590452353602874713526624977572470936999595749669676277240766303535475945713821785251664274274663919320035}
```

code
14.12. \xintOctToBin

f ⋆ Converts from octal to binary. Up to two leading zeroes of the binary output are trimmed.

NEW WITH 1.4n
```
\xintOctToBin{14324471370632244651340507676412402744247076026137040661070442727362552730135470644713643334710661401451155374676654443040246143}
->1100011010100100111001011111000110011010010100100110101001011100000101000111110111110100001010100000010111100100010100111000111110000010110001011111000100000110110001000111000100100010100001011110010001010011100011111000001011000101111100010000011011000100011100010010001010111010111011110010101101010111011000001011101100111000110100100111001011111010001101101110011000111000000011001010010011011010111111001101111101101011001001000110001000001100011001
```

```
->11A9397C66949A97051F7D0A817914E3E0B17C41B11C48BAEF2B5760BB38D272F46DCE46C6032936BF37DAC918
↪814C63
```

code

14.15. \xintBinToDec

f ★ Converts from binary to decimal.

```
  \xintBinToDec{1000110101001001110010111110001100110100101001001101010010111000001010001111
↪1011111010000101010000001011110010001010011100011111000001011000101111100010000011011000100
↪1110001001000101110101110111110010101101010111011000001011101100111000110100100111001011110
↪0011011011100111001000110110001100000000110010100100110110101111110011011111011010110010010
↪1100010000001010011000110001100011}
->27182818284590452353602874713526624977572470936999595749669676277240766303535475945713821
↪8525166427427466391932003
```

code

14.16. \xintBinToOct

f ★ Converts from binary to octal.

The input is first extended if need-be by leading zeros in order to have 3N binary digits, then the output will have N octal digits (thus, if the input did not have a leading zero, the output will not either).

NEW WITH 1.4n

```
  \xintBinToOct{1000110101001001110010111110001100110100101001001101010010111000001010001111
↪1011111010000101010000001011110010001010011100011111000001011000101111100010000011011000100
↪1110001001000101110101110111110010101101010111011000001011101100111000110100100111001011110
↪0011011011100111001000110110001100000000110010100100110110101111110011011111011010110010010
↪1100010000001010011000110001100011}
->4324471370632244651340507676412402744247076026137040661070442727362552730135470644713643333
↪4710661401451155374676654443040246143
```

code

14.17. \xintLowerHex

f ★ NEW WITH 1.4p

The output is the lowercase version of the input (which can be with mixed case). An empty input is allowed. Letters on input can be of catcode *letter* or *other*. Letters in output are of catcode *letter*.

```
  \xintLowerHex{11A9397C66949A97051F7D0A817914E3E0B17C41B11C48BAEF2B5760BB38D272F}\newline
  \xintLowerHex{\xintDecToHex{12379813738877118345}}
11a9397c66949a97051f7d0a817914e3e0b17c41b11c48baef2b5760bb38d272f
abcdef0123456789
```

As it *f*-expands its argument, you can as shown above use it to wrap usage of one of the package macros converting to hexadecimal.

code

14.18. \xintUpperHex

f ★ NEW WITH 1.4p

The output is the uppercase version of the input (which can be with mixed case). An empty input is allowed. Letters on input can be of catcode *letter* or *other*. Letters in output are of catcode *letter*.

```
  \xintUpperHex{11a9397c66949a97051f7d0a817914e3e0b17c41b11c48baef2b5760bb38d272f}\newline
  \xintHexToDec{\xintUpperHex{abcdef0123456789}}
11A9397C66949A97051F7D0A817914E3E0B17C41B11C48BAEF2B5760BB38D272F
12379813738877118345
```

As shown above, use it to wrap the argument of \xintHexToDec if for example some lowercase, or mixed case, hexadecimal input needs to get converted to decimal.

14.19. Maximal sizes of inputs

Table 5 recapitulates the maximal allowed sizes, as found out with the TEX installation of the author. The tests are done putting the macro inside an \edef and compiling with the etex binary.

The value in the second column is the maximal N such that the macro does not raise an error on an input with N digits (if nested in another macro, the maximal input size may become lower than stated). The third column gives the corresponding maximal size of the output.

These maximal sizes depend on TEX parameters such as input stack size, expansion depth, and parameter stack size. The fourth column gives the TEX parameter cited in the error message when trying with N+1 digits. When the limiting parameter is the main memory size, the upper limits depend on external factors such as how many macros are loaded in TEX memory (for example they would be lower in a LATEX document or if xintexpr is loaded), so they are here given simply as indications. They are so large anyhow that basically in practice, this means no real limitation.

Regarding the conversions to and from decimal radix, they allow a more limited range, but are still able to handle inputs with ten thousand digits, one can thus consider in practice that their size limitations are also only of theoretical interest.

	Max length of input	-> length of output	Limiting factor
\xintDecToHex	16042	13323	expansion depth=10000
\xintDecToOct	15040	16654	expansion depth=10000
\xintDecToBin	16042	53291	expansion depth=10000
\xintHexToDec	11072	13333	expansion depth=10000
\xintOctToDec	14763	13333	expansion depth=10000
\xintBinToDec	44290	13333	expansion depth=10000
\xintHexToOct	553514	738019	main memory size=5000000
\xintHexToBin	553514	2214056	main memory size=5000000
\xintOctToHex	711660	533745	main memory size=5000000
\xintOctToBin	711660	2134980	main memory size=5000000
\xintBinToHex	1992650	498163	main memory size=5000000
\xintBinToOct	1868109	622703	main memory size=5000000

Table 5: Maximal sizes for xintbinhex 1.4n macros with TeXLive 2025

15. Macros of the xintgcd package

.1 Introduction 208
.2 \xintBezout 208
.3 \xintEuclideAlgorithm 208
.4 \xintBezoutAlgorithm 209
.5 \xintTypesetEuclideAlgorithm 209
.6 \xintTypesetBezoutAlgorithm 209

15.1. Introduction

This package was included in the original release 1.0 (2013/03/28) of the xint bundle.

At 1.3d macros \xintiiGCD and \xintiiLCM are copied over to xint, hence gcd() and lcm() functions in \xintiiexpr were available simply from loading only xintexpr, and the xintgcd dependency got removed.



From 1.1 to 1.3f the package loaded only xintcore, not xint and neither xinttools.

But at 1.4 it loads automatically both xint and xinttools (the latter being a requirement since 1.09h of the \xintTypesetEuclideAlgorithm and \xintTypesetBezoutAlgorithm macros).

The macros \xintiiGCD and \xintiiLCM got relocated into xint. The macros \xintGCD, \xintLCM, \xintGCDof, and \xintLCMof are *removed*: xintfrac provides under these names more powerful macros handling general fractions and not only integers.

15.2. \xintBezout

code

Num Num f f ★ \xintBezout{N}{M} returns three numbers U, V, D within braces where D is the (non-negative) GCD, and UN + VM = D.

```
  \oodef\X{\xintBezout {10000}{1113}}\meaning\X\par
  \xintAssign {\xintBezout {10000}{1113}}\to\U\V\D
  U: \meaning\U, V: \meaning\V, D: \meaning\D\par
  AU+BV: \xinttheiiexpr 10000*\U+1113*\V\relax\par
  \noindent\oodef\X{\xintBezout {123456789012345}{9876543210321}}\meaning\X\par
  \xintAssign \X\to\U\V\D
  U: \meaning\U, V: \meaning\V, D: \meaning\D\par
  AU+BV: \xinttheiiexpr 123456789012345*\U+9876543210321*\V\relax
macro:->{-131}{1177}{1}
  U: macro:->-131, V: macro:->1177, D: macro:->1
  AU+BV: 1
macro:->{256654313730}{-3208178892607}{3}
  U: macro:->256654313730, V: macro:->-3208178892607, D: macro:->3
  AU+BV: 3
```

15.3. \xintEuclideAlgorithm

code

Num Num f f ★ \xintEuclideAlgorithm{N}{M} applies the Euclide algorithm and keeps a copy of all quotients and remainders.

```
  \edef\X{\xintEuclideAlgorithm {10000}{1113}}\meaning\X
macro:->{5}{10000}{1}{1113}{8}{1096}{1}{17}{64}{8}{2}{1}{8}{0}
```

The first item is the number of steps, the second is N, the third is the GCD, the fourth is M then the first quotient and remainder, the second quotient and remainder, ... until the final quotient and last (zero) remainder.

15.4. \xintBezoutAlgorithm

code

Num Num f f ⋆ \xintBezoutAlgorithm{N}{M} applies the Euclide algorithm and keeps a copy of all quotients and remainders. Furthermore it computes the entries of the successive products of the 2 by 2 matrices $\begin{pmatrix} q & 1 \\ 1 & 0 \end{pmatrix}$ formed from the quotients arising in the algorithm.

```
\edef\X{\xintBezoutAlgorithm {10000}{1113}}\printnumber{\meaning\X}
macro:->{5}{10000}{0}{1}{1}{1113}{1}{0}{8}{1096}{8}{1}{1}{17}{9}{1}{64}{8}{584}{65}{2}{1}{11
↪ 77}{131}{8}{0}{10000}{1113}
```

The first item is the number of steps, the second is N, then 0, 1, the GCD, M, 1, 0, the first quotient, the first remainder, the top left entry of the first matrix, the bottom left entry, and then these four things at each step until the end.

15.5. \xintTypesetEuclideAlgorithm

code

Num Num f f This macro is just an example of how to organize the data returned by \xintEuclideAlgorithm. Copy the source code to a new macro and modify it to what is needed.

```
\xintTypesetEuclideAlgorithm {123456789012345}{9876543210321}
```

$$
\begin{aligned}
123456789012345 &= 12 \times 9876543210321 + 4938270488493\\
9876543210321 &= 2 \times 4938270488493 + 2233335\\
4938270488493 &= 2211164 \times 2233335 + 536553\\
2233335 &= 4 \times 536553 + 87123\\
536553 &= 6 \times 87123 + 13815\\
87123 &= 6 \times 13815 + 4233\\
13815 &= 3 \times 4233 + 1116\\
4233 &= 3 \times 1116 + 885\\
1116 &= 1 \times 885 + 231\\
885 &= 3 \times 231 + 192\\
231 &= 1 \times 192 + 39\\
192 &= 4 \times 39 + 36\\
39 &= 1 \times 36 + 3\\
36 &= 12 \times 3 + 0
\end{aligned}
$$

15.6. \xintTypesetBezoutAlgorithm

code

Num Num f f This macro is just an example of how to organize the data returned by \xintBezoutAlgorithm. Copy the source code to a new macro and modify it to what is needed.

```
\xintTypesetBezoutAlgorithm {10000}{1113}
```

$$
\begin{aligned}
10000 &= 8 \times 1113 + 1096\\
8 &= 8 \times 1 + 0\\
1 &= 8 \times 0 + 1\\
1113 &= 1 \times 1096 + 17\\
9 &= 1 \times 8 + 1\\
1 &= 1 \times 1 + 0\\
1096 &= 64 \times 17 + 8\\
584 &= 64 \times 9 + 8\\
65 &= 64 \times 1 + 1\\
17 &= 2 \times 8 + 1\\
1177 &= 2 \times 584 + 9\\
131 &= 2 \times 65 + 1
\end{aligned}
$$

```
    8 = 8 × 1 + 0
10000 = 8 × 1177 + 584
 1113 = 8 × 131 + 65
  131 × 10000 - 1177 × 1113 = -1
```

16. Macros of the xintseries package

.1 Introduction 211
.2 \xintSeries 211
.3 \xintiSeries 212
.4 \xintRationalSeries 213
.5 \xintRationalSeriesX 216
.6 \xintPowerSeries 218
.7 \xintPowerSeriesX 219
.8 \xintFxPtPowerSeries 220
.9 \xintFxPtPowerSeriesX 221
.10 \xintFloatPowerSeries 222
.11 \xintFloatPowerSeriesX 222
.12 Computing log 2 and π 222

16.1. Introduction

This package was first released with version 1.03 (2013/04/14) of the xint bundle.

The $^{\text{Frac}}f$ expansion type of various macro arguments is only a $^{\text{Num}}f$ if only xint but not xintfrac is loaded. The macro \xintiSeries is special and expects summing big integers obeying the strict format, even if xintfrac is loaded.

The arguments serving as indices are of the $^{\text{num}}x$ expansion type.

In some cases one or two of the macro arguments are only expanded at a later stage not immediately.

> Since 1.3, \xintAdd and \xintSub use systematically the least common multiple of the denominators. Some of the comments in this chapter refer to the earlier situation where often the denominators were simply multiplied together. *They have yet to be updated to reflect the new situation brought by the 1.3 release.* Some of these comments may now be off-synced from the actual computation results and thus may be wrong.

16.2. \xintSeries

code

$^{\text{num}}x\,{}^{\text{num}}x\,{}^{\text{Frac}}f$ ★ \xintSeries{A}{B}{\coeff} computes $\sum_{n=A}^{n=B}$\coeff{n}. The initial and final indices must obey the \numexpr constraint of expanding to numbers at most 2^31-1. The \coeff macro must be a one-parameter *f*-expandable macro, taking on input an explicit number n and producing some number or fraction \coeff{n}; it is expanded at the time it is needed.

```
\def\coeff #1{\xintiiMON{#1}/#1.5} % (-1)^n/(n+1/2)
\fdef\w {\xintSeries {0}{50}{\coeff}} % we want to re-use it
\fdef\z {\xintJrr {\w}[0]} % the [0] for a microsecond gain.
% \xintJrr preferred to \xintIrr: a big common factor is suspected.
% But numbers much bigger would be needed to show the greater efficiency.
\[ \sum_{n=0}^{n=50} \frac{(-1)^n}{n+\frac12} = \xintTeXFrac\z \]
```

$$\sum_{n=0}^{n=50} \frac{(-1)^n}{n+\frac{1}{2}} = \frac{1739093382873709404321127921016266022787 14}{1100274671593900030252799172260397290505 75}$$

The definition of \coeff as \xintiiMON{#1}/#1.5 is quite suboptimal. It allows #1 to be a big integer, but anyhow only small integers are accepted as initial and final indices (they are of the $^{\text{num}}x$ type). Second, when the xintfrac parser sees the #1.5 it will remove the dot hence create a denominator with one digit more. For example 1/3.5 turns internally into 10/35 whereas it would be more efficient to have 2/7. For info here is the non-reduced \w:

$$\frac{8695466914368547021605639605081330113935 7}{5501373357969500151263995861301986452528 75}10^{1}$$

It would have been bigger still in releases earlier than 1.1: now, the xintfrac \xintAdd routine does not multiply blindly denominators anymore, it checks if one is a multiple of the other. However it does not practice systematic reduction to lowest terms.

A more efficient way to code \coeff is illustrated next.

```
\def\coeff #1{\the\numexpr\ifodd #1 -2\else2\fi\relax/\the\numexpr 2*#1+1\relax [0]}%
% The [0] in \coeff is a tiny optimization: in its presence the \xintfracname parser
% sees something which is already in internal format.
\fdef\w {\xintSeries {0}{50}{\coeff}}
\[\sum_{n=0}^{n=50} \frac{(-1)^n}{n+\frac12}=\xintTeXFrac\w\]
```

$$\sum_{n=0}^{n=50} \frac{(-1)^n}{n+\frac{1}{2}} = \frac{1739093382873709404321127921016266022787 14}{1100274671593900030252799172260397290505 75}$$

The reduced form \z as displayed above only differs from this one by a factor of 1.

```
\def\coeffleibnitz #1{\the\numexpr\ifodd #1 1\else-1\fi\relax/#1[0]}
\cnta 1
\loop
% in this loop we recompute from scratch each partial sum!
% we can afford that, as \xintSeries is fast enough.
\noindent\hbox to 2em{\hfil\texttt{\the\cnta.} }%
         \xintTrunc {12}{\xintSeries {1}{\cnta}{\coeffleibnitz}}\dots
\endgraf
\ifnum\cnta < 30 \advance\cnta 1 \repeat
```

```
 1. 1.000000000000...        11. 0.736544011544...        21. 0.716390450794...
 2. 0.500000000000...        12. 0.653210678210...        22. 0.670935905339...
 3. 0.833333333333...        13. 0.730133755133...        23. 0.714414166209...
 4. 0.583333333333...        14. 0.658705183705...        24. 0.672747499542...
 5. 0.783333333333...        15. 0.725371850371...        25. 0.712747499542...
 6. 0.616666666666...        16. 0.662871850371...        26. 0.674285961081...
 7. 0.759523809523...        17. 0.721695379783...        27. 0.711322998118...
 8. 0.634523809523...        18. 0.666139824228...        28. 0.675608712404...
 9. 0.745634920634...        19. 0.718771403175...        29. 0.710091471024...
10. 0.645634920634...        20. 0.668771403175...        30. 0.676758137691...
```

16.3. \xintiSeries

code

num num f ★

\xintiSeries{A}{B}{\coeff} computes $\sum_{n=A}^{n=B}$\coeff{n} where \coeff{n} must *f*-expand to a (possibly long) integer in the strict format.

```
\def\coeff #1{\xintiTrunc {40}{\xintiiMON{#1}/#1.5}}%
% better:
\def\coeff #1{\xintiTrunc {40}
   {\the\numexpr 2*\xintiiMON{#1}\relax/\the\numexpr 2*#1+1\relax [0]}}%
% better still:
\def\coeff #1{\xintiTrunc {40}
 {\the\numexpr\ifodd #1 -2\else2\fi\relax/\the\numexpr 2*#1+1\relax [0]}}%
% (-1)^n/(n+1/2) times 10^40, truncated to an integer.
\[ \sum_{n=0}^{n=50} \frac{(-1)^n}{n+\frac12} \approx
       \xintTrunc {40}{\xintiSeries {0}{50}{\coeff}[-40]}\dots\]
```

$$\sum_{n=0}^{n=50} \frac{(-1)^n}{n+\frac{1}{2}} \approx 1.58059930649352504123678950695672641448 10$$

We should have cut out at least the last two digits: truncating errors originating with the first coefficients of the sum will never go away, and each truncation introduces an uncertainty in the last digit, so as we have 40 terms, we should trash the last two digits, or at least round at 38

digits. It is interesting to compare with the computation where rounding rather than truncation is used, and with the decimal expansion of the exactly computed partial sum of the series:

```
\def\coeff #1{\xintiRound {40} % rounding at 40
  {\the\numexpr\ifodd #1 -2\else2\fi\relax/\the\numexpr 2*#1+1\relax [0]}}%
% (-1)^n/(n+1/2) times 10^40, rounded to an integer.
\[ \sum_{n=0}^{n=50} \frac{(-1)^n}{n+\frac12} \approx
       \xintTrunc {40}{\xintiSeries {0}{50}{\coeff}[-40]}\]
\def\exactcoeff #1%
  {\the\numexpr\ifodd #1 -2\else2\fi\relax/\the\numexpr 2*#1+1\relax [0]}%
\[ \sum_{n=0}^{n=50} \frac{(-1)^n}{n+\frac12}
   = \xintTrunc {50}{\xintSeries {0}{50}{\exactcoeff}}\dots\]
```

$$\sum_{n=0}^{n=50} \frac{(-1)^n}{n+\frac{1}{2}} \approx 1.58059930649352504123678950695672641448068680288367\ldots$$

$$\sum_{n=0}^{n=50} \frac{(-1)^n}{n+\frac{1}{2}} = 1.58059930649352504123678950695672641448068680288367\ldots$$

This shows indeed that our sum of truncated terms estimated wrongly the 39th and 40th digits of the exact result[74] and that the sum of rounded terms fared a bit better.

16.4. \xintRationalSeries

code

num Frac Frac
x f f ★

\xintRationalSeries{A}{B}{f}{\ratio} evaluates $\sum_{n=A}^{n=B} F(n)$, where F(n) is specified indirectly via the data of f=F(A) and the one-parameter macro \ratio which must be such that \macro{n} expands to F(n)/F(n-1). The name indicates that \xintRationalSeries was designed to be useful in the cases where F(n)/F(n-1) is a rational function of n but it may be anything expanding to a fraction. The macro \ratio must be an expandable-only compatible macro and expand to its value after iterated full expansion of its first item. A and B are fed to a \numexpr hence may be count registers or arithmetic expressions built with such; they must obey the TEX bound. The initial term f may be a macro \f, it will be expanded to its value representing F(A).

```
\def\ratio #1{2/#1[0]}% 2/n, to compute exp(2)
\cnta 0 % previously declared count
\begin{quote}
\loop \fdef\z {\xintRationalSeries {0}{\cnta}{1}{\ratio }}%
\noindent$\sum_{n=0}^{\the\cnta} \frac{2^n}{n!}=
           \xintTrunc{12}\z\dots=
           \xintTeXFrac\z=\xintTeXFrac{\xintIrr\z}$\vtop to 5pt{}\par
\ifnum\cnta<20 \advance\cnta 1 \repeat
\end{quote}
```

$\sum_{n=0}^{0} \frac{2^n}{n!} = 1.000000000000\cdots = 1 = 1$

$\sum_{n=0}^{1} \frac{2^n}{n!} = 3.000000000000\cdots = 3 = 3$

$\sum_{n=0}^{2} \frac{2^n}{n!} = 5.000000000000\cdots = \frac{10}{2} = 5$

$\sum_{n=0}^{3} \frac{2^n}{n!} = 6.333333333333\cdots = \frac{38}{6} = \frac{19}{3}$

$\sum_{n=0}^{4} \frac{2^n}{n!} = 7.000000000000\cdots = \frac{168}{24} = 7$

$\sum_{n=0}^{5} \frac{2^n}{n!} = 7.266666666666\cdots = \frac{872}{120} = \frac{109}{15}$

[74] as the series is alternating, we can roughly expect an error of $\sqrt{40}$ and the last two digits are off by 4 units, which is not contradictory to our expectations.

$\sum_{n=0}^{6} \frac{2^n}{n!} = 7.355555555555\cdots = \frac{5296}{720} = \frac{331}{45}$

$\sum_{n=0}^{7} \frac{2^n}{n!} = 7.380952380952\cdots = \frac{37200}{5040} = \frac{155}{21}$

$\sum_{n=0}^{8} \frac{2^n}{n!} = 7.387301587301\cdots = \frac{297856}{40320} = \frac{2327}{315}$

$\sum_{n=0}^{9} \frac{2^n}{n!} = 7.388712522045\cdots = \frac{2681216}{362880} = \frac{20947}{2835}$

$\sum_{n=0}^{10} \frac{2^n}{n!} = 7.388994708994\cdots = \frac{26813184}{3628800} = \frac{34913}{4725}$

$\sum_{n=0}^{11} \frac{2^n}{n!} = 7.389046015712\cdots = \frac{294947072}{39916800} = \frac{164591}{22275}$

$\sum_{n=0}^{12} \frac{2^n}{n!} = 7.389054566832\cdots = \frac{3539368960}{479001600} = \frac{691283}{93555}$

$\sum_{n=0}^{13} \frac{2^n}{n!} = 7.389055882389\cdots = \frac{46011804672}{6227020800} = \frac{14977801}{2027025}$

$\sum_{n=0}^{14} \frac{2^n}{n!} = 7.389056070325\cdots = \frac{644165281792}{87178291200} = \frac{314533829}{42567525}$

$\sum_{n=0}^{15} \frac{2^n}{n!} = 7.389056095384\cdots = \frac{9662479259648}{1307674368000} = \frac{4718007451}{638512875}$

$\sum_{n=0}^{16} \frac{2^n}{n!} = 7.389056098516\cdots = \frac{154599668219904}{20922789888000} = \frac{1572669151}{212837625}$

$\sum_{n=0}^{17} \frac{2^n}{n!} = 7.389056098884\cdots = \frac{2628194359869440}{355687428096000} = \frac{16041225341}{2170943775}$

$\sum_{n=0}^{18} \frac{2^n}{n!} = 7.389056098925\cdots = \frac{47307498477912064}{6402373705728000} = \frac{103122162907}{13956067125}$

$\sum_{n=0}^{19} \frac{2^n}{n!} = 7.389056098930\cdots = \frac{898842471080853504}{121645100408832000} = \frac{4571749222213}{618718975875}$

$\sum_{n=0}^{20} \frac{2^n}{n!} = 7.389056098930\cdots = \frac{17976849421618118656}{2432902008176640000} = \frac{68576238333199}{9280784638125}$

```
\def\ratio #1{-1/#1[0]}% -1/n, comes from the series of exp(-1)
\cnta 0 % previously declared count
\begin{quote}
\loop
\fdef\z {\xintRationalSeries {0}{\cnta}{1}{\ratio }}%
\noindent$\sum_{n=0}^{\the\cnta} \frac{(-1)^n}{n!}=
  \xintTrunc{20}\z\dots=\xintTeXFrac{\z}=\xintTeXFrac{\xintIrr\z}$%
        \vtop to 5pt{}\par
\ifnum\cnta<20 \advance\cnta 1 \repeat
\end{quote}
```

$\sum_{n=0}^{0} \frac{(-1)^n}{n!} = 1.000000000000000000\cdots = 1 = 1$

$\sum_{n=0}^{1} \frac{(-1)^n}{n!} = 0\cdots = 0 = 0$

$\sum_{n=0}^{2} \frac{(-1)^n}{n!} = 0.500000000000000000\cdots = \frac{1}{2} = \frac{1}{2}$

$\sum_{n=0}^{3} \frac{(-1)^n}{n!} = 0.333333333333333333\cdots = \frac{2}{6} = \frac{1}{3}$

$\sum_{n=0}^{4} \frac{(-1)^n}{n!} = 0.375000000000000000\cdots = \frac{9}{24} = \frac{3}{8}$

$\sum_{n=0}^{5} \frac{(-1)^n}{n!} = 0.366666666666666666\cdots = \frac{44}{120} = \frac{11}{30}$

$\sum_{n=0}^{6} \frac{(-1)^n}{n!} = 0.368055555555555555\cdots = \frac{265}{720} = \frac{53}{144}$

$\sum_{n=0}^{7} \frac{(-1)^n}{n!} = 0.367857142857142857\cdots = \frac{1854}{5040} = \frac{103}{280}$

$\sum_{n=0}^{8} \frac{(-1)^n}{n!} = 0.367881944444444444\cdots = \frac{14833}{40320} = \frac{2119}{5760}$

$\sum_{n=0}^{9} \frac{(-1)^n}{n!} = 0.367879188712522045\cdots = \frac{133496}{362880} = \frac{16687}{45360}$

$\sum_{n=0}^{10} \frac{(-1)^n}{n!} = 0.367879464285714285\cdots = \frac{1334961}{3628800} = \frac{16481}{44800}$

$\sum_{n=0}^{11} \frac{(-1)^n}{n!} = 0.36787943923360590027\cdots = \frac{14684570}{39916800} = \frac{1468457}{3991680}$

$\sum_{n=0}^{12} \frac{(-1)^n}{n!} = 0.36787944132128159905\cdots = \frac{176214841}{479001600} = \frac{16019531}{43545600}$

$\sum_{n=0}^{13} \frac{(-1)^n}{n!} = 0.36787944116069116069\cdots = \frac{2290792932}{6227020800} = \frac{63633137}{172972800}$

$\sum_{n=0}^{14} \frac{(-1)^n}{n!} = 0.36787944117216190628\cdots = \frac{32071101049}{87178291200} = \frac{2467007773}{6706022400}$

$\sum_{n=0}^{15} \frac{(-1)^n}{n!} = 0.36787944117139718991\cdots = \frac{481066515734}{1307674368000} = \frac{34361893981}{93405312000}$

$\sum_{n=0}^{16} \frac{(-1)^n}{n!} = 0.36787944117144498468\cdots = \frac{7697064251745}{20922789888000} = \frac{15549624751}{42268262400}$

$\sum_{n=0}^{17} \frac{(-1)^n}{n!} = 0.36787944117144217323\cdots = \frac{130850092279664}{355687428096000} = \frac{8178130767479}{22230464256000}$

$\sum_{n=0}^{18} \frac{(-1)^n}{n!} = 0.36787944117144232942\cdots = \frac{2355301661033953}{6402373705728000} = \frac{138547156531409}{376610217984000}$

$\sum_{n=0}^{19} \frac{(-1)^n}{n!} = 0.36787944117144232120\cdots = \frac{44750731559645106}{121645100408832000} = \frac{92079694567171}{250298560512000}$

$\sum_{n=0}^{20} \frac{(-1)^n}{n!} = 0.36787944117144232161\cdots = \frac{895014631192902121}{2432902008176640000} = \frac{4282366656425369}{11640679464960000}$

We can incorporate an indeterminate if we define \ratio to be a macro with two parameters: \def\ratioexp #1#2{\xintDiv{#1}{#2}}% x/n: x=#1, n=#2. Then, if \x expands to some fraction x, the macro

```
\xintRationalSeries {0}{b}{1}{\ratioexp{\x}}
```

will compute $\sum_{n=0}^{n=b} x^n/n!$:

```
\cnta 0
\def\ratioexp #1#2{\xintDiv{#1}{#2}}% #1/#2
\loop
\noindent
$\sum_{n=0}^{\the\cnta} (.57)^n/n! = \xintTrunc {50}
     {\xintRationalSeries {0}{\cnta}{1}{\ratioexp{.57}}}\dots$
     \vtop to 5pt {}\endgraf
\ifnum\cnta<50 \advance\cnta 10 \repeat
```

$\sum_{n=0}^{0} (.57)^n/n! = 1.00000000000000000000000000000000000000000000000000\dots$

$\sum_{n=0}^{10} (.57)^n/n! = 1.76826705137947002480668058035714285714285714285714\dots$

$\sum_{n=0}^{20} (.57)^n/n! = 1.76826705143373515162089324271187082272833005529082\dots$

$\sum_{n=0}^{30} (.57)^n/n! = 1.76826705143373515162089339282382144915484884979430\dots$

$\sum_{n=0}^{40} (.57)^n/n! = 1.76826705143373515162089339282382144915485219867776\dots$

$\sum_{n=0}^{50} (.57)^n/n! = 1.76826705143373515162089339282382144915485219867776\dots$

Observe that in this last example the x was directly inserted; if it had been a more complicated explicit fraction it would have been worthwile to use \ratioexp\x with \x defined to expand to its value. In the further situation where this fraction x is not explicit but itself defined via a complicated, and time-costly, formula, it should be noted that \xintRationalSeries will do again the evaluation of \x for each term of the partial sum. The easiest is thus when x can be defined as an \edef. If however, you are in an expandable-only context and cannot store in a macro like \x the value to be used, a variant of \xintRationalSeries is needed which will first evaluate this \x and then use this result without recomputing it. This is \xintRationalSeriesX, documented next.

Here is a slightly more complicated evaluation:

```
\cnta 1
\begin{multicols}{2}
\loop \fdef\z {\xintRationalSeries
                  {\cnta}
                  {2*\cnta-1}
                  {\xintiiPow {\the\cnta}{\cnta}/\xintiiFac{\cnta}}
```

```
                    {\ratioexp{\the\cnta}}}%
\fdef\w {\xintRationalSeries {0}{2*\cnta-1}{1}{\ratioexp{\the\cnta}}}%
\noindent
$\sum_{n=\the\cnta}^{\the\numexpr 2*\cnta-1\relax} \frac{\the\cnta^n}{n!}/%
          \sum_{n=0}^{\the\numexpr 2*\cnta-1\relax} \frac{\the\cnta^n}{n!} =
          \xintTrunc{8}{\xintDiv\z\w}\dots$ \vtop to 5pt{}\endgraf
\ifnum\cnta<20 \advance\cnta 1 \repeat
\end{multicols}
```

$\sum_{n=1}^{1} \frac{1^n}{n!} / \sum_{n=0}^{1} \frac{1^n}{n!} = 0.50000000\dots$
$\sum_{n=2}^{3} \frac{2^n}{n!} / \sum_{n=0}^{3} \frac{2^n}{n!} = 0.52631578\dots$
$\sum_{n=3}^{5} \frac{3^n}{n!} / \sum_{n=0}^{5} \frac{3^n}{n!} = 0.53804347\dots$
$\sum_{n=4}^{7} \frac{4^n}{n!} / \sum_{n=0}^{7} \frac{4^n}{n!} = 0.54317053\dots$
$\sum_{n=5}^{9} \frac{5^n}{n!} / \sum_{n=0}^{9} \frac{5^n}{n!} = 0.54502576\dots$
$\sum_{n=6}^{11} \frac{6^n}{n!} / \sum_{n=0}^{11} \frac{6^n}{n!} = 0.54518217\dots$
$\sum_{n=7}^{13} \frac{7^n}{n!} / \sum_{n=0}^{13} \frac{7^n}{n!} = 0.54445274\dots$
$\sum_{n=8}^{15} \frac{8^n}{n!} / \sum_{n=0}^{15} \frac{8^n}{n!} = 0.54327992\dots$
$\sum_{n=9}^{17} \frac{9^n}{n!} / \sum_{n=0}^{17} \frac{9^n}{n!} = 0.54191055\dots$
$\sum_{n=10}^{19} \frac{10^n}{n!} / \sum_{n=0}^{19} \frac{10^n}{n!} = 0.54048295\dots$
$\sum_{n=11}^{21} \frac{11^n}{n!} / \sum_{n=0}^{21} \frac{11^n}{n!} = 0.53907332\dots$
$\sum_{n=12}^{23} \frac{12^n}{n!} / \sum_{n=0}^{23} \frac{12^n}{n!} = 0.53772178\dots$
$\sum_{n=13}^{25} \frac{13^n}{n!} / \sum_{n=0}^{25} \frac{13^n}{n!} = 0.53644744\dots$
$\sum_{n=14}^{27} \frac{14^n}{n!} / \sum_{n=0}^{27} \frac{14^n}{n!} = 0.53525726\dots$
$\sum_{n=15}^{29} \frac{15^n}{n!} / \sum_{n=0}^{29} \frac{15^n}{n!} = 0.53415135\dots$
$\sum_{n=16}^{31} \frac{16^n}{n!} / \sum_{n=0}^{31} \frac{16^n}{n!} = 0.53312615\dots$
$\sum_{n=17}^{33} \frac{17^n}{n!} / \sum_{n=0}^{33} \frac{17^n}{n!} = 0.53217628\dots$
$\sum_{n=18}^{35} \frac{18^n}{n!} / \sum_{n=0}^{35} \frac{18^n}{n!} = 0.53129566\dots$
$\sum_{n=19}^{37} \frac{19^n}{n!} / \sum_{n=0}^{37} \frac{19^n}{n!} = 0.53047810\dots$
$\sum_{n=20}^{39} \frac{20^n}{n!} / \sum_{n=0}^{39} \frac{20^n}{n!} = 0.52971771\dots$

16.5. \xintRationalSeriesX

code

um Frac Frac
x f f f ⋆

\xintRationalSeriesX{A}{B}{\first}{\ratio}{\g} is a parametrized version of \xintRationalSeries where \first is now a one-parameter macro such that \first{\g} gives the initial term and \ratio is a two-parameter macro such that \ratio{n}{\g} represents the ratio of one term to the previous one. The parameter \g is evaluated only once at the beginning of the computation, and can thus itself be the yet unevaluated result of a previous computation.

Let \ratio be such a two-parameter macro; note the subtle differences between

```
\xintRationalSeries {A}{B}{\first}{\ratio{\g}}
and \xintRationalSeriesX {A}{B}{\first}{\ratio}{\g}.
```

First the location of braces differ... then, in the former case \first is a *no-parameter* macro expanding to a fractional number, and in the latter, it is a *one-parameter* macro which will use \g. Furthermore the X variant will expand \g at the very beginning whereas the former non-X former variant will evaluate it each time it needs it (which is bad if this evaluation is time-costly, but good if \g is a big explicit fraction encapsulated in a macro).

The example will use the macro \xintPowerSeries which computes efficiently exact partial sums of power series, and is discussed in the next section.

```
\def\firstterm #1{1[0]}% first term of the exponential series
% although it is the constant 1, here it must be defined as a
% one-parameter macro. Next comes the ratio function for exp:
\def\ratioexp  #1#2{\xintDiv {#1}{#2}}% x/n
% These are the (-1)^{n-1}/n of the log(1+h) series:
\def\coefflog #1{\the\numexpr\ifodd #1 1\else-1\fi\relax/#1[0]}%
% Let L(h) be the first 10 terms of the log(1+h) series and
% let E(t) be the first 10 terms of the exp(t) series.
% The following computes E(L(a/10)) for a=1,...,12.
\begin{multicols}{3}\raggedcolumns
\cnta 0
\loop
```

```
\noindent\xintTrunc {18}{%
     \xintRationalSeriesX {0}{9}{\firstterm}{\ratioexp}
         {\xintPowerSeries{1}{10}{\coefflog}{\the\cnta[-1]}}}\dots
\endgraf
\ifnum\cnta < 12 \advance \cnta 1 \repeat
\end{multicols}
```

1.000000000000000000... 1.499954310225476533... 1.907197560339468199...
1.099999999999083906... 1.599659266069210466... 1.845117565491393752...
1.199999998111624029... 1.698137473697423757... 1.593831932293536053...
1.299999835744121464... 1.791898112718884531...
1.399996091955359088... 1.870485649686617459...

These completely exact operations rapidly create numbers with many digits. Let us print in full the raw fractions created by the operation illustrated above:

E(L(1[-1]))=16359144369311788930343108880608763414825073579102349765726131401415910739539139378719946574105733667711657325234129521868 8/1487194942665946638644674560000000000[-90] (length of numerator: 127)

E(L(12[-2]))=16656583357757234467643895619026874191327320993157183247568125775059356018362231934396045400537542264448715028348166448083362882112998458872460667950411608822312198051669272737296607284122130748172615228417547299717121487194942665946638644674560000000000[-180] (length of numerator: 217)

E(L(123[-3]))=16701199206005550269986632390690022782662159669686695081451916268879387348627326998654607865880397901400311690337802593514890044881469893662763355806673815195853060316740612785673175692992742863679398303407413205084692383474722719804622771982161117197045873620257690491156872157127231823865270550337350533123/1487194942665946638644674560000000000[-270] (length of numerator: 307)

We see that the denominators here remain the same, as our input only had various powers of ten as denominators, and xintfrac efficiently assemble (some only, as we can see) powers of ten. Notice that 1 more digit in an input denominator seems to mean 90 more in the raw output. We can check that with some other test cases:

E(L(1/7))=48228203862750885848048032297655163193719083498752126622944863683478921463537652042196695417787645279493 3/42199678381622718782443725277603122786330638063321058081317416560950056936721328812056161288192000000000[0] (length of numerator: 105; length of denominator: 105)

E(L(1/71))=619003967078535034640655099515947654027294818288439846288288922997238323197498859719400152182490594357208368328392373910672874993163246058732446704305028542916962821162875860387813549997353988721286046761040668975799982582643863990035761895246771100033570420271671092203332984981842891074510835779826956944462566758343900417497150172256263898307611707757919998778523559418340083473473151235522560000000000[0] (length of numerator: 203; length of denominator: 203)

E(L(1/712))=3003564353778406020559670408411885925389093114199308387996560136260710297841744968192908849580413620381324217440556141531542682924131728705303727345332905581415389151732575694112320026364569495366534918031439051104610487529796192058205725999641657806615904929048989464635331466622338692499935178105220909766968489591763105361775507557039697364359215352246041089232853253973804191120212141242471588173404925471664008247098734098515193250428149424064596788874441470533147848207863549778847000617103264666638782677019019130113930837421531810478062025966102914017525760000000000[0] (length of numerator: 288; length of denominator: 288)

Thus decimal numbers such as 0.123 (equivalently 123[-3]) give less computing intensive tasks than fractions such as 1/712: in the case of decimal numbers the (raw) denominators originate in the coefficients of the series themselves, powers of ten of the input within brackets being treated separately. And even then the numerators will grow with the size of the input in a sort

of linear way, the coefficient being given by the order of series: here 10 from the log and 9 from the exp, so 90. One more digit in the input means 90 more digits in the numerator of the output: obviously we can not go on composing such partial sums of series and hope that xint will joyfully do all at the speed of light!

Hence, truncating the output (or better, rounding) is the only way to go if one needs a general calculus of special functions. This is why the package xintseries provides, besides \xintSeries, \xintRationalSeries, or \xintPowerSeries which compute *exact* sums, \xintFxPtPowerSeries for fixed-point computations and a (tentative naive) \xintFloatPowerSeries.

16.6. \xintPowerSeries

code

num Frac Frac
x f f ★

\xintPowerSeries{A}{B}{\coeff}{f} evaluates the sum $\sum_{n=A}^{n=B}$\coeff{n}$\cdot f^n$. The initial and final indices are given to a \numexpr expression. The \coeff macro (which, as argument to \xintPowerSeries is expanded only at the time \coeff{n} is needed) should be defined as a one-parameter expandable macro, its input will be an explicit number.

The f can be either a fraction directly input or a macro \f expanding to such a fraction. It is actually more efficient to encapsulate an explicit fraction f in such a macro, if it has big numerators and denominators (`big' means hundreds of digits) as it will then take less space in the processing until being (repeatedly) used.

This macro computes the *exact* result (one can use it also for polynomial evaluation), using a Horner scheme which helps avoiding a denominator build-up (this problem however, even if using a naive additive approach, is much less acute since release 1.1 and its new policy regarding \xintAdd).

```
\def\geom #1{1[0]} % the geometric series
\def\f {5/17[0]}
\[ \sum_{n=0}^{n=20} \Bigl(\frac 5{17}\Bigr)^n
 =\xintTeXFrac{\xintIrr{\xintPowerSeries {0}{20}{\geom}{\f}}}
 =\xintTeXFrac{\xinttheexpr (17^21-5^21)/12/17^20\relax}\]
```

$$\sum_{n=0}^{n=20}\left(\frac{5}{17}\right)^n = \frac{57576611593776579768853 41}{40642314066475725224016 01} = \frac{69091933912531895722624092}{48770776879770870268819212}$$

```
\def\coefflog #1{1/#1[0]}% 1/n
\def\f {1/2[0]}%
\[ \log 2 \approx \sum_{n=1}^{20} \frac1{n\cdot 2^n}
    = \xintTeXFrac {\xintIrr {\xintPowerSeries {1}{20}{\coefflog}{\f}}}\]
\[ \log 2 \approx \sum_{n=1}^{50} \frac1{n\cdot 2^n}
    = \xintTeXFrac {\xintIrr {\xintPowerSeries {1}{50}{\coefflog}{\f}}}\]
```

$$\log 2 \approx \sum_{n=1}^{20}\frac{1}{n\cdot 2^n} = \frac{42299423848079}{61025172848640}$$

$$\log 2 \approx \sum_{n=1}^{50}\frac{1}{n\cdot 2^n} = \frac{60463469751752265663579884559739219}{87230347965792839223946208178339840}$$

```
\setlength{\columnsep}{0pt}
\begin{multicols}{3}
\cnta 1 % previously declared count
\loop   % in this loop we recompute from scratch each partial sum!
% we can afford that, as \xintPowerSeries is fast enough.
\noindent\hbox to 2em{\hfil\texttt{\the\cnta.} }%
         \xintTrunc {12}
             {\xintPowerSeries {1}{\cnta}{\coefflog}{\f}}\dots
```

```
 \endgraf
 \ifnum \cnta < 30 \advance\cnta 1 \repeat
 \end{multicols}
```

```
 1. 0.500000000000...        11. 0.693109245355...        21. 0.693147159757...
 2. 0.625000000000...        12. 0.693129590407...        22. 0.693147170594...
 3. 0.666666666666...        13. 0.693138980431...        23. 0.693147175777...
 4. 0.682291666666...        14. 0.693143340085...        24. 0.693147178261...
 5. 0.688541666666...        15. 0.693145374590...        25. 0.693147179453...
 6. 0.691145833333...        16. 0.693146328265...        26. 0.693147180026...
 7. 0.692261904761...        17. 0.693146777052...        27. 0.693147180302...
 8. 0.692750186011...        18. 0.693146988980...        28. 0.693147180435...
 9. 0.692967199900...        19. 0.693147089367...        29. 0.693147180499...
10. 0.693064856150...        20. 0.693147137051...        30. 0.693147180530...
```

```
 \def\coeffarctg  #1{1/\the\numexpr\ifodd #1 -2*#1-1\else2*#1+1\fi\relax }%
 % the above gives (-1)^n/(2n+1). The sign being in the denominator,
 %             **** no [0] should be added ****,
 % else nothing is guaranteed to work (even if it could by sheer luck)
 % Notice in passing this aspect of \numexpr:
 %         ****  \numexpr -(1)\relax is ilegal !!! ****
 \def\f {1/25[0]}% 1/5^2
 \[\mathrm{Arctg}(\frac15)\approx \frac15\sum_{n=0}^{15} \frac{(-1)^n}{(2n+1)25^n}
 = \xintTeXFrac{\xintIrr {\xintDiv {\xintPowerSeries {0}{15}{\coeffarctg}{\f}}{5}}}\]
```

$$\mathrm{Arctg}(\frac{1}{5}) \approx \frac{1}{5}\sum_{n=0}^{15}\frac{(-1)^n}{(2n+1)25^n} = \frac{165918726519122955895391793269168}{840539304153062403202056884765625}$$

code

16.7. \xintPowerSeriesX

num num Frac Frac
x x f f

This is the same as \xintPowerSeries apart from the fact that the last parameter f is expanded once and for all before being then used repeatedly. If the f parameter is to be an explicit big fraction with many (dozens) digits, rather than using it directly it is slightly better to have some macro \g defined to expand to the explicit fraction and then use \xintPowerSeries with \g; but if f has not yet been evaluated and will be the output of a complicated expansion of some \f, and if, due to an expanding only context, doing \edef\g{\f} is no option, then \xintPowerSeriesX should be used with \f as last parameter.

```
 \def\ratioexp #1#2{\xintDiv {#1}{#2}}% x/n
 % These are the (-1)^{n-1}/n of the log(1+h) series:
 \def\coefflog #1{\the\numexpr\ifodd #1 1\else-1\fi\relax/#1[0]}%
 % Let L(h) be the first 10 terms of the log(1+h) series and
 % let E(t) be the first 10 terms of the exp(t) series.
 % The following computes L(E(a/10)-1) for a=1,..., 12.
 \begin{multicols}{3}\raggedcolumns
 \cnta 1
 \loop
 \noindent\xintTrunc {18}{%
    \xintPowerSeriesX {1}{10}{\coefflog}
   {\xintSub
       {\xintRationalSeries {0}{9}{1[0]}{\ratioexp{\the\cnta[-1]}}}
       {1}}}\dots
 \endgraf
```

```
  \ifnum\cnta < 12 \advance \cnta 1 \repeat
  \end{multicols}
```

0.099999999998556159... 0.499511320760604148... -1.597091692317639401...
0.199999995263443554... 0.593980619762352217... -12.648937932093322763...
0.299999338075041781... 0.645144282733914916... -66.259639046914679687...
0.399974460740121112... 0.398118280111436442... -304.768437445462801227...

code
16.8. \xintFxPtPowerSeries

num num
x x ★
Frac Frac num
f f x ★

\xintFxPtPowerSeries{A}{B}{\coeff}{f}{D} computes $\sum_{n=A}^{n=B}$\coeff{n}·f^n with each term of the series truncated to D digits after the decimal point. As usual, A and B are completely expanded through their inclusion in a \numexpr expression. Regarding D it will be similarly be expanded each time it is used inside an \xintTrunc. The one-parameter macro \coeff is similarly expanded at the time it is used inside the computations. Idem for f. If f itself is some complicated macro it is thus better to use the variant \xintFxPtPowerSeriesX which expands it first and then uses the result of that expansion.

The current (1.04) implementation is: the first power f^A is computed exactly, then *truncated*. Then each successive power is obtained from the previous one by multiplication by the exact value of f, and truncated. And \coeff{n}·f^n is obtained from that by multiplying by \coeff{n} (untruncated) and then truncating. Finally the sum is computed exactly. Apart from that \xintFxPtPowerSeries (where FxPt means `fixed-point') is like \xintPowerSeries.

There should be a variant for things of the type $\sum c_n \frac{f^n}{n!}$ to avoid having to compute the factorial from scratch at each coefficient, the same way \xintFxPtPowerSeries does not compute f^n from scratch at each n. Perhaps in the next package release.

$e^{-\frac{1}{2}} \approx$

1.00000000000000000000 0.60653056795634920635 0.60653065971263344622
0.50000000000000000000 0.60653066483754960317 0.60653065971263342289
0.62500000000000000000 0.60653065945526069224 0.60653065971263342361
0.60416666666666666667 0.60653065972437513778 0.60653065971263342359
0.60677083333333333333 0.60653065971214266299 0.60653065971263342359
0.60651041666666666667 0.60653065971265234943 0.60653065971263342359
0.60653211805555555555 0.60653065971263274611

```
  \def\coeffexp #1{1/\xintiiFac {#1}[0]}% 1/n!
  \def\f {-1/2[0]}% [0] for faster input parsing
  \cnta 0 % previously declared \count register
  \noindent\loop
  $\xintFxPtPowerSeries {0}{\cnta}{\coeffexp}{\f}{20}$\\
  \ifnum\cnta<19 \advance\cnta 1 \repeat\par
  \xintFxPtPowerSeries {0}{19}{\coeffexp}{\f}{25}= 0.6065306597126334236037992
```

It is no difficulty for xintfrac to compute exactly, with the help of \xintPowerSeries, the nineteenth partial sum, and to then give (the start of) its exact decimal expansion:

```
  \xintPowerSeries {0}{19}{\coeffexp}{\f} = 38682746160036397317757/63777066403145711616000
```

$= 0.60653065971263342360379915212 6\ldots$

Thus, one should always estimate a priori how many ending digits are not reliable: if there are N terms and N has k digits, then digits up to but excluding the last k may usually be trusted. If we are optimistic and the series is alternating we may even replace N with $\sqrt{N}$ to get the number k of digits possibly of dubious significance.

code

16.9. \xintFxPtPowerSeriesX

num num
Frac Frac x x
f f x ★

\xintFxPtPowerSeriesX{A}{B}{\coeff}{\f}{D} computes, exactly as \xintFxPtPowerSeries, the sum of \coeff{n}·\f^n from n=A to n=B with each term of the series being *truncated* to D digits after the decimal point. The sole difference is that \f is first expanded and it is the result of this which is used in the computations.

Let us illustrate this on the numerical exploration of the identity

```
log(1+x) = -log(1/(1+x))
```

Let L(h)=log(1+h), and D(h)=L(h)+L(-h/(1+h)). Theoretically thus, D(h)=0 but we shall evaluate L(h) and -h/(1+h) keeping only 10 terms of their respective series. We will assume h < 0.5. With only ten terms kept in the power series we do not have quite 3 digits precision as $2^{10} = 1024$. So it wouldn't make sense to evaluate things more precisely than, say circa 5 digits after the decimal points.

```
\cnta 0
\def\coefflog #1{\the\numexpr\ifodd#1 1\else-1\fi\relax/#1[0]}% (-1)^{n-1}/n
\def\coeffalt #1{\the\numexpr\ifodd#1 -1\else1\fi\relax [0]}%   (-1)^n
\begin{multicols}2
\loop
\noindent \hbox to 2.5cm {\hss\texttt{D(\the\cnta/100): }}%
\xintAdd {\xintFxPtPowerSeriesX {1}{10}{\coefflog}{\the\cnta [-2]}{5}}
         {\xintFxPtPowerSeriesX {1}{10}{\coefflog}
             {\xintFxPtPowerSeriesX {1}{10}{\coeffalt}{\the\cnta [-2]}{5}}
          {5}}\endgraf
\ifnum\cnta < 49 \advance\cnta 7 \repeat
\end{multicols}
```

```
  D(0/100): 0/1[0]                    D(28/100): 4/1[-5]
  D(7/100): 2/1[-5]                   D(35/100): 4/1[-5]
 D(14/100): 2/1[-5]                   D(42/100): 9/1[-5]
 D(21/100): 3/1[-5]                   D(49/100): 42/1[-5]
```

Let's say we evaluate functions on [-1/2,+1/2] with values more or less also in [-1/2,+1/2] and we want to keep 4 digits of precision. So, roughly we need at least 14 terms in series like the geometric or log series. Let's make this 15. Then it doesn't make sense to compute intermediate summands with more than 6 digits precision. So we compute with 6 digits precision but return only 4 digits (rounded) after the decimal point. This result with 4 post-decimal points precision is then used as input to the next evaluation.

```
\begin{multicols}2
\loop
\noindent \hbox to 2.5cm {\hss\texttt{D(\the\cnta/100): }}%
\dtt{\xintRound{4}
 {\xintAdd {\xintFxPtPowerSeriesX {1}{15}{\coefflog}{\the\cnta [-2]}{6}}
           {\xintFxPtPowerSeriesX {1}{15}{\coefflog}
                  {\xintRound {4}{\xintFxPtPowerSeriesX {1}{15}{\coeffalt}
                                 {\the\cnta [-2]}{6}}}
            {6}}%
 }}\endgraf
\ifnum\cnta < 49 \advance\cnta 7 \repeat
\end{multicols}
```

```
 D(0/100): 0                          D(28/100): -0.0001
 D(7/100): 0.0000                     D(35/100): -0.0001
D(14/100): 0.0000                     D(42/100): -0.0000
D(21/100): -0.0001                    D(49/100): -0.0001
```

Not bad... I have cheated a bit: the `four-digits precise' numeric evaluations were left unrounded in the final addition. However the inner rounding to four digits worked fine and made the next step faster than it would have been with longer inputs. The morale is that one should not use the raw results of \xintFxPtPowerSeriesX with the D digits with which it was computed, as the last are to be considered garbage. Rather, one should keep from the output only some smaller number of digits. This will make further computations faster and not less precise. I guess there should be some macro to do this final truncating, or better, rounding, at a given number D'<D of digits. Maybe for the next release.

16.10. \xintFloatPowerSeries

code

[num x] num x num x
 Frac   Frac
  f   f   ⋆

\xintFloatPowerSeries[P]{A}{B}{\coeff}{f} computes $\sum_{n=A}^{n=B}$\coeff{n}$\cdot f^n$ with a floating point precision given by the optional parameter P or by the current setting of \xintDigits.

In the current, preliminary, version, no attempt has been made to try to guarantee to the final result the precision P. Rather, P is used for all intermediate floating point evaluations. So rounding errors will make some of the last printed digits invalid. The operations done are first the evaluation of f^A using \xintFloatPow, then each successive power is obtained from this first one by multiplication by f using \xintFloatMul, then again with \xintFloatMul this is multiplied with \coeff{n}, and the sum is done adding one term at a time with \xintFloatAdd. To sum up, this is just the naive transformation of \xintFxPtPowerSeries from fixed point to floating point.

```
\def\coefflog #1{\the\numexpr\ifodd#1 1\else-1\fi\relax/#1[0]}%
\xintFloatPowerSeries [8]{1}{30}{\coefflog}{-1/2[0]}
-6.9314718e-1
```

16.11. \xintFloatPowerSeriesX

code

[num x] num x num x
 Frac   Frac
  f   f   ⋆

\xintFloatPowerSeriesX[P]{A}{B}{\coeff}{f} is like \xintFloatPowerSeries with the difference that f is expanded once and for all at the start of the computation, thus allowing efficient chaining of such series evaluations.

```
\def\coeffexp #1{1/\xintiiFac {#1}[0]}% 1/n! (exact, not float)
\def\coefflog #1{\the\numexpr\ifodd#1 1\else-1\fi\relax/#1[0]}%
\xintFloatPowerSeriesX [8]{0}{30}{\coeffexp}
    {\xintFloatPowerSeries [8]{1}{30}{\coefflog}{-1/2[0]}}
5.0000001e-1
```

16.12. Computing log 2 and π

In this final section, the use of \xintFxPtPowerSeries (and \xintPowerSeries) will be illustrated on the (expandable... why make things simple when it is so easy to make them difficult!) computations of the first digits of the decimal expansion of the familiar constants log 2 and π.

Let us start with log 2. We will get it from this formula (which is left as an exercise):

```
log(2)=-2 log(1-13/256)-5 log(1-1/9)
```

The number of terms to be kept in the log series, for a desired precision of 10^{-D} was roughly estimated without much theoretical analysis. Computing exactly the partial sums with \xintPowerSeries and then printing the truncated values, from D=0 up to D=100 showed that it worked in terms of quality of the approximation. Because of possible strings of zeroes or nines in the exact decimal expansion (in the present case of log 2, strings of zeroes around the fourtieth and the

sixtieth decimals), this does not mean though that all digits printed were always exact. In the end one always end up having to compute at some higher level of desired precision to validate the earlier result.

Then we tried with \xintFxPtPowerSeries: this is worthwile only for D's at least 50, as the exact evaluations are faster (with these short-length f's) for a lower number of digits. And as expected the degradation in the quality of approximation was in this range of the order of two or three digits. This meant roughly that the 3+1=4 ending digits were wrong. Again, we ended up having to compute with five more digits and compare with the earlier value to validate it. We use truncation rather than rounding because our goal is not to obtain the correct rounded decimal expansion but the correct exact truncated one.

```
\def\coefflog #1{1/#1[0]}% 1/n
\def\xa {13/256[0]}%  we will compute log(1-13/256)
\def\xb {1/9[0]}%     we will compute log(1-1/9)
\def\LogTwo #1{%
  %  get log(2)=-2log(1-13/256)- 5log(1-1/9)
  \romannumeral0\expandafter\logtwo
  % number of terms for 1/9:
  \the\numexpr #1*150/143\expandafter.%
  % number of Terms for 13/256:
  \the\numexpr #1*100/129\expandafter.%
  % We print #1 digits, but we know the ending ones are garbage
  % Use \numexpr to allow a \count as #1
  \the\numexpr #1.%
}%
\def\logtwo #1.#2.#3.{%
  % #1=nb of terms for 1/9,
  % #2=nb of terms for 13/256,
  % #3=nb of digits for computations, also used for printing
  \xinttrunc {#3}% will terminate the \romannumeral0
  {\xintAdd
   {\xintMul {2}{\xintFxPtPowerSeries {1}{#2}{\coefflog}{\xa}{#3}}}
   {\xintMul {5}{\xintFxPtPowerSeries {1}{#1}{\coefflog}{\xb}{#3}}}%
  }%
}%
\noindent $\log 2 \approx \LogTwo {60}\dots$\endgraf
\noindent\phantom{$\log 2$}${}\approx{}$\printnumber{\LogTwo {65}}\dots\endgraf
\noindent\phantom{$\log 2$}${}\approx{}$\printnumber{\LogTwo {70}}\dots\endgraf
```

$\log 2 \approx 0.693147180559945309417232121458176568075500134360255254120484\ldots$
$\approx 0.693147180559945309417232121458176568075500134360255254120680000711\ldots$
$\approx 0.693147180559945309417232121458176568075500134360255254120680009493372 3\ldots$

Here is the code doing an exact evaluation of the partial sums. We have added a +1 to the number of digits for estimating the number of terms to keep from the log series: we experimented that this gets exactly the first D digits, for all values from D=0 to D=100, except in one case (D=40) where the last digit is wrong. For values of D higher than 100 it is more efficient to use the code using \xintFxPtPowerSeries.

```
\def\coefflog #1{1/#1[0]}% 1/n
\def\xa {13/256[0]}%  we will compute log(1-13/256)
\def\xb {1/9[0]}%     we will compute log(1-1/9)
\def\LogTwo #1{% get log(2)=-2log(1-13/256)- 5log(1-1/9)
  \romannumeral0\expandafter\logtwo
  \the\numexpr (#1+1)*150/143\expandafter.%
  \the\numexpr (#1+1)*100/129\expandafter.%
```

```
    \the\numexpr #1.%
  }%
  \def\logtwo #1.#2.#3.{% #3=nb of digits for truncating an EXACT partial sum
    \xinttrunc {#3}% will terminate the \romannumeral0
      {\xintAdd
        {\xintMul {2}{\xintPowerSeries {1}{#2}{\coefflog}{\xa}}}
        {\xintMul {5}{\xintPowerSeries {1}{#1}{\coefflog}{\xb}}}%
      }%
  }%
```

Let us turn now to Pi, computed with the Machin formula (but see also the approach via the Brent-Salamin algorithm with \xintfloatexpr) Again the numbers of terms to keep in the two arctg series were roughly estimated, and some experimentations showed that removing the last three digits was enough (at least for D=0-100 range). And the algorithm does print the correct digits when used with D=1000 (to be convinced of that one needs to run it for D=1000 and again, say for D=1010.) A theoretical analysis could help confirm that this algorithm always gets better than 10^{-D} precision, but again, strings of zeroes or nines encountered in the decimal expansion may falsify the ending digits, nines may be zeroes (and the last non-nine one should be increased) and zeroes may be nine (and the last non-zero one should be decreased).

```
  \def\coeffarctg #1{\the\numexpr\ifodd#1 -1\else1\fi\relax
                     /\the\numexpr 2*#1+1\relax [0]}%
  %\def\coeffarctg #1{\romannumeral0\xintmon{#1}/\the\numexpr 2*#1+1\relax }%
  \def\xa {1/25[0]}%      1/5^2, the [0] for faster parsing
  \def\xb {1/57121[0]}% 1/239^2, the [0] for faster parsing
  \def\Machin #1{% #1 may be a count register
    \romannumeral0\expandafter\machin
    % number of terms for arctg(1/5):
    \the\numexpr (#1+3)*5/7\expandafter.%
    % number of terms for arctg(1/239):
    \the\numexpr (#1+3)*10/45\expandafter.%
    % do the computations with 3 additional digits:
    \the\numexpr #1+3\expandafter.%
    % allow #1 to be a count register:
    \the\numexpr #1.%
  }%
  \def\machin #1.#2.#3.#4.{%
    \xinttrunc {#4}% will terminate the \romannumeral0
    {\xintSub
      {\xintMul {16/5}{\xintFxPtPowerSeries {0}{#1}{\coeffarctg}{\xa}{#3}}}
      {\xintMul{4/239}{\xintFxPtPowerSeries {0}{#2}{\coeffarctg}{\xb}{#3}}}%
    }%
  }%
  \begin{framed}
    \[ \pi = \Machin{60}\dots \]
  \end{framed}
```

$$\pi = 3.14159265358979323846264338327950288419716939937510582097494 4\ldots$$

Here is a variant\MachinBis, which evaluates the partial sums *exactly* using \xintPowerSeries, before their final truncation. No need for a ``+3'' then.

```
  \def\MachinBis #1{% #1 may be a count register,
    % the final result will be truncated to #1 digits post decimal point
```

```
  \romannumeral0\expandafter\machinbis
  % number of terms for arctg(1/5):
  \the\numexpr #1*5/7\expandafter.%
  % number of terms for arctg(1/239):
  \the\numexpr #1*10/45\expandafter.%
  % allow #1 to be a count register:
  \the\numexpr #1.%
}%
\def\machinbis #1.#2.#3.{%
  \xinttrunc {#3}% will terminate the \romannumeral0
   {\xintSub
     {\xintMul {16/5}{\xintPowerSeries {0}{#1}{\coeffarctg}{\xa}}}
     {\xintMul{4/239}{\xintPowerSeries {0}{#2}{\coeffarctg}{\xb}}}%
   }%
}%
```

Let us use this variant for a loop showing the build-up of digits:

```
\begin{multicols}{2}
  \cnta 0 % previously declared \count register
  \loop \noindent
        \centeredline{\dtt{\MachinBis{\cnta}}}%
  \ifnum\cnta < 30
  \advance\cnta 1 \repeat
\end{multicols}
```

```
3.
3.1
3.14
3.141
3.1415
3.14159
3.141592
3.1415926
3.14159265
3.141592653
3.1415926535
3.14159265358
3.141592653589
3.1415926535897
3.14159265358979
3.141592653589793
3.1415926535897932
3.14159265358979323
3.141592653589793238
3.1415926535897932384
3.14159265358979323846
3.141592653589793238462
3.1415926535897932384626
3.14159265358979323846264
3.141592653589793238462643
3.1415926535897932384626433
3.14159265358979323846264338
3.141592653589793238462643383
3.1415926535897932384626433832
3.14159265358979323846264338327
3.141592653589793238462643383279
```

You want more digits and have some time? Save the following to a file, it is the \Machin code. Compile with etex (or pdftex or xetex or luatex):

```
% Compile with e-TeX extensions enabled (etex, pdftex, ...)
\input xintfrac.sty
\input xintseries.sty
% pi = 16 Arctg(1/5) - 4 Arctg(1/239) (John Machin's formula)
\def\coeffarctg #1{\the\numexpr\ifodd#1 -1\else1\fi\relax
                   /\the\numexpr 2*#1+1\relax [0]}%
\def\xa {1/25[0]}%
\def\xb {1/57121[0]}%
\def\Machin #1{%
  \romannumeral0\expandafter\machin
```

```
  \the\numexpr (#1+3)*5/7\expandafter.%
  \the\numexpr (#1+3)*10/45\expandafter.%
  \the\numexpr #1+3\expandafter.%
  \the\numexpr #1.%
}%
\def\machin #1.#2.#3.#4.{%
  \xinttrunc {#4}% will terminate the \romannumeral0
    {\xintSub
      {\xintMul {16/5}{\xintFxPtPowerSeries {0}{#1}{\coeffarctg}{\xa}{#3}}}
      {\xintMul{4/239}{\xintFxPtPowerSeries {0}{#2}{\coeffarctg}{\xb}{#3}}}%
    }%
}%
\xintresettimer
\edef\Z {\Machin {1000}}
\edef\W {\xinttheseconds}
\immediate\write128{1000 places of pi via Machin formula (took \W s):}
\immediate\write128{\Z}
\bye
```

This will log the first 1000 digits of π after the decimal point. On my laptop (a 2012 model) this took about 5.05 seconds last time I tried.[75] [76]

As mentioned in the introduction, the file pi.tex by D. Roegel shows that orders of magnitude faster computations are possible within TEX, but recall our constraints of complete expandability and be merciful, please.

Why truncating rather than rounding? One of our main competitors on the market of scientific computing, a canadian product (not encumbered with expandability constraints, and having barely ever heard of TEX ;-), prints numbers rounded in the last digit. Why didn't we follow suit in the macros \xintFxPtPowerSeries and \xintFxPtPowerSeriesX? To round at D digits, and excluding a rewrite or cloning of the division algorithm which anyhow would add to it some overhead in its final steps, xintfrac needs to truncate at D+1, then round. And rounding loses information! So, with more time spent, we obtain a worst result than the one truncated at D+1 (one could imagine that additions and so on, done with only D digits, cost less; true, but this is a negligeable effect per summand compared to the additional cost for this term of having been truncated at D+1 then rounded). Rounding is the way to go when setting up algorithms to evaluate functions destined to be composed one after the other: exact algebraic operations with many summands and an f variable which is a fraction are costly and create an even bigger fraction; replacing f with a reasonable rounding, and rounding the result, is necessary to allow arbitrary chaining.

But, for the computation of a single constant, we are really interested in the exact decimal expansion, so we truncate and compute more terms until the earlier result gets validated. Finally if we do want the rounding we can always do it on a value computed with D+1 truncation.

[75] With 1.09i and earlier xint, this used to be 42 seconds; starting with 1.09j, and prior to 1.2, it was 16 seconds (this was probably due to a more efficient division with denominators at most 9999). The 1.2 xintcore achieved a further gain at 5.6 seconds.

[76] With \xintDigits :=1001\relax, the non-optimized implementation with the iter of xintexpr fame using the Brent-Salamin algorithm, took, last time I tried (1.2i), about 7 seconds on my laptop (the last two digits were wrong, which is ok as they serve as guard digits), and for obtaining about 500 digits, it was about 1.7s. This is not bad, taking into account that the syntax is almost free rolling speech, contrarily to the code above for the Machin formula computation; we would like to use the quadratically convergent Brent-Salamin algorithm for more digits, but with such computations with numbers of one thousand digits we are beyond the border of the reasonable range for xint. Innocent people not knowing what it means to compute with TEX, and with the extra constraint of expandability will wonder why this is at least thousands of times slower than with any other language (with a little Python program using the Decimal library, I timed the Brent-Salamin algorithm to 4.4ms for about 1000 digits and 1.14ms for 500 digits.) I will just say that for example digits are represented and manipulated via their ascii-code ! all computations must convert from ascii-code to cpu words; furthermore nothing can be stored away. And there is no memory storage with O(1) time access... if expandability is to be verified.

17. Macros of the xintcfrac package

.1 Introduction 227
.2 Package overview 227
.3 \xintCFrac 232
.4 \xintGCFrac 232
.5 \xintGGCFrac 232
.6 \xintGCtoGCx 233
.7 \xintFtoC 233
.8 \xintFtoCs 233
.9 \xintFtoCx 233
.10 \xintFtoGC 234
.11 \xintFGtoC 234
.12 \xintFtoCC 234
.13 \xintCstoF 235
.14 \xintCtoF 235
.15 \xintGCtoF 236
.16 \xintCstoCv 236
.17 \xintCtoCv 237
.18 \xintGCtoCv 237
.19 \xintFtoCv 237
.20 \xintFtoCCv 237
.21 \xintCntoF 237
.22 \xintGCntoF 238
.23 \xintCntoCs 238
.24 \xintCntoGC 238
.25 \xintGCntoGC 239
.26 \xintCstoGC 239
.27 \xintiCstoF, \xintiGCtoF, \xintiCstoCv, \xintiGCtoCv 240
.28 \xintGCtoGC 240
.29 Euler's number e 240

17.1. Introduction

First version of this package was included in release 1.04 (2013/04/25) of the xint bundle. It was kept almost unchanged until 1.09m of 2014/02/26 which brought some new macros: \xintFtoC, \xintCtoF, \xintCtoCv, dealing with sequences of braced partial quotients rather than comma separated ones, \xintFGtoC which is to produce ``guaranteed'' coefficients of some real number known approximately, and \xintGGCFrac for displaying arbitrary material as a continued fraction; also, some changes to existing macros: \xintFtoCs and \xintCntoCs insert spaces after the commas, \xintCstoF and \xintCstoCv authorize spaces in the input also before the commas.

\xintCstoF and \xintCstoCv create a partial dependency on xinttools as they use its \xintCSVtoList. Starting at 1.4n the loading of xinttools is done automatically, formerly it was up to user to do it.

17.2. Package overview

The package computes partial quotients and convergents of a fraction, or conversely start from coefficients and obtain the corresponding fraction; three macros \xintCFrac, \xintGCFrac and \xintGGCFrac are for typesetting, the others can be nested (if applicable) or see their outputs further processed by other macros from the xint bundle, particularly the macros of xinttools dealing with sequences of braced items or comma separated lists.

A *simple* continued fraction has coefficients [c0,c1,...,cN] (usually called partial quotients, but I dislike this entrenched terminology), where c0 is a positive or negative integer and the others are positive integers.

Typesetting is usually done via the amsmath macro \cfrac:

```
\[ c_0 + \cfrac{1}{c_1+\cfrac1{c_2+\cfrac1{c_3+\cfrac1{\ddots}}}}\]
```

$$c_0 + \cfrac{1}{c_1 + \cfrac{1}{c_2 + \cfrac{1}{c_3 + \cfrac{1}{\ddots}}}}$$

Here is a concrete example:

```
\[ \xintTeXFrac {208341/66317}=\xintCFrac {208341/66317}\]%
```

$$\frac{208341}{66317} = 3 + \cfrac{1}{7 + \cfrac{1}{15 + \cfrac{1}{1 + \cfrac{1}{292 + \cfrac{1}{2}}}}}$$

But it is the macro \xintCFrac which did all the work of *computing* the continued fraction *and* using \cfrac from amsmath to typeset it.

A *generalized* continued fraction has the same structure but the numerators are not restricted to be 1, and numbers used in the continued fraction may be arbitrary, also fractions, irrationals, complex, indeterminates.[77] The *centered* continued fraction is an example:

```
\[ \xintTeXFrac {915286/188421}=\xintGCFrac {5+-1/7+1/39+-1/53+-1/13}
  =\xintCFrac {915286/188421}\]
```

$$\frac{915286}{188421} = 5 - \cfrac{1}{7 + \cfrac{1}{39 - \cfrac{1}{53 - \cfrac{1}{13}}}} = 4 + \cfrac{1}{1 + \cfrac{1}{6 + \cfrac{1}{38 + \cfrac{1}{1 + \cfrac{1}{51 + \cfrac{1}{1 + \cfrac{1}{12}}}}}}}$$

The macro \xintGCFrac, contrarily to \xintCFrac, does not compute anything, it just typesets starting from a generalized continued fraction in inline format, which in this example was input literally. We also used \xintCFrac for comparison of the two types of continued fractions.

To let TEX compute the centered continued fraction of f there is \xintFtoCC:

```
\[\xintTeXFrac {915286/188421}\to\xintFtoCC {915286/188421}\]
```

$$\frac{915286}{188421} \longrightarrow 5 + -1/7 + 1/39 + -1/53 + -1/13$$

The package macros are expandable and may be nested (naturally \xintCFrac and \xintGCFrac must be at the top level, as they deal with typesetting).

```
\[\xintGCFrac {\xintFtoCC{915286/188421}}\]
```

$$5 - \cfrac{1}{7 + \cfrac{1}{39 - \cfrac{1}{53 - \cfrac{1}{13}}}}$$

The `inline' format expected on input by \xintGCFrac is

$a_0 + b_0/a_1 + b_1/a_2 + b_2/a_3 + \cdots + b_{n-2}/a_{n-1} + b_{n-1}/a_n$

Fractions among the coefficients are allowed but they must be enclosed within braces. Signed integers may be left without braces (but the + signs are mandatory). No spaces are allowed around the plus and fraction symbols. The coefficients may themselves be macros, as long as these macros are *f*-expandable.

```
\[ \xintTeXFrac{\xintGCtoF {1+-1/57+\xintPow {-3}{7}/\xintiiQuo {132}{25}}}
    = \xintGCFrac {1+-1/57+\xintPow {-3}{7}/\xintiiQuo {132}{25}}\]
```

[77] xintcfrac may be used with indeterminates, for basic conversions from one inline format to another, but not for actual computations. See \xintGGCFrac.

$$\frac{1907}{1902} = 1 - \cfrac{1}{57 - \cfrac{2187}{5}}$$

To compute the actual fraction one has \xintGCtoF:

```
\[\xintTeXFrac{\xintGCtoF {1+-1/57+\xintPow {-3}{7}/\xintiiQuo {132}{25}}}\]
```

$$\frac{1907}{1902}$$

For non-numeric input there is \xintGGCFrac.

```
\[\xintGGCFrac {a_0+b_0/a_1+b_1/a_2+b_2/\ddots+\ddots/a_{n-1}+b_{n-1}/a_n}\]
```

$$a_0 + \cfrac{b_0}{a_1 + \cfrac{b_1}{a_2 + \cfrac{b_2}{\ddots + \cfrac{\ddots}{a_{n-1} + \cfrac{b_{n-1}}{a_n}}}}}$$

For regular continued fractions, there is a simpler comma separated format:

```
\[-7,6,19,1,33\to\xintTeXFrac{\xintCstoF{-7,6,19,1,33}}=
                  \xintCFrac{\xintCstoF{-7,6,19,1,33}}\]
```

$$-7, 6, 19, 1, 33 \to \frac{-28077}{4108} = -7 + \cfrac{1}{6 + \cfrac{1}{19 + \cfrac{1}{1 + \cfrac{1}{33}}}}$$

The macro \xintFtoCs produces from a fraction f the comma separated list of its coefficients.

```
\[\xintTeXFrac{1084483/398959}=[\xintFtoCs{1084483/398959}]\]
```

$$\frac{1084483}{398959} = [2, 1, 2, 1, 1, 4, 1, 1, 6, 1, 1, 8, 1, 1, 10, 2]$$

If one prefers other separators, one can use the two arguments macros \xintFtoCx whose first argument is the separator (which may consist of more than one token) which is to be used.

```
\[\xintTeXFrac{2721/1001}=\xintFtoCx {+1/(}{2721/1001})\cdots)\]
```

$$\frac{2721}{1001} = 2 + 1/(1 + 1/(2 + 1/(1 + 1/(1 + 1/(4 + 1/(1 + 1/(1 + 1/(6 + 1/(2) \cdots)$$

This allows under Plain TEX with amstex to obtain the same effect as with LATEX+\amsmath+\xintCFrac:

```
$$\xintTeXOver{2721/1001}=\xintFtoCx {+\cfrac1\\ }{2721/1001}\endcfrac$$
```

As a shortcut to \xintFtoCx with separator 1+/, there is \xintFtoGC:

```
2721/1001=\xintFtoGC {2721/1001}
```

2721/1001=2+1/1+1/2+1/1+1/1+1/4+1/1+1/1+1/6+1/2 Let us compare in that case with the output of \xintFtoCC:

```
2721/1001=\xintFtoCC {2721/1001}
```

2721/1001=3+-1/4+-1/2+1/5+-1/2+1/7+-1/2 To obtain the coefficients as a sequence of braced numbers, there is \xintFtoC (this is a shortcut for \xintFtoCx {}). This list (sequence) may then be manipulated using the various macros of xinttools such as the non-expandable macro \xintAssignArray or the expandable \xintApply and \xintListWithSep.

Conversely to go from such a sequence of braced coefficients to the corresponding fraction there is \xintCtoF.

The `\printnumber' (subsection 2.8) macro which we use in this document to print long numbers can also be useful on long continued fractions.

```
\printnumber{\xintFtoCC {3503701890635072020435104 9/2442417378861974045581 80}}
```

143+1/2+1/5+-1/4+-1/4+-1/4+-1/3+1/2+1/2+1/6+-1/22+1/2+1/10+-1/5+-1/11+-1/3+1/4+-1/2+1/2+1/4+-1/2+1/23+1/3+1/8+-1/6+-1/9 If we apply \xintGCtoF to this generalized continued fraction, we discover that the original fraction was reducible:

```
\xintGCtoF {143+1/2+...+-1/9}=2897319801297630107/20197107104701740
```

When a generalized continued fraction is built with integers, and numerators are only 1's or -1's, the produced fraction is irreducible. And if we compute it again with the last sub-fraction omitted we get another irreducible fraction related to the bigger one by a Bézout identity. Doing this here we get:

```
\xintGCtoF {143+1/2+...+-1/6}=328124887710626729/2287346221788023
```

and indeed:

$$\begin{vmatrix} 2897319801297630107 & 328124887710626729 \\ 20197107104701740 & 2287346221788023 \end{vmatrix} = 1$$

The various fractions obtained from the truncation of a continued fraction to its initial terms are called the convergents. The macros of xintcfrac such as \xintFtoCv, \xintFtoCCv, and others which compute such convergents, return them as a list of braced items, with no separator (as does \xintFtoC for the partial quotients). Here is an example:

```
\[\xintTeXFrac{915286/188421}\to
  \xintListWithSep{,}{\xintApply\xintTeXFrac{\xintFtoCv{915286/188421}}}\]
```

$$\frac{915286}{188421} \to 4, 5, \frac{34}{7}, \frac{1297}{267}, \frac{1331}{274}, \frac{69178}{14241}, \frac{70509}{14515}, \frac{915286}{188421}$$

```
\[\xintTeXFrac{915286/188421}\to
  \xintListWithSep{,}{\xintApply\xintTeXFrac{\xintFtoCCv{915286/188421}}}\]
```

$$\frac{915286}{188421} \to 5, \frac{34}{7}, \frac{1331}{274}, \frac{70509}{14515}, \frac{915286}{188421}$$

We thus see that the `centered convergents' obtained with \xintFtoCCv are among the fuller list of convergents as returned by \xintFtoCv.

Here is a more complicated use of \xintApply and \xintListWithSep. We first define a macro which will be applied to each convergent:

```
\newcommand{\mymacro}[1]{$\xintTeXFrac{#1}=[\xintFtoCs{#1}]$\vtop to 6pt{}}
```

Next, we use the following code:

```
$\xintTeXFrac{49171/18089}\to{}$
\xintListWithSep {, }{\xintApply{\mymacro}{\xintFtoCv{49171/18089}}}
```

It produces:

$\frac{49171}{18089} \to 2 = [2]$, $3 = [3]$, $\frac{8}{3} = [2, 1, 2]$, $\frac{11}{4} = [2, 1, 3]$, $\frac{19}{7} = [2, 1, 2, 2]$, $\frac{87}{32} = [2, 1, 2, 1, 1, 4]$, $\frac{106}{39} = [2, 1, 2, 1, 1, 5]$, $\frac{193}{71} = [2, 1, 2, 1, 1, 4, 2]$, $\frac{1264}{465} = [2, 1, 2, 1, 1, 4, 1, 1, 6]$, $\frac{1457}{536} = [2, 1, 2, 1, 1, 4, 1, 1, 7]$, $\frac{2721}{1001} = [2, 1, 2, 1, 1, 4, 1, 1, 6, 2]$, $\frac{23225}{8544} = [2, 1, 2, 1, 1, 4, 1, 1, 6, 1, 1, 8]$, $\frac{49171}{18089} = [2, 1, 2, 1, 1, 4, 1, 1, 6, 1, 1, 8, 2]$

The macro \xintCntoF allows to specify the coefficients as a function given by a one-parameter macro. The produced values do not have to be integers.

```
\def\cn #1{\xintiiPow {2}{#1}}% 2^n
  \[\xintTeXFrac{\xintCntoF {6}{\cn}}=\xintCFrac [l]{\xintCntoF {6}{\cn}}\]
```

$$\frac{3541373}{2449193} = 1 + \cfrac{1}{2 + \cfrac{1}{4 + \cfrac{1}{8 + \cfrac{1}{16 + \cfrac{1}{32 + \cfrac{1}{64}}}}}}$$

Notice the use of the optional argument [l] to \xintCFrac. Other possibilities are [r] and (default) [c].

```
\def\cn #1{\xintPow {2}{-#1}}%
  \[\xintTeXFrac{\xintCntoF {6}{\cn}}=\xintGCFrac [r]{\xintCntoGC {6}{\cn}}=
     [\xintFtoCs {\xintCntoF {6}{\cn}}]\]
```

$$\frac{3159019}{2465449} = 1 + \cfrac{1}{\frac{1}{2} + \cfrac{1}{\frac{1}{4} + \cfrac{1}{\frac{1}{8} + \cfrac{1}{\frac{1}{16} + \cfrac{1}{\frac{1}{32} + \frac{1}{\frac{1}{64}}}}}}} = [1, 3, 1, 1, 4, 14, 1, 1, 1, 1, 79, 2, 1, 1, 2]$$

We used \xintCntoGC as we wanted to display also the continued fraction and not only the fraction returned by \xintCntoF.

There are also \xintGCntoF and \xintGCntoGC which allow the same for generalized fractions. An initial portion of a generalized continued fraction for π is obtained like this

```
\def\an #1{\the\numexpr 2*#1+1\relax }%
\def\bn #1{\the\numexpr (#1+1)*(#1+1)\relax }%
\[\xintTeXFrac{\xintDiv {4}{\xintGCntoF {5}{\an}{\bn}}} =
       \cfrac{4}{\xintGCFrac{\xintGCntoGC {5}{\an}{\bn}}} =
                \xintTrunc {10}{\xintDiv {4}{\xintGCntoF {5}{\an}{\bn}}}\dots\]
```

$$\frac{92736}{29520} = \cfrac{4}{1 + \cfrac{1}{3 + \cfrac{4}{5 + \cfrac{9}{7 + \cfrac{16}{9 + \frac{25}{11}}}}}} = 3.1414634146\ldots$$

We see that the quality of approximation is not fantastic compared to the simple continued fraction of π with about as many terms:

```
\[\xintTeXFrac{\xintCstoF{3,7,15,1,292,1,1}}=
             \xintGCFrac{3+1/7+1/15+1/1+1/292+1/1+1/1}=
                      \xintTrunc{10}{\xintCstoF{3,7,15,1,292,1,1}}\dots\]
```

$$\frac{208341}{66317} = 3 + \cfrac{1}{7 + \cfrac{1}{15 + \cfrac{1}{1 + \cfrac{1}{292 + \cfrac{1}{1 + \frac{1}{1}}}}}} = 3.1415926534\ldots$$

When studying the continued fraction of some real number, there is always some doubt about how many terms are valid, when computed starting from some approximation. If $f \leqslant x \leqslant g$ and f, g both have the same first K partial quotients, then x also has the same first K quotients and convergents. The macro \xintFGtoC outputs as a sequence of braced items the common partial quotients of its two arguments. We can thus use it to produce a sure list of valid convergents of π for example, starting from some proven lower and upper bound:

```
$$\pi\to [\xintListWithSep{,}
```

```
          {\xintFGtoC {3.14159265358979323}{3.14159265358979324}}, \dots]$$
  \noindent$\pi\to\xintListWithSep{,\allowbreak\;}
     {\xintApply{\xintTeXFrac}
     {\xintCtoCv{\xintFGtoC {3.14159265358979323}{3.14159265358979324}}}}, \dots$
```

$$\pi \longrightarrow [3, 7, 15, 1, 292, 1, 1, 1, 2, 1, 3, 1, 14, 2, 1, 1, \dots]$$

$$\pi \longrightarrow 3, \frac{22}{7}, \frac{333}{106}, \frac{355}{113}, \frac{103993}{33102}, \frac{104348}{33215}, \frac{208341}{66317}, \frac{312689}{99532}, \frac{833719}{265381}, \frac{1146408}{364913}, \frac{4272943}{1360120}, \frac{5419351}{1725033}, \frac{80143857}{25510582}, \frac{165707065}{52746197}, \frac{245850922}{78256779}, \frac{411557987}{131002976}, \dots$$

code

17.3. \xintCFrac

Frac
f

\xintCFrac{f} is a math-mode only, LaTeX with amsmath only, macro which first computes then displays with the help of \cfrac the simple continued fraction corresponding to the given fraction. It admits an optional argument which may be [l], [r] or (the default) [c] to specify the location of the one's in the numerators of the sub-fractions. This macro is *f*-expandable in the sense that it prepares expandably the whole expression with the multiple \cfrac's, but it is not completely expandable naturally as \cfrac isn't.

code

17.4. \xintGCFrac

f \xintGCFrac{a+b/c+d/e+f/g+h/...+x/y} uses similarly \cfrac to prepare the typesetting with the amsmath \cfrac (LaTeX) of a generalized continued fraction given in inline format (or as macro which will *f*-expand to it). It admits the same optional argument as \xintCFrac. Plain TeX with amstex users, see \xintGCtoGCx.

```
\[\xintGCFrac {1+\xintPow{1.5}{3}/{1/7}+{-3/5}/\xintiiFac {6}}\]
```

$$1+\cfrac{3375\cdot 10^{-3}}{\frac{1}{7}-\cfrac{\frac{3}{5}}{720}}$$

This is mostly a typesetting macro, although it does trigger the expansion of the coefficients. See \xintGCtoF if you are impatient to see this specific fraction computed.

It admits an optional argument within square brackets which may be either [l], [c] or [r]. Default is [c] (numerators are centered).

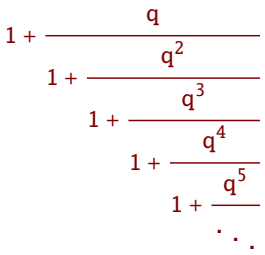
Numerators and denominators are made arguments to the \xintTeXFrac macro. This allows them to be themselves fractions or anything *f*-expandable giving numbers or fractions, but also means however that they can not be arbitrary material, they can not contain color changing macros for example. One of the reasons is that \xintGCFrac tries to determine the signs of the numerators and chooses accordingly to use + or -.

code

17.5. \xintGGCFrac

f \xintGGCFrac{a+b/c+d/e+f/g+h/...+x/y} is a clone of \xintGCFrac, hence again LaTeX specific with package amsmath. It does not assume the coefficients to be numbers as understood by xintfrac. The macro can be used for displaying arbitrary content as a continued fraction with \cfrac, using only plus signs though. Note though that it will first *f*-expand its argument, which may be thus be one of the xintcfrac macros producing a (general) continued fraction in inline format, see \xintFtoCx for an example. If this expansion is not wished, it is enough to start the argument with a space.

```
\[\xintGGCFrac {1+q/1+q^2/1+q^3/1+q^4/1+q^5/\ddots}\]
```



code
17.6. \xintGCtoGCx

nnf ⋆ \xintGCtoGCx{sepa}{sepb}{a+b/c+d/e+f/...+x/y} returns the list of the coefficients of the generalized continued fraction of f, each one within a pair of braces, and separated with the help of sepa and sepb. Thus

```
\xintGCtoGCx :;{1+2/3+4/5+6/7} gives 1:2;3:4;5:6;7
```

The following can be used byt Plain TEX+amstex users to obtain an output similar as the ones produced by \xintGCFrac and \xintGGCFrac:

```
$$\xintGCtoGCx {+\cfrac}{\\}{a+b/...}\endcfrac$$
$$\xintGCtoGCx {+\cfrac\xintTeXOver}{\\\xintTeXOver}{a+b/...}\endcfrac$$
```

code
17.7. \xintFtoC

Frac f ⋆ \xintFtoC{f} computes the coefficients of the simple continued fraction of f and returns them as a list (sequence) of braced items.

```
\fdef\test{\xintFtoC{-5262046/89233}}\texttt{\meaning\test}
```

macro:->{-59}{33}{27}{100}

code
17.8. \xintFtoCs

Frac f ⋆ \xintFtoCs{f} returns the comma separated list of the coefficients of the simple continued fraction of f. Notice that starting with 1.09m a space follows each comma (mainly for usage in text mode, as in math mode spaces are produced in the typeset output by TEX itself).

```
\[ \xintTeXsignedFrac{-5262046/89233} \to [\xintFtoCs{-5262046/89233}]\]
```

$$-\frac{5262046}{89233} \to [-59, 33, 27, 100]$$

code
17.9. \xintFtoCx

Frac n f ⋆ \xintFtoCx{sep}{f} returns the list of the coefficients of the simple continued fraction of f separated with the help of sep, which may be anything (and is kept unexpanded). For example, with Plain TEX and amstex,

```
$$\xintFtoCx {+\cfrac1\\ }{-5262046/89233}\endcfrac$$
```

will display the continued fraction using \cfrac. Each coefficient is inside a brace pair { }, allowing a macro to end the separator and fetch it as argument, for example, again with Plain TEX and amstex:

```
   \def\highlight #1{\ifnum #1>200 \textcolor{red}{#1}\else #1\fi}
   $$\xintFtoCx {+\cfrac1\\ \highlight}{104348/33215}\endcfrac$$
```

Due to the different and extremely cumbersome syntax of \cfrac under LaTeX it proves a bit tortuous to obtain there the same effect. Actually, it is partly for this purpose that 1.09m added \xintGGCFrac. We thus use \xintFtoCx with a suitable separator, and then the whole thing as argument to \xintGGCFrac:

```
\def\highlight #1{\ifnum #1>200 \fcolorbox{blue}{white}{\boldmath\color{red}$#1$}%
    \else #1\fi}
\[\xintGGCFrac {\xintFtoCx {+1/\highlight}{208341/66317}}\]
```

$$3+\cfrac{1}{7+\cfrac{1}{15+\cfrac{1}{1+\cfrac{1}{\boxed{292}+\frac{1}{2}}}}}$$

code
17.10. \xintFtoGC

Frac f ★ \xintFtoGC{f} does the same as \xintFtoCx{+1/}{f}. Its output may thus be used in the package macros expecting such an `inline format'.

```
  566827/208524=\xintFtoGC {566827/208524}
566827/208524=2+1/1+1/2+1/1+1/1+1/4+1/1+1/1+1/6+1/1+1/1+1/8+1/1+1/1+1/11
```

code
17.11. \xintFGtoC

Frac Frac f f ★ \xintFGtoC{f}{g} computes the common initial coefficients to two given fractions f and g. Notice that any real number f<x<g or f>x>g will then necessarily share with f and g these common initial coefficients for its regular continued fraction. The coefficients are output as a sequence of braced numbers. This list can then be manipulated via macros from xinttools, or other macros of xintcfrac.

```
  \fdef\test{\xintFGtoC{-5262046/89233}{-5314647/90125}}\texttt{\meaning\test}
macro:->{-59}{33}{27}
  \fdef\test{\xintFGtoC{3.141592653}{3.141592654}}\texttt{\meaning\test}
macro:->{3}{7}{15}{1}
  \fdef\test{\xintFGtoC{3.1415926535897932384}{3.1415926535897932385}}\meaning\test
macro:->{3}{7}{15}{1}{292}{1}{1}{1}{2}{1}{3}{1}{14}{2}{1}{1}{2}{2}{2}
  \xintRound {30}{\xintCstoF{\xintListWithSep{,}{\test}}}
3.14159265358979323838637750639 0
  \xintRound {30}{\xintCtoF{\test}}
3.14159265358979323838637750639 0
  \fdef\test{\xintFGtoC{1.41421356237309}{1.4142135623731}}\meaning\test
macro:->{1}{2}{2}{2}{2}{2}{2}{2}{2}{2}{2}{2}{2}{2}{2}{2}{2}{2}{2}{2}{2}{2}{2}{2}
```

code
17.12. \xintFtoCC

Frac f ★ \xintFtoCC{f} returns the `centered' continued fraction of f, in `inline format'.

```
  566827/208524=\xintFtoCC {566827/208524}
566827/208524=3+-1/4+-1/2+1/5+-1/2+1/7+-1/2+1/9+-1/2+1/11
  \[\xintTeXFrac{566827/208524} = \xintGCFrac{\xintFtoCC{566827/208524}}\]
```

$$\frac{566827}{208524} = 3 - \cfrac{1}{4 - \cfrac{1}{2 + \cfrac{1}{5 - \cfrac{1}{2 + \cfrac{1}{7 - \cfrac{1}{2 + \cfrac{1}{9 - \cfrac{1}{2 + \cfrac{1}{11}}}}}}}}}$$

17.13. \xintCstoF

code

f ★ \xintCstoF{a,b,c,d,...,z} computes the fraction corresponding to the coefficients, which may be fractions or even macros expanding to such fractions. The final fraction may then be highly reducible.

Starting with release 1.09m spaces before commas are allowed and trimmed automatically (spaces after commas were already silently handled in earlier releases).

```
\[\xintGCFrac {-1+1/3+1/-5+1/7+1/-9+1/11+1/-13}=
  \xintTeXsignedFrac{\xintCstoF {-1,3,-5,7,-9,11,-13}}=\xintTeXsignedFrac{\xintGCtoF
    {-1+1/3+1/-5+1/7+1/-9+1/11+1/-13}}\]
```

$$-1 + \cfrac{1}{3 + \cfrac{1}{-5 + \cfrac{1}{7 + \cfrac{1}{-9 + \cfrac{1}{11 + \cfrac{1}{-13}}}}}} = -\frac{75887}{118187} = -\frac{75887}{118187}$$

```
\[\xintGCFrac{{1/2}+1/{1/3}+1/{1/4}+1/{1/5}}=
                                \xintTeXFrac{\xintCstoF {1/2,1/3,1/4,1/5}}\]
```

$$\frac{1}{2} + \cfrac{1}{\frac{1}{3} + \cfrac{1}{\frac{1}{4} + \frac{1}{\frac{1}{5}}}} = \frac{159}{66}$$

A generalized continued fraction may produce a reducible fraction (\xintCstoF tries its best not to accumulate in a silly way superfluous factors but will not do simplifications which would be obvious to a human, like simplification by 3 in the result above).

17.14. \xintCtoF

code

f ★ \xintCtoF{{a}{b}{c}...{z}} computes the fraction corresponding to the coefficients, which may be fractions or even macros.

```
  \xintCtoF {\xintApply {\xintiiPow 3}{\xintSeq {1}{5}}}
14946960/4805083
  \[ \xintTeXFrac{14946960/4805083}=\xintCFrac {14946960/4805083}\]
```

$$\frac{14946960}{4805083} = 3 + \cfrac{1}{9 + \cfrac{1}{27 + \cfrac{1}{81 + \cfrac{1}{243}}}}$$

In the example above the power of 3 was already pre-computed via the expansion done by \xintApply, but if we try with \xintApply { \xintiiPow 3} where the space will stop this expansion, we can check that \xintCtoF will itself provoke the needed coefficient expansion.

17.15. \xintGCtoF

code

f ⋆ \xintGCtoF{a+b/c+d/e+f/g+.....+v/w+x/y} computes the fraction defined by the inline generalized continued fraction. Coefficients may be fractions but must then be put within braces. They can be macros. The plus signs are mandatory.

```
\[\xintGCFrac {1+\xintPow{1.5}{3}/{1/7}+{-3/5}/\xintiiFac {6}} =
\xintTeXFrac{\xintGCtoF {1+\xintPow{1.5}{3}/{1/7}+{-3/5}/\xintiiFac {6}}} =
\xintTeXFrac{\xintIrr{\xintGCtoF
                  {1+\xintPow{1.5}{3}/{1/7}+{-3/5}/\xintiiFac {6}}}}\]
```

$$1 + \cfrac{3375 \cdot 10^{-3}}{\frac{1}{7} - \cfrac{\frac{3}{5}}{720}} = \frac{88629000}{3579000} = \frac{29543}{1193}$$

```
\[ \xintGCFrac{{1/2}+{2/3}/{4/5}+{1/2}/{1/5}+{3/2}/{5/3}} =
   \xintTeXFrac{\xintGCtoF {{1/2}+{2/3}/{4/5}+{1/2}/{1/5}+{3/2}/{5/3}}} \]
```

$$\frac{1}{2} + \cfrac{\frac{2}{3}}{\frac{4}{5} + \cfrac{\frac{1}{2}}{\frac{1}{5} + \cfrac{\frac{3}{2}}{\frac{5}{3}}}} = \frac{4270}{4140}$$

The macro tries its best not to accumulate superfluous factor in the denominators, but doesn't reduce the fraction to irreducible form before returning it and does not do simplifications which would be obvious to a human.

17.16. \xintCstoCv

code

f ⋆ \xintCstoCv{a,b,c,d,...,z} returns the sequence of the corresponding convergents, each one within braces.

It is allowed to use fractions as coefficients (the computed convergents have then no reason to be the real convergents of the final fraction). When the coefficients are integers, the convergents are irreducible fractions, but otherwise it is not necessarily the case.

```
\xintListWithSep:{\xintCstoCv{1,2,3,4,5,6}}
```

1/1:3/2:10/7:43/30:225/157:1393/972

```
\xintListWithSep:{\xintCstoCv{1,1/2,1/3,1/4,1/5,1/6}}
```

1/1:3/1:9/7:45/19:225/159:1575/729

```
\[\xintListWithSep{\to}{\xintApply\xintTeXFrac{\xintCstoCv {\xintPow
        {-.3}{-5},7.3/4.57,\xintCstoF{3/4,9,-1/3}}}}\]
```

$$\frac{-100000}{243} \to \frac{-72888949}{177390} \to \frac{-2700356878}{6567804}$$

code
17.17. \xintCtoCv

f ★ \xintCtoCv{{a}{b}{c}...{z}} returns the sequence of the corresponding convergents, each one within braces.

```
\fdef\test{\xintCtoCv {11111111111}}\texttt{\meaning\test}
```

macro:->{1/1}{2/1}{3/2}{5/3}{8/5}{13/8}{21/13}{34/21}{55/34}{89/55}{144/89}

code
17.18. \xintGCtoCv

f ★ \xintGCtoCv{a+b/c+d/e+f/g+.....+v/w+x/y} returns the list of the corresponding convergents. The coefficients may be fractions, but must then be inside braces. Or they may be macros, too.

The convergents will in the general case be reducible. To put them into irreducible form, one needs one more step, for example it can be done with \xintApply\xintIrr.

```
\[\xintListWithSep{,}{\xintApply\xintTeXFrac
                {\xintGCtoCv{3+{-2}/{7/2}+{3/4}/12+{-56}/3}}}\]
\[\xintListWithSep{,}{\xintApply\xintTeXFrac{\xintApply\xintIrr
                {\xintGCtoCv{3+{-2}/{7/2}+{3/4}/12+{-56}/3}}}}\]
```

$$3, \frac{17}{7}, \frac{834}{342}, \frac{1306}{542}$$

$$3, \frac{17}{7}, \frac{139}{57}, \frac{653}{271}$$

code
17.19. \xintFtoCv

Frac f ★ \xintFtoCv{f} returns the list of the (braced) convergents of f, with no separator. To be treated with \xintAssignArray or \xintListWithSep.

```
\[\xintListWithSep{\to}{\xintApply\xintTeXFrac{\xintFtoCv{5211/3748}}}\]
```

$$1 \to \frac{3}{2} \to \frac{4}{3} \to \frac{7}{5} \to \frac{25}{18} \to \frac{32}{23} \to \frac{57}{41} \to \frac{317}{228} \to \frac{374}{269} \to \frac{691}{497} \to \frac{5211}{3748}$$

code
17.20. \xintFtoCCv

Frac f ★ \xintFtoCCv{f} returns the list of the (braced) centered convergents of f, with no separator. To be treated with \xintAssignArray or \xintListWithSep.

```
\[\xintListWithSep{\to}{\xintApply\xintTeXFrac{\xintFtoCCv{5211/3748}}}\]
```

$$1 \to \frac{4}{3} \to \frac{7}{5} \to \frac{32}{23} \to \frac{57}{41} \to \frac{374}{269} \to \frac{691}{497} \to \frac{5211}{3748}$$

code
17.21. \xintCntoF

num x f ★ \xintCntoF{N}{\macro} computes the fraction f having coefficients c(j)=\macro{j} for j=0,1,...,
↪ N. The N parameter is given to a \numexpr. The values of the coefficients, as returned by \macro do not have to be positive, nor integers, and it is thus not necessarily the case that the original c(j) are the true coefficients of the final f.

```
\def\macro #1{\the\numexpr 1+#1*#1\relax} \xintCntoF {5}{\macro}
```

72625/49902[0]

This example shows that the fraction is output with a trailing number in square brackets (representing a power of ten), this is for consistency with what do most macros of xintfrac, and does not have to be always this annoying [0] as the coefficients may for example be numbers in scientific notation. To avoid these trailing square brackets, for example if the coefficients are known to be integers, there is always the possibility to filter the output via \xintPRaw, or \xintIrr (the latter is overkill in the case of integer coefficients, as the fraction is guaranteed to be irreducible then).

17.22. \xintGCntoF

code

num x f f ★ \xintGCntoF{N}{\macroA}{\macroB} returns the fraction f corresponding to the inline generalized continued fraction a0+b0/a1+b1/a2+....+b(N-1)/aN, with a(j)=\macroA{j} and b(j)=\macroB{j}. The N parameter is given to a \numexpr.

```
\def\coeffA #1{\the\numexpr #1+4-3*((#1+2)/3)\relax }%
\def\coeffB #1{\the\numexpr \ifodd #1 -\fi 1\relax }% (-1)^n
\[\xintGCFrac{\xintGCntoGC {6}{\coeffA}{\coeffB}} =
                                \xintTeXFrac{\xintGCntoF {6}{\coeffA}{\coeffB}}\]
```

$$1+\cfrac{1}{2-\cfrac{1}{3+\cfrac{1}{1-\cfrac{1}{2+\cfrac{1}{3-\cfrac{1}{1}}}}}} = \frac{39}{25}$$

There is also \xintGCntoGC to get the `inline format' continued fraction.

17.23. \xintCntoCs

code

num x f ★ \xintCntoCs{N}{\macro} produces the comma separated list of the corresponding coefficients, from n=0 to n=N. The N is given to a \numexpr.

```
\xintCntoCs {5}{\macro}
```

1, 2, 5, 10, 17, 26

```
\[ \xintTeXFrac{\xintCntoF{5}{\macro}}=\xintCFrac{\xintCntoF {5}{\macro}}\]
```

$$\frac{72625}{49902} = 1+\cfrac{1}{2+\cfrac{1}{5+\cfrac{1}{10+\cfrac{1}{17+\cfrac{1}{26}}}}}$$

17.24. \xintCntoGC

code

num x f ★ \xintCntoGC{N}{\macro} evaluates the c(j)=\macro{j} from j=0 to j=N and returns a continued fraction written in inline format: {c(0)}+1/{c(1)}+1/...+1/{c(N)}. The parameter N is given to a \numexpr. The coefficients, after expansion, are, as shown, being enclosed in an added pair of braces, they may thus be fractions.

```
\def\macro #1{\the\numexpr\ifodd#1 -1-#1\else1+#1\fi\relax/\the\numexpr 1+#1*#1\relax}
\fdef\x{\xintCntoGC {5}{\macro}}\meaning\x
\[\xintGCFrac{\xintCntoGC {5}{\macro}}\]
```

```
macro:->{1/\the \numexpr 1+0*0\relax }+1/{-2/\the \numexpr 1+1*1\relax }+1/{3/\the \numexpr 1+2*2\relax }+1/{-4/\the \numexpr 1+3*3\relax }+1/{5/\the \numexpr 1+4*4\relax }+1/{-6/\the \numexpr 1+5*5\relax }
```

$$1+\cfrac{1}{\frac{-2}{2}+\cfrac{1}{\frac{3}{5}+\cfrac{1}{\frac{-4}{10}+\cfrac{1}{\frac{5}{17}+\frac{1}{\frac{-6}{26}}}}}}$$

17.25. \xintGCntoGC

code

num x f f ⋆ \xintGCntoGC{N}{\macroA}{\macroB} evaluates the coefficients and then returns the corresponding {a0}+{b0}/{a1}+{b1}/{a2}+...+{b(N-1)}/{aN} inline generalized fraction. N is givent to a \numexpr. The coefficients are enclosed into pairs of braces, and may thus be fractions, the fraction slash will not be confused in further processing by the continued fraction slashes.

```
\def\an #1{\the\numexpr  #1*#1*#1+1\relax}%
\def\bn #1{\the\numexpr \ifodd#1 -\fi 1*(#1+1)\relax}%
$\xintGCntoGC {5}{\an}{\bn}=\xintGCFrac {\xintGCntoGC {5}{\an}{\bn}} =
\displaystyle\xintTeXFrac {\xintGCntoF {5}{\an}{\bn}}$\par
```

$$1+1/2+-2/9+3/28+-4/65+5/126 = 1+\cfrac{1}{2-\cfrac{2}{9+\cfrac{3}{28-\cfrac{4}{65+\frac{5}{126}}}}} = \frac{5797655}{3712466}$$

17.26. \xintCstoGC

code

f ⋆ \xintCstoGC{a,b,..,z} transforms a comma separated list (or something expanding to such a list) into an `inline format' continued fraction {a}+1/{b}+1/...+1/{z}. The coefficients are just copied and put within braces, without expansion. The output can then be used in \xintGCFrac for example.

```
\[\xintGCFrac {\xintCstoGC {-1,1/2,-1/3,1/4,-1/5}}=\xintTeXsignedFrac{\xintCstoF {-1,1/2,-1/3,1/4,-1/5}}\]
```

$$-1+\cfrac{1}{\frac{1}{2}+\cfrac{1}{\frac{-1}{3}+\cfrac{1}{\frac{1}{4}+\frac{1}{\frac{-1}{5}}}}} = -\frac{145}{83}$$

17.27. \xintiCstoF, \xintiGCtoF, \xintiCstoCv, \xintiGCtoCv

f ⋆ Essentially the same as the corresponding macros without the `i', but for integer-only input. Infinitesimally faster, mainly for internal use by the package.

17.28. \xintGCtoGC

f ⋆ \xintGCtoGC{a+b/c+d/e+f/g+.....+v/w+x/y} expands (with the usual meaning) each one of the coefficients and returns an inline continued fraction of the same type, each expanded coefficient being enclosed within braces.

```
\fdef\x {\xintGCtoGC {1+\xintPow{1.5}{3}/{1/7}+{-3/5}/%
                        \xintiiFac {6}+\xintCstoF {2,-7,-5}/16}} \meaning\x
macro:->{1}+{3375/1[-3]}/{1/7}+{-3/5}/{720}+{67/36}/{16}
```

To be honest I have forgotten for which purpose I wrote this macro in the first place.

17.29. Euler's number e

Let us explore the convergents of Euler's number e. The volume of computation is kept minimal by the following steps:

- a comma separated list of the first 36 coefficients is produced by \xintCntoCs,
- this is then given to \xintiCstoCv which produces the list of the convergents (there is also \xintCstoCv, but our coefficients being integers we used the infinitesimally faster \xintiCstoCv),
- then the whole list was converted into a sequence of one-line paragraphs, each convergent becomes the argument to a macro printing it together with its decimal expansion with 30 digits after the decimal point.
- A count register \cnta was used to give a line count serving as a visual aid: we could also have done that in an expandable way, but well, let's relax from time to time...

```
\def\cn #1{\the\numexpr\ifcase \numexpr #1+3-3*((#1+2)/3)\relax
                           1\or1\or2*(#1/3)\fi\relax }
% produces the pattern 1,1,2,1,1,4,1,1,6,1,1,8,... which are the
% coefficients of the simple continued fraction of e-1.
\cnta 0
\def\mymacro #1{\advance\cnta by 1
                \noindent
                \hbox to 3em {\hfil\small\dtt{\the\cnta.} }%
                $\xintTrunc {30}{\xintAdd {1[0]}{#1}}\dots=
                 \xintTeXFrac{\xintAdd {1[0]}{#1}}$}%
\xintListWithSep{\vtop to 6pt{}\vbox to 12pt{}\par}
    {\xintApply\mymacro{\xintiCstoCv{\xintCntoCs {35}{\cn}}}}
```

1. $2.000000000000000000000000000000\cdots = 2$
2. $3.000000000000000000000000000000\cdots = 3$
3. $2.666666666666666666666666666666\cdots = \frac{8}{3}$
4. $2.750000000000000000000000000000\cdots = \frac{11}{4}$
5. $2.714285714285714285714285714285\cdots = \frac{19}{7}$
6. $2.718750000000000000000000000000\cdots = \frac{87}{32}$

7. $2.71794871794871794871794871794\cdots = \frac{106}{39}$
8. $2.71830985915492957746478873239 4\cdots = \frac{193}{71}$
9. $2.71827956989247311827956989247 3\cdots = \frac{1264}{465}$
10. $2.71828358208955223880597014925 3\cdots = \frac{1457}{536}$
11. $2.71828171828171828171828171828 1\cdots = \frac{2721}{1001}$
12. $2.71828183520599250936329588014 9\cdots = \frac{23225}{8544}$
13. $2.71828182294394971189104243059 1\cdots = \frac{25946}{9545}$
14. $2.71828182873569572668472552379 8\cdots = \frac{49171}{18089}$
15. $2.71828182844540131803502507417 2\cdots = \frac{517656}{190435}$
16. $2.71828182847058372177782893096 2\cdots = \frac{566827}{208524}$
17. $2.71828182845856341127785060620 2\cdots = \frac{1084483}{398959}$
18. $2.71828182845906511407452954664 8\cdots = \frac{13580623}{4996032}$
19. $2.71828182845902801320706559102 6\cdots = \frac{14665106}{5394991}$
20. $2.71828182845904585140462108494 9\cdots = \frac{28245729}{10391023}$
21. $2.71828182845904521352198375822 1\cdots = \frac{410105312}{150869313}$
22. $2.71828182845904525462479502709 2\cdots = \frac{438351041}{161260336}$
23. $2.71828182845904523475756063147 9\cdots = \frac{848456353}{312129649}$
24. $2.71828182845904523537901337277 2\cdots = \frac{14013652689}{5155334720}$
25. $2.71828182845904523534353553278 7\cdots = \frac{14862109042}{5467464369}$
26. $2.71828182845904523536075323018 8\cdots = \frac{28875761731}{10622799089}$
27. $2.71828182845904523536027459394 1\cdots = \frac{534625820200}{196677847971}$
28. $2.71828182845904523536029912091 1\cdots = \frac{563501581931}{207300647060}$
29. $2.71828182845904523536028717990 0\cdots = \frac{1098127402131}{403978495031}$
30. $2.71828182845904523536028747861 1\cdots = \frac{22526049624551}{8286870547680}$
31. $2.71828182845904523536028746472 6\cdots = \frac{23624177026682}{8690849042711}$
32. $2.71828182845904523536028747150 3\cdots = \frac{46150226651233}{16977719590391}$
33. $2.71828182845904523536028747134 9\cdots = \frac{1038929163353808}{382200680031313}$
34. $2.71828182845904523536028747135 5\cdots = \frac{1085079390005041}{399178399621704}$
35. $2.71828182845904523536028747135 2\cdots = \frac{2124008553358849}{781379079653017}$
36. $2.71828182845904523536028747135 2\cdots = \frac{52061284670617417}{19152276311294112}$

One can with no problem compute much bigger convergents. Let's get the 200th convergent. It turns out to have the same first 268 digits after the decimal point as e-1. Higher convergents get more and more digits in proportion to their index: the 500th convergent already gets 799 digits correct! To allow speedy compilation of the source of this document when the need arises, I limit here to the 200th convergent.

```
\fdef\z {\xintCntoF {199}{\cn}}%
```

```
\begingroup\parindent 0pt \leftskip 2.5cm
\indent\llap {Numerator = }\printnumber{\xintNumerator\z}\par
\indent\llap {Denominator = }\printnumber{\xintDenominator\z}\par
\indent\llap {Expansion = }\printnumber{\xintTrunc{268}\z}\dots\par\endgroup
```

```
  Numerator = 568964038871896267597523892315807875293889017667917446057232024547192296961118
            ↪ 230175243860174995310817731367012417086097496343293829 06
Denominator = 331123817669737619306256360816356753365468823729314438156205615463246659728581
            ↪ 865461337692063148916019550614570592553376611426452172 23
  Expansion = 1.718281828459045235360287471352662497757247093699959574966967627724076630353 5
            ↪ 475945713821785251664274274663919320030599218174135966290435729003342952605956
            ↪ 307381323286279434907632338298807531952510190115738341879307021540891499348841
            ↪ 675092447614606680822648001684774118...
```

One can also use a centered continued fraction: we get more digits but there are also more computations as the numerators may be either 1 or -1.

Part III.
The xintexpr and allied packages source code

18 An introduction and a brief timeline . . . 244
19 Package xintkernel implementation . . . 248
20 Package xinttools implementation . . . 272
21 Package xintcore implementation . . . 315
22 Package xint implementation . . . 330
23 Package xintbinhex implementation . . . 353
24 Package xintgcd implementation . . . 374
25 Package xintfrac implementation . . . 385
26 Package xintseries implementation . . . 460
27 Package xintcfrac implementation . . . 470
28 Package xintexpr implementation . . . 493
29 Package calxcore implementation . . . 512
30 Package calxint implementation . . . 573
31 Package calxfp implementation . . . 630
32 Package calx implementation . . . 700
33 Library calxlog implementation . . . 849
34 Library calxlog data tables . . . 928
35 Library calxtrig implementation . . . 958
36 Library calxtrig data tables . . . 1042
37 Cumulative line and macro count . . . 1058

18. An introduction and a brief timeline

This is 2.0a2 of 2026/11/26.

The xintchanges.md file, included in the CTAN upload, contains the complete list of changes relevant to user level since the initial release of the package:

texdoc xintchanges.md

It exists in HTML format at the package web page:

https://jfbu.github.io/xint/CHANGES.html

At 1.4m I added hyperlinks to the macro code. Each instance of a macro in the code is linked with target the location of its definition (via \def or \let or variants). This has been done via a heist on doc (v2 version) automated indexing which has been transformed here into automated hyperlinking. Furthermore the codeline where the macro is defined will link to its description in the user manual part of xint.pdf. In (optional build) xintsource.pdf the link is more modestly targeting the sectioning heading referencing the macro name, if available. In xint.pdf the macros documented in the user manual are marked with a *source* link on top of them, targeting their source code.

The sad truth however is that my code is poorly documented. The comments

- are often too scarce,
- have occasional excessive verbosity,
- and are generally inadequate or irrelevant.

The macro comments have had a distinct tendency to record the changes across releases or even those occurring during pre-release development phase, rather than explaining the interface, or perhaps an algorithm. As I am aware of that, I have a mechanism of "private comments" which are removed by the dtx build script. But then I sometimes use it en masse as it would be too much work to clean-up the existing comments, and as a result the code is not commented at all anymore... A typical example is with \xintiiSquareRoot which is amply documented in the private sources but only 10% of it could be of any value to any other reader than myself and it would be simply the description of what #1, #2, ... stand for. As a result I converted at some point everything into private comments. Extracting the useful parts describing the macro parameters and checking they are actually still valid would be very time-consuming. The real problem here is that the actual underlying algorithms are rarely if ever described.

- Release 2.0a2 of 2026/11/26.
  - The main memory footprint of xintexpr at the default precision of 16 digits has been reduced by a factor of four with respect to 2.0a1. It remains about 2.5 times that of 1.4q.
  - The pre-2.0 semantics of \xintDigits and \xintSetDigits are restored: only their starred variants also update the precision of the scientific libraries. These now compute at a precision rounded up to a multiple of four digits, with possible double rounding. Library reloads respect \xintglobaldefstrue.
  - The commands \xintreloadcalxlog and \xintreloadcalxtrig replace \xintreloadxintlog and \xintreloadxinttrig, retained as aliases. \xintreloadscilibs is functional again.
  - Integer powers in \calxeval and \xintfloateval again have a dedicated implementation based on binary exponentiation, typically almost twice as fast as the pre-2.0 implementation at 16 digits. They are thus normally no longer subject to the 100-digit precision limitation of the scientific libraries.
  - The user manual has received various corrections and clarifications.
- Release 2.0a1 of 2026/11/01. This is a major release in which the internals have been entirely rewritten for speed. A new calx package provides the functionality of \xintfloateval and \xintiieval, also available under the aliases \calxeval and \calxinteval, respectively. The files xintlog.sty and xinttrig.sty are gone, replaced by support files in the calx namespace.

  The calx package adds almost no new user interface. The main novelty is \calxsetup, which can be used to configure the signal handlers. By default, no error condition is trapped in the sense of deliberately raising a low-level TeX error. Instead, such conditions generate NaN, +Infinity, or -Infinity, which propagate through subsequent calculations.

The computations of trigonometric functions, logarithms, exponentials, and powers are now roughly eight times faster.

The legacy interface of xintfrac is preserved unchanged. Its macros do not accept non-finite values and still trap error conditions such as division by zero. Loading xintexpr automatically loads calx and adds \xinteval for computations with fractions. Most documents should compile unchanged, without requiring any modification to their sources.

- Version 1.4q dated 2026/10/31 was never released.
  - xintexpr: since 1.4a, round(x,n) and floor(x,n) admit a negative second argument. This worked fine if the result was involved in any operation, but depending on release since 1.4a and whether using \xintfloateval or \xinteval if the expression contained nothing else, it would either print with garbage or cause a low-level TEX error.
  - xintexpr: binary, octal, or hexadecimal inputs all support a fractional part; but due to an oversight regarding internal numeric types, when the integer part was zero, it could happen that the result was erroneously given as zero, for example this was teh case with \xinteval{"0.0001}.

- Version 1.4p dated 2026/03/01 was never released.
  - xintkernel: document (in the source code part of xint.pdf) \xintresettimer, \xintelapsedtime and \xinttheseconds, already added at 1.4n.
  - xintbinhex: add \xintLowerHex and \xintUpperHex.
  - xinttools: an array created via \xintAssignArray can be used with negative indices.
  - xintexpr: add all hyperbolic functions and their inverses.

- Release 1.4o of 2025/09/06 (with an 1.4n the day before) is mainly a maintenance release, after a few years in a dormant state. The long expected overhaul of floating point is again postponed.
  - xintbinhex handles the octal base, and can manage (much) larger inputs,
  - 0x, 0o, and 0b prefixes, and ' added to the \xinteval syntax,
  - [h], [o], [b] optional parameter of \xintiieval,
  - Babel active characters are auto-tamed in \xinteval (hotfix at 1.4o, the feature worked at 1.4n with \xintiieval but not with the others),
  - Compatibility with OpTEX,
  - Compatibility with ConTEXt (only mkxl).

  The extensive hyperlinking added in 2022 to the docs is better shown to user because xint.pdf now (again) contains both the user manual and the commented source code. The packaging was trimmed.

- Release 1.4m of 2022/06/09 is mainly a documentation upgrade, which added hyperlinks inside the commented macro code, as well as from the user manual to the source code (in xint-all.pdf, which is an optional build). It adds compatibility with miniltx. It also inaugurates usage of the engine string comparison primitive.

- Release 1.4i of 2021/06/11: extension of the «simultaneous assignments» concept (backwards compatible).

- Release 1.4g of 2021/05/25: powers are now parsed in a right associative way. Removal of the single-character operators &, |, and = (deprecated at 1.1). Reformatted expandable error messages.

- Release 1.4e of 2021/05/05: logarithms and exponentials up to 62 digits, trigonometry still mainly done at high level but with guard digits so all digits up to the last one included can be trusted for faithful rounding and high probability of correct rounding.

- Release 1.4 of 2020/01/31: xintexpr overhaul to use \expanded based expansion control. Many new features, in particular support for input and output of nested structures. Breaking changes, main ones being the (provisory) drop of x*[a, b,...], x+[a, b,...] et al.syntax and the requirement of \expanded primitive (currently required only by xintexpr).
- Release 1.3e of 2019/04/05: packages xinttrig, xintlog; \xintdefefunc ``non-protected'' variant of \xintdeffunc (at 1.4 the two got merged and \xintdefefunc became a deprecated alias for \xintdeffunc). Indices removed from xintsource.pdf.
- Release 1.3d of 2019/01/06: fix of 1.2p bug for division with a zero dividend and a one-digit divisor, \xinteval et al. wrappers, gcd() and lcm() work with fractions.
- Release 1.3c of 2018/06/17: documentation better hyperlinked, indices added to xintsource.pdf. Colon in := now optional for \xintdefvar and \xintdeffunc.
- Release 1.3b of 2018/05/18: randomness related additions (still WIP).
- Release 1.3a of 2018/03/07: efficiency fix of the mechanism for recursive functions.
- Release 1.3 of 2018/03/01: addition and subtraction use systematically least common multiple of denominators. Extensive under-the-hood refactoring of \xintNewExpr and \xintdeffunc which now allow recursive definitions. Removal of 1.2o deprecated macros.
- Release 1.2q of 2018/02/06: fix of 1.2l subtraction bug in special situation; tacit multiplication extended to cases such as 10!20!30!.
- Release 1.2p of 2017/12/05: maps // and /: to the floored, not truncated, division. Simultaneous assignments possible with \xintdefvar. Efficiency improvements in xinttools.
- Release 1.2o of 2017/08/29: massive deprecations of those macros from xintcore and xint which filtered their arguments via \xintNum.
- Release 1.2n of 2017/08/06: improvements of xintbinhex.
- Release 1.2m of 2017/07/31: rewrite of xintbinhex in the style of the 1.2 techniques.
- Release 1.2l of 2017/07/26: under the hood efficiency improvements in the style of the 1.2 techniques; subtraction refactored. Compatibility of most xintfrac macros with arguments using non-delimited \the\numexpr or \the\mathcode etc...
- Release 1.2i of 2016/12/13: under the hood efficiency improvements in the style of the 1.2 techniques.
- Release 1.2 of 2015/10/10: complete refactoring of the core arithmetic macros and faster \xintexpr parser.
- Release 1.1 of 2014/10/28: extensive changes in xintexpr. Addition and subtraction do not multiply denominators blindly but sometimes produce smaller ones. Also with that release, packages xintkernel and xintcore got extracted from xinttools and xint.
- Release 1.09g of 2013/11/22: the xinttools package is extracted from xint; addition of \xintloop and \xintiloop.
- Release 1.09c of 2013/10/09: \xintFor, \xintNewNumExpr (ancestor of \xintNewExpr/\xintdeffunc mechanism).
- Release 1.09a of 2013/09/24: support for functions by xintexpr.
- Release 1.08 of 2013/06/07: the xintbinhex package.

- Release 1.07 of 2013/05/25: support for floating point numbers added to xintfrac and first release of the xintexpr package (provided \xintexpr and \xintfloatexpr).
- Release 1.04 of 2013/04/25: the xintcfrac package.
- Release 1.03 of 2013/04/14: the xintfrac and xintseries packages.
- Release 1.0 of 2013/03/28: initial release of the xint and xintgcd packages.

19. Package xintkernel implementation

Contents

.1 Introduction . . . 248
.2 Catcodes, ε-TEX and reload detection . . . 248
.2.1 \XINTrestorecatcodes, \XINTsetcatcodes, \XINTrestorecatcodesendinput . . . 249
.3 Package identification . . . 251
.4 Constants . . . 252
.5 Token management utilities . . . 253
.6 "gob til" macros and UD style fork . . . 254
.7 \xint_afterfi . . . 255
.8 \xint_bye, \xint_Bye . . . 255
.9 \xintdothis, \xintorthat . . . 255
.10 \xint_zapspaces . . . 256
.11 \odef, \oodef, \fdef . . . 256
.12 \xintMessage, \ifxintverbose . . . 256
.13 \ifxintglobaldefs, \XINT_global . . . 257
.14 (WIP) Expandable error message . . . 257
.15 \xint_noxpd (for contex-mkxl compatibility) 259
.16 \xintstrcmp . . . 260
.17 \xintresettimer, \xintelapsedtime, \xinttheseconds . . . 260
.18 \xintReverseOrder . . . 261
.19 \xintLength . . . 262
.20 \xintLastItem . . . 262
.21 \xintFirstItem . . . 263
.22 \xintLastOne . . . 263
.23 \xintFirstOne . . . 264
.24 \xintLengthUpTo . . . 264
.25 \xintreplicate, \xintReplicate . . . 265
.26 \xintgobble, \xintGobble . . . 266
.27 Random number generation . . . 268
.27.1 \xinttexuniformdeviate . . . 269
.27.2 \xinttexuniformdeviate_dgts . . . 269
.27.3 \xintUniformDeviate . . . 270
.27.4 \xintsetrandomseed . . . 271

19.1. Introduction

This package provides the common minimal code base for loading management and catcode control and also a few programming utilities. With 1.2 a few more helper macros and all \chardef's have been moved here. The package is loaded by both xintcore.sty and xinttools.sty hence by all other packages.

Modified at 1.1 (2014/10/28). Separated package.

Modified at 1.2i (2016/12/13). \xintreplicate, \xintgobble, \xintLengthUpTo and \xintLastItem, and faster \xintLength.

Modified at 1.3b (2018/05/18). \xintUniformDeviate.

Modified at 1.4 (2020/01/31). \xintReplicate, \xintGobble, \xintLastOne, \xintFirstOne.

Modified at 1.4l (2022/05/29). Fix the 1.4 added bug that \XINTrestorecatcodes forgot to restore the catcode of ^^A which is set to 3 by \XINTsetcatcodes.

Modified at 1.4m (2022/06/10). Fix incompatibility under ε-TEX with miniltx, if latter was loaded before xintexpr. The fix happens here because it relates to matters of \ProvidesPackage.

19.2. Catcodes, ε-TEX and reload detection

The code for reload detection was initially copied from Heiko Oberdiek's packages, then modified.

The method for catcodes was also initially directly inspired by these packages.

1.4l replaces Info level user messages issued in case of problems such as \numexpr not being available with Warning level messages (in the LaTeX terminolgy). Should arguably be Error level in that case.

xintkernel.sty was the only xint package emitting such an Info, now Warning in case of being loaded twice (via \input in non-LaTeX). This was probably a left-over from initial development stage of the loading architecture for debugging. Starting with 1.4l, it will abort input silently in such case.

Also at 1.4l I refactored a bit the loading code in the xint*sty files for no real reason other than losing time.

```
\begingroup\catcode61\catcode48\catcode32=10\relax%
  \catcode13=5    % ^^M
  \endlinechar=13 %
  \catcode123=1   % {
  \catcode125=2   % }
  \catcode44=12   % ,
  \catcode46=12   % .
  \catcode58=12   % :
  \catcode94=7    % ^
  \def\space{ }\newlinechar10
  \let\z\relax
  \expandafter\ifx\csname numexpr\endcsname\relax
    \expandafter\ifx\csname PackageWarningNoLine\endcsname\relax
      \immediate\write128{^^JPackage xintkernel Warning:^^J%
                          \space\space\space\space
                          \numexpr not available, aborting input.^^J}%
    \else
      \PackageWarningNoLine{xintkernel}{\numexpr not available, aborting input}%
    \fi
    \def\z{\endgroup\endinput}%
  \else
    \expandafter\ifx\csname XINTsetupcatcodes\endcsname\relax
    \else
       \def\z{\endgroup\endinput}%
    \fi
  \fi
  \ifx\z\relax\else\expandafter\z\fi%
```

19.2.1. \XINTrestorecatcodes, \XINTsetcatcodes, \XINTrestorecatcodesendinput

Modified at 1.4e (2021/05/05). Renamed \XINT{set,restore}catcodes to be without underscores, to facilitate the reloading process for xintlog.sty and xinttrig.sty in uncontrolled contexts.

Modified at 1.4l (2022/05/29). Fix the 1.4 bug of omission of \catcode1 restore.

Reordered all catcodes assignements for easier maintenance and dropped most disparate indications of which packages make use of which settings.

The \XINTrestorecatcodes is somewhat misnamed as it is more a template to be used in an \edef to help define actual catcode restoring macros.

However \edef needs usually { and } so there is a potential difficulty with telling people to do \edef\myrestore{\XINTrestorecatcodes}, and I almost added at 1.4l some \XINTsettorestore:#1->\edef#1{\XINTrestorecatcodes} but well, this is not public interface anyhow. The reloading method of xintlog.sty and xinttrig.sty does protect itself though against such irreal usage possibility with non standard { or }.

Removed at 1.4l the \XINT_setcatcodes and \XINT_restorecatcodes not used anywhere now. Used by old version of xintsession.tex, but not anymore since a while.

Modified at 1.4n (2025/09/05). Compatibility with OpTEX. It has a cactode 11 letter, but \abc_ or \abc_d will be interpreted un expectedly (but not \abc_de). So we must make sure this is deactivated during the whole duration of loading the xint modules (perhaps it is mainly xinttools which uses \XINT_x, \XINT_y, which is problematic).

We need catcode letter _ for tokenization of \PrepareCatcodes to let it define \XINTrestorecatcodes conveniently for its usage in an \edef to query at that time the _ status in OpTEX.

```
  \catcode95=11 % _
  \def\PrepareCatcodes
```

```
30  {%
31      \endgroup
32      \def\XINTrestorecatcodes
33      {% prepared for use in \edef
34          \catcode0=\the\catcode0     % ^^@
35          \catcode1=\the\catcode1     % ^^A
36          \catcode13=\the\catcode13   % ^^M
37          \catcode32=\the\catcode32   % <space>
38          \catcode33=\the\catcode33   % !
39          \catcode34=\the\catcode34   % "
40          \catcode35=\the\catcode35   % #
41          \catcode36=\the\catcode36   % $
42          \catcode38=\the\catcode38   % &
43          \catcode39=\the\catcode39   % '
44          \catcode40=\the\catcode40   % (
45          \catcode41=\the\catcode41   % )
46          \catcode42=\the\catcode42   % *
47          \catcode43=\the\catcode43   % +
48          \catcode44=\the\catcode44   % ,
49          \catcode45=\the\catcode45   % -
50          \catcode46=\the\catcode46   % .
51          \catcode47=\the\catcode47   % /
52          \catcode58=\the\catcode58   % :
53          \catcode59=\the\catcode59   % ;
54          \catcode60=\the\catcode60   % <
55          \catcode61=\the\catcode61   % =
56          \catcode62=\the\catcode62   % >
57          \catcode63=\the\catcode63   % ?
58          \catcode64=\the\catcode64   % @
59          \catcode91=\the\catcode91   % [
60          \catcode93=\the\catcode93   % ]
61          \catcode94=\the\catcode94   % ^
62          \catcode95=\the\catcode95   % _
63          \catcode96=\the\catcode96   % `
64          \catcode123=\the\catcode123 % {
65          \catcode124=\the\catcode124 % |
66          \catcode125=\the\catcode125 % }
67          \catcode126=\the\catcode126 % ~
68          \endlinechar=\the\endlinechar\relax
69          \ifdefined\_ifmathsb\_ifmathsb\noexpand\_mathsbon\_fi\fi %
70      }%
```

The \noexpand here before \endinput is required. This feels to me a bit surprising, but is a fact, and the source of this must be in the \edef implementation but I have not checked it out at this time.

Modified at 1.4n (2025/09/05). Compatibility with OpTEX.

```
71      \edef\XINTrestorecatcodesendinput
72      {%
73           \XINTrestorecatcodes\noexpand\endinput %
74      }%
75      \def\XINTsetcatcodes
76      {% standard settings with a few xint*sty specific ones
77          \catcode0=12    % for \romannumeral`&&@
```

```
78            \catcode1=3     % for safe separator &&A
79            \catcode13=5    % ^^M
80            \catcode32=10   % <space>
81            \catcode33=12   % ! but used as LETTER inside xintexpr.sty
82            \catcode34=12   % "
83            \catcode35=6    % #
84            \catcode36=3    % $
85            \catcode38=7    % & SUPERSCRIPT for && as replacement of ^^
86            \catcode39=12   % '
87            \catcode40=12   % (
88            \catcode41=12   % )
89            \catcode42=12   % *
90            \catcode43=12   % +
91            \catcode44=12   % ,
92            \catcode45=12   % -
93            \catcode46=12   % .
94            \catcode47=12   % /
95            \catcode58=11   % : LETTER
96            \catcode59=12   % ;
97            \catcode60=12   % <
98            \catcode61=12   % =
99            \catcode62=12   % >
100           \catcode63=11   % ? LETTER
101           \catcode64=11   % @ LETTER
102           \catcode91=12   % [
103           \catcode93=12   % ]
104           \catcode94=11   % ^ LETTER
105           \catcode95=11   % _ LETTER
106           \catcode96=12   % `
107           \catcode123=1   % {
108           \catcode124=12  % |
109           \catcode125=2   % }
110           \catcode126=3   % ~ MATH SHIFT
111           \endlinechar=13 %
112           \ifdefined\_mathsboff\_mathsboff\fi % Compatibility with OpTeX
113       }%
114       \XINTsetcatcodes
115   }%
116 \PrepareCatcodes
```

Other modules could possibly be loaded under a different catcode regime. (or with a different status of _ under OpTEX).

```
117 \def\XINTsetupcatcodes {% for use by other modules
118       \edef\XINTrestorecatcodesendinput
119       {%
120            \XINTrestorecatcodes\noexpand\endinput %
121       }%
122       \XINTsetcatcodes
123 }%
```

19.3. Package identification

Inspired from Heiko Oberdiek's packages.

Modified at 1.09b (2013/10/03). Re-usability in the other modules. Also I assume now that if \ProvidesPackage exists it then does define \ver@<pkgname>.sty, code of HO for some reason escaping me (compatibility with LaTeX 2.09 or other things ??) seems to set extra precautions. [nine years later I understood my mistake, see below].

Modified at 1.09c (2013/10/09). Usage of ε-TEX \ifdefined.

Modified at 1.4m (2022/06/10). Nine years too late, I understand that the HO "extra precautions" were there for some respectable reasons including etex+miniltx and surely other things I can not imagine. So let's now make sure \ver@xintkernel.sty and friends get defined on load, even if \ProvidesPackage exists! However I remain careless in using \ifdefined which could be fooled if some previous macro file ended up testing for \ProvidesPackage in a way letting it to \relax. I do not test for that. If I fixed that carelessness here I would have to fix it in other places where I use similarly \ifdefined\RequirePackage or \ifdefined\PackageWarning or whatever.

```
\ifdefined\ProvidesPackage
   \def\XINT_providespackage\ProvidesPackage#1[#2]{%
              \ProvidesPackage{#1}[{#2}]%
              \expandafter\ifx\csname ver@#1.sty\endcsname\relax
                    \expandafter\xdef\csname ver@#1.sty\endcsname{#2}%
                          \fi
        }%
\else
   \def\XINT_providespackage\ProvidesPackage#1[#2]{%
              \immediate\write-1{Package: #1 #2}%
              \expandafter\xdef\csname ver@#1.sty\endcsname{#2}%
        }%
\fi
\XINT_providespackage
\ProvidesPackage {xintkernel}%
  [2026/11/26 v2.0a2 Paraphernalia for the xint packages (JFB)]%
```

19.4. Constants

```
\chardef\xint_c_     0
\chardef\xint_c_i    1
\chardef\xint_c_ii   2
\chardef\xint_c_iii  3
\chardef\xint_c_iv   4
\chardef\xint_c_v    5
\chardef\xint_c_vi   6
\chardef\xint_c_vii  7
\chardef\xint_c_viii 8
\chardef\xint_c_ix     9
\chardef\xint_c_x     10
%\chardef\xint_c_xi   11
\chardef\xint_c_xii   12
\chardef\xint_c_xiv   14
\chardef\xint_c_xvi   16
\chardef\xint_c_xvii  17
\chardef\xint_c_xviii 18
\chardef\xint_c_xx    20
\chardef\xint_c_xxii  22
\chardef\xint_c_ii^v          32
\chardef\xint_c_ii^vi         64
```

```
%\chardef\xint_c_c           100
\chardef\xint_c_ii^vii       128
\mathchardef\xint_c_ii^viii  256
\mathchardef\xint_c_ii^ix    512
\mathchardef\xint_c_ii^xii  4096
%\mathchardef\xint_c_m      1000
```

Some of these usages of \newcount were in xintcore or in xint or in xintbinhex possibly conditionally on whether (pdf)\uniformdeviate is available. At 1.4n, let's not bother with outdated restrictions of TEX.

Modified at 1.4p (2026/03/01). new naming scheme without ^ regarding higher powers of ten, for matters of easier interconnection with poormanlog.

```
\ifdefined\m@ne\let\xint_c_mone\m@ne
          \else\csname newcount\endcsname\xint_c_mone \xint_c_mone -1 %
\fi
\mathchardef\xint_c_e_iv                                    10000
\newcount\xint_c_e_v             \xint_c_e_v                100000
%\newcount\xint_c_e_vi            \xint_c_e_vi              1000000
%\newcount\xint_c_e_vii           \xint_c_e_vii            10000000
\newcount\xint_c_e_viii          \xint_c_e_viii          100000000
\newcount\xint_c_e_ix            \xint_c_e_ix           1000000000
\newcount\xint_c_e_viii_mone     \xint_c_e_viii_mone      99999999
\newcount\xint_c_nine_e_viii     \xint_c_nine_e_viii     900000000
\newcount\xint_c_xi_e_viii_mone  \xint_c_xi_e_viii_mone 1099999999
\newcount\xint_c_xii_e_viii      \xint_c_xii_e_viii     1200000000
```

Modified at 1.4n (2025/09/05). For some reason this next one used to be defined by \newcount but \mathchardef is ok.

```
\mathchardef\xint_c_ii^xiv                  16384 % "4000, 2**14
\newcount\xint_c_ii^xv     \xint_c_ii^xv    32768 % 2**15
\newcount\xint_c_ii^xvi    \xint_c_ii^xvi   65536 % 2**16
\newcount\xint_c_ii^xxi    \xint_c_ii^xxi 2097152 % "200000, 2**21
```

19.5. Token management utilities

Added at 1.2 (2015/10/10). Check if \empty and \space have their standard meanings and raise a warning if not.

Modified at 1.4 (2020/01/31). Warn user if needed, and force then \empty and \space to have their standard meanings. This will be triggered even if the sole difference is that they are \long.

Modified at 1.4n (2025/09/05). The Warning used to be emitted only to the log file, make it go to console output as well. But why do I spend time on such silly things.

```
\def\XINT_tmpa { }%
\ifx\XINT_tmpa\space\else
   \immediate\write128{Package xintkernel Warning:}%
   \immediate\write128{The \string\space\XINT_tmpa macro does not have its
     meaning as in Plain or LaTeX, but is:}%
   \immediate\write128{\XINT_tmpa\XINT_tmpa\XINT_tmpa\XINT_tmpa\meaning\space.}%
   \let\space\XINT_tmpa
   \immediate\write128{Forcing it to be the usual one. Fingers crossed.}%
\fi
\def\XINT_tmpa {}%
\ifx\XINT_tmpa\empty\else
```

```
    \immediate\write128{Package xintkernel Warning:}%
    \immediate\write128{The \string\empty\space macro does not have its
      meaning as in Plain or LaTeX, but is:}%
    \immediate\write128{\space\space\space\space\meaning\empty.}%
    \let\empty\XINT_tmpa
    \immediate\write128{Forcing it to be the usual one. Fingers crossed.}%
\fi
\let\XINT_tmpa\relax
\let\xint_gobble_\empty
\long\def\xint_gobble_i     #1{}%
\long\def\xint_gobble_ii    #1#2{}%
\long\def\xint_gobble_iii   #1#2#3{}%
\long\def\xint_gobble_iv    #1#2#3#4{}%
\long\def\xint_gobble_v     #1#2#3#4#5{}%
\long\def\xint_gobble_vi    #1#2#3#4#5#6{}%
\long\def\xint_gobble_vii   #1#2#3#4#5#6#7{}%
\long\def\xint_gobble_viii #1#2#3#4#5#6#7#8{}%
```

Modified at 1.3b (2018/05/18). Moved here \xint_gobandstop_... macros because this is handy for \xintRandomDigits.

For legacy reasons most top level macros use \romannumeral0 trigger. This is stopped by a space token. Later in the history of the package \romanumeral`&&@ was used with & of catcode 7. Also there are a few instances of \romannumeral ended by \z@. But \romannumeral0 remains the publicly documented one, with CamelCase macros using it as prefix to lowercased macros.

```
\let\xint_gob_andstop_\space
\long\def\xint_gob_andstop_i     #1{ }%
\long\def\xint_gob_andstop_ii    #1#2{ }%
\long\def\xint_gob_andstop_iii   #1#2#3{ }%
\long\def\xint_gob_andstop_iv    #1#2#3#4{ }%
\long\def\xint_gob_andstop_v     #1#2#3#4#5{ }%
\long\def\xint_gob_andstop_vi    #1#2#3#4#5#6{ }%
\long\def\xint_gob_andstop_vii   #1#2#3#4#5#6#7{ }%
\long\def\xint_gob_andstop_viii #1#2#3#4#5#6#7#8{ }%
\let\xint_stop_aftergobble\xint_gob_andstop_i
\long\def\xint_firstofone  #1{#1}%
\long\def\xint_firstoftwo  #1#2{#1}%
\long\def\xint_secondoftwo #1#2{#2}%
\long\def\xint_stop_atfirstofone  #1{ #1}%
\long\def\xint_stop_atfirstoftwo  #1#2{ #1}%
\long\def\xint_stop_atsecondoftwo #1#2{ #2}%
\long\def\xint_exchangetwo_keepbraces    #1#2{{#2}{#1}}%
```

Moved here from xint at 1.4n.

```
\long\def\xint_firstofthree  #1#2#3{#1}%
\long\def\xint_secondofthree #1#2#3{#2}%
\long\def\xint_thirdofthree  #1#2#3{#3}%
\long\def\xint_stop_atfirstofthree  #1#2#3{ #1}%
\long\def\xint_stop_atsecondofthree #1#2#3{ #2}%
\long\def\xint_stop_atthirdofthree  #1#2#3{ #3}%
```

19.6. “gob til” macros and UD style fork

```
\long\def\xint_gob_til_R #1\R {}%
\long\def\xint_gob_til_W #1\W {}%
```

```
\long\def\xint_gob_til_Z #1\Z {}%
\long\def\xint_gob_til_zero #10{}%
\long\def\xint_gob_til_one  #11{}%
\long\def\xint_gob_til_zeros_iii   #1000{}%
\long\def\xint_gob_til_zeros_iv    #10000{}%
\long\def\xint_gob_til_eightzeroes #100000000{}%
\long\def\xint_gob_til_dot    #1.{}%
\long\def\xint_gob_til_G     #1G{}%
\long\def\xint_gob_til_minus #1-{}%
\long\def\xint_UDzerominusfork #10-#2#3\krof {#2}%
\long\def\xint_UDzerofork       #10#2#3\krof {#2}%
\long\def\xint_UDsignfork       #1-#2#3\krof {#2}%
\long\def\xint_UDwfork         #1\W#2#3\krof {#2}%
\long\def\xint_UDXINTWfork     #1\XINT_W#2#3\krof {#2}%
\long\def\xint_UDzerosfork     #100#2#3\krof {#2}%
\long\def\xint_UDonezerofork   #110#2#3\krof {#2}%
\long\def\xint_UDsignsfork     #1--#2#3\krof {#2}%
\let\xint:\char
\long\def\xint_gob_til_xint:#1\xint:{}%
\long\def\xint_gob_til_^#1^{}%
\def\xint_bracedstopper{\xint:}%
\long\def\xint_gob_til_exclam #1!{}% This ! has catcode 12
\long\def\xint_gob_til_sc #1;{}%
```

19.7. \xint_afterfi

```
\long\def\xint_afterfi #1#2\fi {\fi #1}%
```

19.8. \xint_bye, \xint_Bye

Modified at 1.09 (2013/09/23). \xint_bye

Modified at 1.2i (2016/12/13). \xint_Bye for \xintDSRr and \xintRound. Also \xint_stop_after↵ ↪ bye.

```
\long\def\xint_bye #1\xint_bye {}%
\long\def\xint_Bye #1\xint_bye {}%
\long\def\xint_stop_afterbye #1\xint_bye { }%
```

19.9. \xintdothis, \xintorthat

Modified at 1.1 (2014/10/28).

Modified at 1.2 (2015/10/10). Names without underscores.

To be used this way:

```
\if..\xint_dothis{..}\fi
\if..\xint_dothis{..}\fi
\if..\xint_dothis{..}\fi
...more such...
\xint_orthat{...}
```

Ancient testing indicated it is more efficient to list first the more improbable clauses.

```
\long\def\xint_dothis #1#2\xint_orthat #3{\fi #1}% 1.1
\let\xint_orthat \xint_firstofone
\long\def\xintdothis #1#2\xintorthat #3{\fi #1}%
\let\xintorthat \xint_firstofone
```

19.10. \xint_zapspaces

Modified at 1.1 (2014/10/28). This little (quite fragile in the normal sense i.e. non robust in the normal sense of programming lingua) utility zaps leading, intermediate, trailing, spaces in completely expanding context (\edef, \csname...\endcsname).

Usage: \xint_zapspaces foo<space>\xint_gobble_i

Explanation: if there are leading spaces, then the first #1 will be empty, and the first #2 being undelimited will be stripped from all the remaining leading spaces, if there was more than one to start with. Of course brace-stripping may occur. And this iterates: each time a #2 is removed, either we then have spaces and next #1 will be empty, or we have no spaces and #1 will end at the first space. Ultimately #2 will be \xint_gobble_i.

The \zap@spaces of LaTeX2e handles unexpectedly things such as

\zap@spaces 1 {22} 3 4 \@empty

(spaces are not all removed). This does not happen with \xint_zapspaces.

But for example \foo{aa} {bb} {cc} where \foo is a macro with three non-delimited arguments breaks expansion, as expansion of \foo will happen with \xint_zapspaces still around, and even if it wasn't it would have stripped the braces around {bb}, certainly breaking other things.

Despite such obvious shortcomings it is enough for our purposes. It is currently used by xintexpr at various locations e.g. cleaning up optional argument of \xintiexpr and \xintfloatexpr; maybe in future internal usage will drop this in favour of a more robust utility.

Modified at 1.2e (2015/11/22). \xint_zapspaces_o.

Modified at 1.2i (2016/12/13). Made \long.

ATTENTION THAT xinttools HAS AN \xintzapspaces WHICH SHOULD NOT GET CONFUSED WITH THIS ONE.

```
\long\def\xint_zapspaces #1 #2{#1#2\xint_zapspaces }% 1.1
\long\def\xint_zapspaces_o #1{\expandafter\xint_zapspaces#1 \xint_gobble_i}%
```

19.11. \odef, \oodef, \fdef

May be prefixed with \global. No parameter text.

```
\def\xintodef #1{\expandafter\def\expandafter#1\expandafter }%
\def\xintoodef #1{\expandafter\expandafter\expandafter\def
                  \expandafter\expandafter\expandafter#1%
                  \expandafter\expandafter\expandafter }%
\def\xintfdef #1#2%
    {\expandafter\def\expandafter#1\expandafter{\romannumeral`&&@#2}}%
\ifdefined\odef\else\let\odef\xintodef\fi
\ifdefined\oodef\else\let\oodef\xintoodef\fi
\ifdefined\fdef\else\let\fdef\xintfdef\fi
```

19.12. \xintMessage, \ifxintverbose

Modified at 1.2c (2015/11/16). For use by \xintdefvar and \xintdeffunc of xintexpr.

Modified at 1.2e (2015/11/22). Uses \write128 rather than \write16 for compatibility with future extended range of output streams, in LuaTEX in particular.

Modified at 1.3e (2019/04/05). Set the \newlinechar.

```
\edef\XINT_fourspaces{\space\space\space\space}%
\def\xintMessage #1#2#3{%
    \edef\XINT_newlinechar{\the\newlinechar}%
    \newlinechar10
    \immediate\write128{Package #1 #2: (on line \the\inputlineno)}%
    \immediate\write128{\XINT_fourspaces#3}%
```

```
    \newlinechar\XINT_newlinechar\space
}%
\newif\ifxintverbose
```

19.13. \ifxintglobaldefs, \XINT_global

Modified at 1.3c (2018/06/17).

```
\newif\ifxintglobaldefs
\def\XINT_global{\ifxintglobaldefs\global\fi}%
```

19.14. (WIP) Expandable error message

Modified at 1.2l (2017/07/26). But really belongs to next major release beyond 1.3. Basically copied over from l3kernel code. Using \ ! / control sequence, which must be left undefined. \xintError: would be 6 letters more.

Modified at 1.4 (2020/01/31). Finally rather than \ ! / I use \xint/.

Modified at 1.4g (2021/05/25). Rewrote to use not an undefined control sequence but trigger "Use of \xint/ doesn't match its definition." message.

Modified at 1.4g (2021/05/25). Things evolve fast and I switch to a third method which will exploit "Paragraph ended before \foo was complete" style error. See

https://github.com/latex3/latex3/issues/931#issuecomment-845367201

However I can not fully exploit this because xint may be used with Plain etex which does not set \newlinechar. I can only use a poorman version with no usage of ^^J. Also xintsession could use the ^^J, maybe I will integrate it there.

I. Explanations on 2021/05/19 and 2021/05/20 before final change

First I tried out things with undefined control sequence such as

```
\       an error was reported by xint ...
```

whose output produces a nice symmetrical display with no \, and with ... both on left and right but this reduces drastically the available space for the actual error context. No go. But see 2021/05/20 update below!

Having replaced \xint/ by "\xint ", I next opted provisorily for "\Hit RET at ?" control sequence, despite it being quite longer. And then I thought about using "\ xint error", possibly with an included ^^J in the name, or in the context.

I experimented with ^^J in the context. But the context size is much constrained, and when \errorcontextlines is at its default value of 5 for etex, not -1 as done by LaTeX, having the info shifted to the right makes it actually more visible. (however I have now updated xintsession to 0.2b which sets \errorcontextlines to 0)

So I was finally back here to square one, apart from having replaced "\xint/" by the more longish "\ xint error", hesitating with "\xinterrupt"...

Then I had the idea to replace the undefined control sequence method by a method with a macro \foo defined as \def\foo.{} but used as \foo<space> for example. This gives something like this (the first line will be otherwise if engine is run with -file-line-error):

```
   ! Use of \xint/ doesn't match its definition.
   <argument> \xint/
                          Ooops, looks like we are missing a ] (hit RET)
```

\xint/<space> (where the space is the unexpected token, the definition expecting rather a full stop) makes for 7 characters to compare to \ xint error which had 12, so I gained back 5.

Back to ^^J: I had overlooked that TeX in the first part of the error message will display \macro fully, so inserting ^^J in its name allows arbitrarily long expandable error messages... as pointed out by BLF in latex3/issues#931 as I read on the morning of 2021/05/20. This is very nice but requires to predefine control sequences for each message, and also the actual arguments #1, #2, ... values can appear only in the context.

And the situation with ^^J is somewhat complicated:

xintsession sets the \newlinechar to 10, but this is not the case with bare usage of xintexpr with etex. And this matters. To discuss ^^J we have to separate two locations:

- it appears in the control sequence name,
- or in the context (which itself has two parts)

1) When in the context, what happens with ^^J is independent of the setting of \newlinechar, and with TeXLive pdflatex the ^^J will induce a linebreak, but with xelatex it must be used with option -8bit.

2) When in the control sequence name the behaviour in log/terminal of ^^J is influenced by the setting of \newlinechar. Although with pdflatex it will always induce a linebreak, the actual count of characters where TeX will forcefully break is influenced by whether ^^J is or not \newlinechar. And with xelatex if it is \newlinechar, it does not depend then if -8bit or not, but if not \newlinechar then it does and TeX forceful breaks also change as for pdflatex.

So, the control sequence name trick can be used to obtain arbitrarily long messages, but the \newlinechar must be set.

And in the context, we can try to insert some ^^J but this would need with xetex the -8bit option, and anyhow the context size is limited, and there is apparently no trick to get it larger.

So, in view of all the above I decided not to use ^^J (rather &&J here) at all, whether here in the control sequence or the context or inserted in \XINT_signalcondition in the context!

I also have a problem with usage from bnumexpr or polexpr for example, they would need their own to avoid perhaps displaying \xint/ or analogous.

II. Finally I modified again the method (completely, and no more need for funny catcode 7 space as delimiter) as this allows a longer context message, starting at start of line, and which obeys ^^J if \newlinechar is set to it. It also allows to incorporate non-limited generic explanations as a postfix, with linebreaks if \newlinechar is known.

But as xintexpr can be used with Plain+etex which does not set the \newlinechar, I can't use ^^J out of thee box. I can in xintsession. What I decided finally is to make a conditional definition here.

In both cases I include the "hit RET" (how rather "hit <return>") in the control sequence name serving to both provide extra information and trigger the error from being defined short and finding a \par.

The maximal size was increased from 48 characters (method with \xint/ being badly delimited), to now 55 characters (using "! xint error:<^^J or space>" as prefix to the message). Longer messages are truncated at 56 characters with an appended "\ETC.".

As it is late on this 2021/05/20, and in order to not have to change all usages, I keep \XINT_signalcondition (in xintcore) as a one argument macro for time being, so will not include a more specific module name.

The \par token has a special role here, and can't be (I)nserted without damage, but who would want to insert it in an expandable computation anyhow... and I don't need it in my custom error messages for sure.

On 2021/05/21 I add a test about \newlinechar at time of package loading, and make two distinct definitions: one using ^^J in the control sequence, the other not using it.

The -file-line-error toggle makes it impossible to control if the line-break on first line will match next lines. In the ^^J branch I insert "| " (no, finally " " with two spaces) at start of continuation lines. Also I preferred to ensure a good-looking first line break for the case it starts with a "! Paragraph ended ..." because a priori error messages will be read if -file-line-error was emitted only a fortiori (this toggle suggests some IDE launched TeX and probably -interaction=nonstopmode).

I will perhaps make another definition in xintsession (it currently loads xintexpr prior to having set the \newlinechar, so the no ^^J definition will be used, if nothing else is modified there).

With some hesitation I do not insert a ^^J after "! xint error:", as Emacs/AucTeX will display

only the first line prominently and then the rest (which is in file:line:error mode) in one block under "--- TeX said ---". I use the ^^J only in the generic helper message embedded in the control sequence. The cases with or without \newlinechar being 10 diverge a bit, as in the latter case I had to ensure acceptable linebreaks at 79 chars, and I did that first and then had spent enough time on the matter not to add more to backport the latest ^^J style message.

Modified at 1.4m (2022/06/10). Shorten the error message. I am always too verbose initially.

Modified at 1.4n (2025/09/05). Let not use explicit \par token as delimiter, but implicit one from empty line. This is for compatibility with OpTEX. Indeed, I observed that with OpTEX, {\foo\par} wuth \foo being short reports an extra } rather than saying ``Paragraph ended before \foo was complete.''.

```
\ifnum\newlinechar=10
\expandafter\def\csname
xint<...> is done, but will resume:&&J \space
hit <return> at the ? prompt to try fixing the error above&&J \space
which has been encountered before expansion\endcsname
#1\xint:{}%
\def\XINT_expandableerror#1{%
\def\XINT_expandableerror##1{%
   \expandafter
   \XINT_expandableerrorcontinue
   #1! xint error: ##1%

}}\expandafter\XINT_expandableerror\csname
xint<...> is done, but will resume:&&J \space
hit <return> at the ? prompt to try fixing the error above&&J \space
which has been encountered before expansion\endcsname
\else
\expandafter\def\csname
xint<...> is done, but will resume: hit <return> at \space
the ? prompt to try fixing the error encountered before expansion\endcsname
#1\xint:{}%
\def\XINT_expandableerror#1{%
\def\XINT_expandableerror##1{%
   \expandafter
   \XINT_expandableerrorcontinue
   #1! xint error: ##1%

}}\expandafter\XINT_expandableerror\csname
xint<...> is done, but will resume: hit <return> at \space
the ? prompt to try fixing the error encountered before expansion\endcsname
\fi
\def\XINT_expandableerrorcontinue#1%

{#1}%
```

19.15. \xint_noxpd (for contex-mkxl compatibility)

Added at 1.4n (2025/09/05).

The LaTeX3 code (in l3names.dtx) uses ConTEXt's \normalunexpanded and \normalexpanded which appear to be available both in Mark II and Mark IV. But I will not do any testing with earlier context, so here I shall assume the \expanded behaves as expected and I can use \notexpanded for \unexpanded. This is abstracted into an alias \xint_noxpd.

I don't know if there is mode of running context where the \errhelp tokens will be shown (trying context --mkII was not conclusive; by the way \errmessage in e-TEX adds a full stop which is missing with context).

About the \let primitive I am not sure it behaves fully as in other engines.

Note that this uses \xintMessage in case of old ConTEXt so we had to have it defined first.

```
\let\xint_noxpd\unexpanded
\ifdefined\contextversion
  \let\xint_noxpd\notexpanded
  \ifdefined\notexpanded
  \else
    \xintMessage{xintkernel}{Error}{This ConTeXt appears to be too old.}
    \errhelp{xint only supports the ConTeXt-LMTX as of 2025/09/05.}%
    \errmessage{The \noexpand\notexpanded primitive does not exist.}%
  \fi
\fi
```

19.16. \xintstrcmp

Added at 1.4m (2022/06/10) [on 2022/06/05]. For the LuaTEX engine the code is copied over from l3names.dtx. I also looked at Heiko Oberdiek's pdftexcmds.sty and pdftexcmds.lua, but I removed \luaescapestring and used token.scan_string() as seen in l3names.dtx (and I did try to inform myself about this in the LuaTEX manual, with limited success). I am not sure about the syntax below with the local's. Should I use \directlua0? Testing was minimal. Memo: even with pdftex's \pdfstrcmp, braces around the arguments are mandatory.

```
\ifdefined\strcmp\let\xintstrcmp\strcmp
\else\ifdefined\pdfstrcmp\let\xintstrcmp\pdfstrcmp
\else\ifdefined\directlua\directlua{%
xintkernel = xintkernel or {}
local minus_tok = token.new(string.byte'-', 12)
local zero_tok = token.new(string.byte'0', 12)
local one_tok = token.new(string.byte'1', 12)
function xintkernel.strcmp()
  local A = token.scan_string()
  local B = token.scan_string()
  if A < B then
    tex.write(minus_tok, one_tok)
  else
    tex.write(A == B and zero_tok or one_tok)
  end
end
}\def\xintstrcmp{%
  \directlua{xintkernel.strcmp()}%
}%
\else
\xintMessage{xintkernel}{Error}{Could not set-up \string\xintstrcmp.}%
\errhelp{What kind of format are you using? Perhaps write the author? Bye now}%
\errmessage{Sorry, could not find or define string comparison primitive}\fi\fi\fi
```

19.17. \xintresettimer, \xintelapsedtime, \xinttheseconds

Added at 1.4n (2025/09/05). If \resettimer is defined, the code assumes \elapsedtime is, too.

I completely forgot at 1.4n release time to document these engine-agnostic utilities in the manual. They were originally in the xint.dtx preamble, to allow testing a build of the documentation also with the Unicode engines (but the release is done with latex+dvipdfmx). They are used only once there, for the example with \xintUniformDeviate.

Perhaps I should for LuaTEX copy the l3kernel code (here the code was picked from some tex.sx answer and was in my files for years) but I am not Lua-proficient enough to do this confidently... probably their // 1 and math.tointeger() is more efficient than the math.floor() and there must be a reason why they use gettimeofday(), while os.clock() is only used to set the ``epoch''.

I have been using for years the underlying syntax trick here for \xinttheseconds, and I usually never bother actually removing the catcode 12 pt, but well let's do it here.

I hesitate whether with LuaTEX I should avoid the scaled seconds intermediate in favor of the raw os.clock()-basetime data but I would have to waste time seeing what happens with tex.write() then. And it would not help really in comparing with PDFTEX or XeTEX.

Modified at 1.4o (2025/09/06).

xinttimer replaces here in the Lua code xintelapsedtimer, which was a silly name used when I adopted in a haste at 1.4n while improving the user manual on \xintUniformDeviate.

```
\ifdefined\resettimer
  \let\xintresettimer\resettimer   \let\xintelapsedtime\elapsedtime
\else
\ifdefined\pdfresettimer
  \let\xintresettimer\pdfresettimer\let\xintelapsedtime\pdfelapsedtime
\else
\ifdefined\directlua
  \directlua{xinttimer_basetime=0}%
  \def\xintresettimer{\directlua{xinttimer_basetime = os.clock()}}%
  \def\xintelapsedtime{\numexpr
      \directlua{tex.print(math.floor((os.clock()-xinttimer_basetime)*65536+0.5))}%
  \relax}%
\fi\fi\fi
\def\xintstrippt{\expandafter\XINT_strippt\the}%
\expanded{\xint_noxpd{\def\XINT_strippt#1}\detokenize{pt}}{#1}%
\def\xinttheseconds{\xintstrippt\dimexpr\xintelapsedtime sp\relax}%
```

19.18. \xintReverseOrder

Modified at 1.0 (2013/03/28). Does not expand its argument. The whole of xint codebase now contains only two calls to \XINT_rord_main (in xintgcd).

Attention: removes brace pairs (and swallows spaces).

For digit tokens a faster reverse macro is provided by (1.2) \xintReverseDigits in xint.

For comma separated items, 1.2g has \xintCSVReverse in xinttools.

```
\def\xintReverseOrder {\romannumeral0\xintreverseorder }%
\long\def\xintreverseorder #1%
{%
    \XINT_rord_main {}#1%
      \xint:
        \xint_bye\xint_bye\xint_bye\xint_bye
        \xint_bye\xint_bye\xint_bye\xint_bye
      \xint:
}%
\long\def\XINT_rord_main #1#2#3#4#5#6#7#8#9%
{%
    \xint_bye #9\XINT_rord_cleanup\xint_bye
```

```
    \XINT_rord_main {#9#8#7#6#5#4#3#2#1}%
}%
\def\XINT_rord_cleanup #1{%
\long\def\XINT_rord_cleanup\xint_bye\XINT_rord_main ##1##2\xint:
{%
    \expandafter#1\xint_gob_til_xint: ##1%
}}\XINT_rord_cleanup { }%
```

19.19. \xintLength

Modified at 1.0 (2013/03/28). Does not expand its argument. See \xintNthElt{0} from xinttools which f-expands its argument.

Modified at 1.2g (2016/03/19). Added \xintCSVLength to xinttools.

Modified at 1.2i (2016/12/13). Rewrote this venerable macro. New code about 40% faster across all lengths. Syntax with \romannumeral0 adds some slight (negligible) overhead; it is done to fit some general principles of structure of the xint package macros but maybe at some point I should drop it. And in fact it is often called directly via the \numexpr access point. (bad coding...)

Remark: the argument may contain \par tokens. But generally speaking most other macros of the xint bundle are not declared \long. As \xintLength produces a numeric quantity it is conceivable that it could serve in the input of some of the xint macros. For example something such as \xinteval{\xintLength{\par\par}^3} or \xintiiMul{\xintLength{\par\par\par}}{17}. They both fail. I have known this issue for many years. It is only needed to make long those macros which grab the argument and *f*-expand it. After expansion, of course no \par tokens is admissible as numerical input. Still this is quite some work due to size of the codebase. Waiting for a real-life bug report... (LaTeX3 people have fixed that on their side by making all declarations \long per default, and this may also perhaps increase slightly the efficiency, not checked).

```
\def\xintLength {\romannumeral0\xintlength }%
\def\xintlength #1{%
\long\def\xintlength ##1%
{%
    \expandafter#1\the\numexpr\XINT_length_loop
    ##1\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:
       \xint_c_viii\xint_c_vii\xint_c_vi\xint_c_v
       \xint_c_iv\xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye
    \relax
}}\xintlength{ }%
\long\def\XINT_length_loop #1#2#3#4#5#6#7#8#9%
{%
    \xint_gob_til_xint: #9\XINT_length_finish_a\xint:
    \xint_c_ix+\XINT_length_loop
}%
\def\XINT_length_finish_a\xint:\xint_c_ix+\XINT_length_loop
    #1#2#3#4#5#6#7#8#9%
{%
    #9\xint_bye
}%
```

19.20. \xintLastItem

Modified at 1.2i (2016/12/13). One level of braces removed in output. Output empty if input empty. Attention! This means that an empty input or an input ending with a empty brace pair

both give same output.

The \xint: token must not be among items. \xintFirstItem added at 1.4 for usage in xintexpr. It must contain neither \xint: nor \xint_bye in its first item.

```
\def\xintLastItem {\romannumeral0\xintlastitem }%
\long\def\xintlastitem #1%
{%
    \XINT_last_loop {}.#1%
    {\xint:\XINT_last_loop_enda}{\xint:\XINT_last_loop_endb}%
    {\xint:\XINT_last_loop_endc}{\xint:\XINT_last_loop_endd}%
    {\xint:\XINT_last_loop_ende}{\xint:\XINT_last_loop_endf}%
    {\xint:\XINT_last_loop_endg}{\xint:\XINT_last_loop_endh}\xint_bye
}%
\long\def\XINT_last_loop #1.#2#3#4#5#6#7#8#9%
{%
    \xint_gob_til_xint: #9%
        {#8}{#7}{#6}{#5}{#4}{#3}{#2}{#1}\xint:
    \XINT_last_loop {#9}.%
}%
\long\def\XINT_last_loop_enda #1#2\xint_bye{ #1}%
\long\def\XINT_last_loop_endb #1#2#3\xint_bye{ #2}%
\long\def\XINT_last_loop_endc #1#2#3#4\xint_bye{ #3}%
\long\def\XINT_last_loop_endd #1#2#3#4#5\xint_bye{ #4}%
\long\def\XINT_last_loop_ende #1#2#3#4#5#6\xint_bye{ #5}%
\long\def\XINT_last_loop_endf #1#2#3#4#5#6#7\xint_bye{ #6}%
\long\def\XINT_last_loop_endg #1#2#3#4#5#6#7#8\xint_bye{ #7}%
\long\def\XINT_last_loop_endh #1#2#3#4#5#6#7#8#9\xint_bye{ #8}%
```

19.21. \xintFirstItem

1.4. There must be neither \xint: nor \xint_bye in its first item.

```
\def\xintFirstItem          {\romannumeral0\xintfirstitem }%
\long\def\xintfirstitem  #1{\XINT_firstitem #1{\xint:\XINT_firstitem_end}\xint_bye}%
\long\def\XINT_firstitem #1#2\xint_bye{\xint_gob_til_xint: #1\xint:\space #1}%
\def\XINT_firstitem_end\xint:{ }%
```

19.22. \xintLastOne

As xintexpr 1.4 uses {c1}{c2}....{cN} storage when gathering comma separated values we need to not handle identically an empty list and a list with an empty item (as the above allows hierarchical structures). But \xintLastItem removed one level of brace pair so it is anadequate for the last() function.

By the way it is logical to interpret «item» as meaning {cj} inclusive of the braces; but legacy xint user manual was not written in this spirit. And thus \xintLastItem did brace stripping, thus we need another name for maintaining backwards compatibility (although the cardinality of users is small).

The \xint: token must not be found (visible) among the item contents.

```
\def\xintLastOne {\romannumeral0\xintlastone }%
\long\def\xintlastone #1%
{%
    \XINT_lastone_loop {}.#1%
    {\xint:\XINT_lastone_loop_enda}{\xint:\XINT_lastone_loop_endb}%
    {\xint:\XINT_lastone_loop_endc}{\xint:\XINT_lastone_loop_endd}%
```

```
    {\xint:\XINT_lastone_loop_ende}{\xint:\XINT_lastone_loop_endf}%
    {\xint:\XINT_lastone_loop_endg}{\xint:\XINT_lastone_loop_endh}\xint_bye
}%
\long\def\XINT_lastone_loop #1.#2#3#4#5#6#7#8#9%
{%
    \xint_gob_til_xint: #9%
        {#8}{#7}{#6}{#5}{#4}{#3}{#2}{#1}\xint:
    \XINT_lastone_loop {{#9}}.%
}%
\long\def\XINT_lastone_loop_enda #1#2\xint_bye{{#1}}%
\long\def\XINT_lastone_loop_endb #1#2#3\xint_bye{{#2}}%
\long\def\XINT_lastone_loop_endc #1#2#3#4\xint_bye{{#3}}%
\long\def\XINT_lastone_loop_endd #1#2#3#4#5\xint_bye{{#4}}%
\long\def\XINT_lastone_loop_ende #1#2#3#4#5#6\xint_bye{{#5}}%
\long\def\XINT_lastone_loop_endf #1#2#3#4#5#6#7\xint_bye{{#6}}%
\long\def\XINT_lastone_loop_endg #1#2#3#4#5#6#7#8\xint_bye{{#7}}%
\long\def\XINT_lastone_loop_endh #1#2#3#4#5#6#7#8#9\xint_bye{ #8}%
```

19.23. \xintFirstOne

For xintexpr 1.4 too. Jan 3, 2020.

This is an experimental macro, don't use it. If input is nil (empty set) it expands to nil, if not it fetches first item and braces it. Fetching will have stripped one brace pair if item was braced to start with, which is the case in non-symbolic xintexpr data objects.

I have not given much thought to this (make it shorter, allow all tokens, (we could first test if empty via combination with \detokenize), etc...) as I need to get xint 1.4 out soon. So in particular attention that the macro assumes the \xint: token is absent from first item of input.

```
\def\xintFirstOne  {\romannumeral0\xintfirstone }%
\long\def\xintfirstone  #1{\XINT_firstone #1{\xint:\XINT_firstone_empty}\xint:}%
\long\def\XINT_firstone #1#2\xint:{\xint_gob_til_xint: #1\xint:{#1}}%
\def\XINT_firstone_empty\xint:#1{ }%
```

19.24. \xintLengthUpTo

Modified at 1.2i (2016/12/13). For use by \xintKeep and \xintTrim (xinttools). The argument N **must be non-negative**.

\xintLengthUpTo{N}{List} produces -0 if length(List)>N, else it returns N-length(List). Hence subtracting it from N always computes min(N,length(List)).

Modified at 1.2j (2016/12/22). Changed ending and interface to core loop.

```
\def\xintLengthUpTo {\romannumeral0\xintlengthupto}%
\long\def\xintlengthupto #1#2%
{%
    \expandafter\XINT_lengthupto_loop
    \the\numexpr#1.#2\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:
         \xint_c_vii\xint_c_vi\xint_c_v\xint_c_iv
         \xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye.%
}%
\def\XINT_lengthupto_loop_a #1%
{%
    \xint_UDsignfork
      #1\XINT_lengthupto_gt
      -\XINT_lengthupto_loop
```

```
    \krof #1%
}%
\long\def\XINT_lengthupto_gt #1\xint_bye.{-0}%
\long\def\XINT_lengthupto_loop #1.#2#3#4#5#6#7#8#9%
{%
    \xint_gob_til_xint: #9\XINT_lengthupto_finish_a\xint:%
    \expandafter\XINT_lengthupto_loop_a\the\numexpr #1-\xint_c_viii.%
}%
\def\XINT_lengthupto_finish_a\xint:\expandafter\XINT_lengthupto_loop_a
    \the\numexpr #1-\xint_c_viii.#2#3#4#5#6#7#8#9%
{%
    \expandafter\XINT_lengthupto_finish_b\the\numexpr #1-#9\xint_bye
}%
\def\XINT_lengthupto_finish_b #1#2.%
{%
    \xint_UDsignfork
       #1{-0}%
        -{ #1#2}%
    \krof
}%
```

19.25. \xintreplicate, \xintReplicate

Modified at 1.2i (2016/12/13). This is cloned from LaTeX3's \prg_replicate:nn, see Joseph's post at

http://tex.stackexchange.com/questions/16189/repeat-command-n-times

I posted there an alternative not using the chained \csname's but it is a bit less efficient (except perhaps for thousands of repetitions). The code in Joseph's post does abs(#1) replications when input #1 is negative and then activates an error triggering macro; here we simply do nothing when #1 is negative.

Usage: \romannumeral\xintreplicate{N}{stuff}

When N is already explicit digits (even N=0, but non-negative) one can call the macro as

\romannumeral\XINT_rep N\endcsname {foo}

to skip the \numexpr.

Modified at 1.4 (2020/01/31). Added \xintReplicate ! The reason I did not before is that the prevailing habits in xint source code was to trigger with \romannumeral0 not \romannumeral which is the lowercased named macros. Thus adding the camelcase one creates a couple \xintReplicate /\xintreplicate not obeying the general mold.

```
\def\xintReplicate{\romannumeral\xintreplicate}%
\def\xintreplicate#1%
   {\expandafter\XINT_replicate\the\numexpr#1\endcsname}%
\def\XINT_replicate #1{\xint_UDsignfork
                         #1\XINT_rep_neg
                          -\XINT_rep
                       \krof #1}%
\long\def\XINT_rep_neg #1\endcsname #2{\xint_c_}%
\def\XINT_rep   #1{\csname XINT_rep_f#1\XINT_rep_a}%
\def\XINT_rep_a #1{\csname XINT_rep_#1\XINT_rep_a}%
\def\XINT_rep_\XINT_rep_a{\endcsname}%
\long\expandafter\def\csname XINT_rep_0\endcsname #1%
    {\endcsname{#1#1#1#1#1#1#1#1#1#1}}%
\long\expandafter\def\csname XINT_rep_1\endcsname #1%
```

```
    {\endcsname{#1#1#1#1#1#1#1#1#1#1}#1}%
\long\expandafter\def\csname XINT_rep_2\endcsname #1%
    {\endcsname{#1#1#1#1#1#1#1#1#1#1}#1#1}%
\long\expandafter\def\csname XINT_rep_3\endcsname #1%
    {\endcsname{#1#1#1#1#1#1#1#1#1#1}#1#1#1}%
\long\expandafter\def\csname XINT_rep_4\endcsname #1%
    {\endcsname{#1#1#1#1#1#1#1#1#1#1}#1#1#1#1}%
\long\expandafter\def\csname XINT_rep_5\endcsname #1%
    {\endcsname{#1#1#1#1#1#1#1#1#1#1}#1#1#1#1#1}%
\long\expandafter\def\csname XINT_rep_6\endcsname #1%
    {\endcsname{#1#1#1#1#1#1#1#1#1#1}#1#1#1#1#1#1}%
\long\expandafter\def\csname XINT_rep_7\endcsname #1%
    {\endcsname{#1#1#1#1#1#1#1#1#1#1}#1#1#1#1#1#1#1}%
\long\expandafter\def\csname XINT_rep_8\endcsname #1%
    {\endcsname{#1#1#1#1#1#1#1#1#1#1}#1#1#1#1#1#1#1#1}%
\long\expandafter\def\csname XINT_rep_9\endcsname #1%
    {\endcsname{#1#1#1#1#1#1#1#1#1#1}#1#1#1#1#1#1#1#1#1}%
\long\expandafter\def\csname XINT_rep_f0\endcsname #1%
    {\xint_c_}%
\long\expandafter\def\csname XINT_rep_f1\endcsname #1%
    {\xint_c_ #1}%
\long\expandafter\def\csname XINT_rep_f2\endcsname #1%
    {\xint_c_ #1#1}%
\long\expandafter\def\csname XINT_rep_f3\endcsname #1%
    {\xint_c_ #1#1#1}%
\long\expandafter\def\csname XINT_rep_f4\endcsname #1%
    {\xint_c_ #1#1#1#1}%
\long\expandafter\def\csname XINT_rep_f5\endcsname #1%
    {\xint_c_ #1#1#1#1#1}%
\long\expandafter\def\csname XINT_rep_f6\endcsname #1%
    {\xint_c_ #1#1#1#1#1#1}%
\long\expandafter\def\csname XINT_rep_f7\endcsname #1%
    {\xint_c_ #1#1#1#1#1#1#1}%
\long\expandafter\def\csname XINT_rep_f8\endcsname #1%
    {\xint_c_ #1#1#1#1#1#1#1#1}%
\long\expandafter\def\csname XINT_rep_f9\endcsname #1%
    {\xint_c_ #1#1#1#1#1#1#1#1#1}%
```

19.26. \xintgobble, \xintGobble

Modified at 1.2i (2016/12/13). I hesitated about allowing as many as 9^6-1=531440 tokens to gobble, but 9^5-1=59058 is too low for playing with long decimal expansions.
Usage: \romannumeral\xintgobble{N}...

Modified at 1.4 (2020/01/31). Added \xintGobble.

```
\def\xintGobble{\romannumeral\xintgobble}%
\def\xintgobble #1%
   {\csname xint_c_\expandafter\XINT_gobble_a\the\numexpr#1.0}%
\def\XINT_gobble #1.{\csname xint_c_\XINT_gobble_a #1.0}%
\def\XINT_gobble_a #1{\xint_gob_til_zero#1\XINT_gobble_d0\XINT_gobble_b#1}%
\def\XINT_gobble_b #1.#2%
  {\expandafter\XINT_gobble_c
      \the\numexpr (#1+\xint_c_v)/\xint_c_ix-\xint_c_i\expandafter.%
```

```
      \the\numexpr #2+\xint_c_i.#1.}%
\def\XINT_gobble_c #1.#2.#3.%
  {\csname XINT_g#2\the\numexpr#3-\xint_c_ix*#1\relax\XINT_gobble_a #1.#2}%
\def\XINT_gobble_d0\XINT_gobble_b0.#1{\endcsname}%
\expandafter\let\csname XINT_g10\endcsname\endcsname
\long\expandafter\def\csname XINT_g11\endcsname#1{\endcsname}%
\long\expandafter\def\csname XINT_g12\endcsname#1#2{\endcsname}%
\long\expandafter\def\csname XINT_g13\endcsname#1#2#3{\endcsname}%
\long\expandafter\def\csname XINT_g14\endcsname#1#2#3#4{\endcsname}%
\long\expandafter\def\csname XINT_g15\endcsname#1#2#3#4#5{\endcsname}%
\long\expandafter\def\csname XINT_g16\endcsname#1#2#3#4#5#6{\endcsname}%
\long\expandafter\def\csname XINT_g17\endcsname#1#2#3#4#5#6#7{\endcsname}%
\long\expandafter\def\csname XINT_g18\endcsname#1#2#3#4#5#6#7#8{\endcsname}%
\expandafter\let\csname XINT_g20\endcsname\endcsname
\long\expandafter\def\csname XINT_g21\endcsname #1#2#3#4#5#6#7#8#9%
 {\endcsname}%
\long\expandafter\edef\csname XINT_g22\endcsname #1#2#3#4#5#6#7#8#9%
 {\expandafter\noexpand\csname XINT_g21\endcsname}%
\long\expandafter\edef\csname XINT_g23\endcsname #1#2#3#4#5#6#7#8#9%
 {\expandafter\noexpand\csname XINT_g22\endcsname}%
\long\expandafter\edef\csname XINT_g24\endcsname #1#2#3#4#5#6#7#8#9%
 {\expandafter\noexpand\csname XINT_g23\endcsname}%
\long\expandafter\edef\csname XINT_g25\endcsname #1#2#3#4#5#6#7#8#9%
 {\expandafter\noexpand\csname XINT_g24\endcsname}%
\long\expandafter\edef\csname XINT_g26\endcsname #1#2#3#4#5#6#7#8#9%
 {\expandafter\noexpand\csname XINT_g25\endcsname}%
\long\expandafter\edef\csname XINT_g27\endcsname #1#2#3#4#5#6#7#8#9%
 {\expandafter\noexpand\csname XINT_g26\endcsname}%
\long\expandafter\edef\csname XINT_g28\endcsname #1#2#3#4#5#6#7#8#9%
 {\expandafter\noexpand\csname XINT_g27\endcsname}%
\expandafter\let\csname XINT_g30\endcsname\endcsname
\long\expandafter\edef\csname XINT_g31\endcsname #1#2#3#4#5#6#7#8#9%
 {\expandafter\noexpand\csname XINT_g28\endcsname}%
\long\expandafter\edef\csname XINT_g32\endcsname #1#2#3#4#5#6#7#8#9%
 {\noexpand\csname XINT_g31\expandafter\noexpand\csname XINT_g28\endcsname}%
\long\expandafter\edef\csname XINT_g33\endcsname #1#2#3#4#5#6#7#8#9%
 {\noexpand\csname XINT_g32\expandafter\noexpand\csname XINT_g28\endcsname}%
\long\expandafter\edef\csname XINT_g34\endcsname #1#2#3#4#5#6#7#8#9%
 {\noexpand\csname XINT_g33\expandafter\noexpand\csname XINT_g28\endcsname}%
\long\expandafter\edef\csname XINT_g35\endcsname #1#2#3#4#5#6#7#8#9%
 {\noexpand\csname XINT_g34\expandafter\noexpand\csname XINT_g28\endcsname}%
\long\expandafter\edef\csname XINT_g36\endcsname #1#2#3#4#5#6#7#8#9%
 {\noexpand\csname XINT_g35\expandafter\noexpand\csname XINT_g28\endcsname}%
\long\expandafter\edef\csname XINT_g37\endcsname #1#2#3#4#5#6#7#8#9%
 {\noexpand\csname XINT_g36\expandafter\noexpand\csname XINT_g28\endcsname}%
\long\expandafter\edef\csname XINT_g38\endcsname #1#2#3#4#5#6#7#8#9%
 {\noexpand\csname XINT_g37\expandafter\noexpand\csname XINT_g28\endcsname}%
\expandafter\let\csname XINT_g40\endcsname\endcsname
\expandafter\edef\csname XINT_g41\endcsname
 {\noexpand\csname XINT_g38\expandafter\noexpand\csname XINT_g31\endcsname}%
\expandafter\edef\csname XINT_g42\endcsname
 {\noexpand\csname XINT_g41\expandafter\noexpand\csname XINT_g41\endcsname}%
```

```
\expandafter\edef\csname XINT_g43\endcsname
 {\noexpand\csname XINT_g42\expandafter\noexpand\csname XINT_g41\endcsname}%
\expandafter\edef\csname XINT_g44\endcsname
 {\noexpand\csname XINT_g43\expandafter\noexpand\csname XINT_g41\endcsname}%
\expandafter\edef\csname XINT_g45\endcsname
 {\noexpand\csname XINT_g44\expandafter\noexpand\csname XINT_g41\endcsname}%
\expandafter\edef\csname XINT_g46\endcsname
 {\noexpand\csname XINT_g45\expandafter\noexpand\csname XINT_g41\endcsname}%
\expandafter\edef\csname XINT_g47\endcsname
 {\noexpand\csname XINT_g46\expandafter\noexpand\csname XINT_g41\endcsname}%
\expandafter\edef\csname XINT_g48\endcsname
 {\noexpand\csname XINT_g47\expandafter\noexpand\csname XINT_g41\endcsname}%
\expandafter\let\csname XINT_g50\endcsname\endcsname
\expandafter\edef\csname XINT_g51\endcsname
 {\noexpand\csname XINT_g48\expandafter\noexpand\csname XINT_g41\endcsname}%
\expandafter\edef\csname XINT_g52\endcsname
 {\noexpand\csname XINT_g51\expandafter\noexpand\csname XINT_g51\endcsname}%
\expandafter\edef\csname XINT_g53\endcsname
 {\noexpand\csname XINT_g52\expandafter\noexpand\csname XINT_g51\endcsname}%
\expandafter\edef\csname XINT_g54\endcsname
 {\noexpand\csname XINT_g53\expandafter\noexpand\csname XINT_g51\endcsname}%
\expandafter\edef\csname XINT_g55\endcsname
 {\noexpand\csname XINT_g54\expandafter\noexpand\csname XINT_g51\endcsname}%
\expandafter\edef\csname XINT_g56\endcsname
 {\noexpand\csname XINT_g55\expandafter\noexpand\csname XINT_g51\endcsname}%
\expandafter\edef\csname XINT_g57\endcsname
 {\noexpand\csname XINT_g56\expandafter\noexpand\csname XINT_g51\endcsname}%
\expandafter\edef\csname XINT_g58\endcsname
 {\noexpand\csname XINT_g57\expandafter\noexpand\csname XINT_g51\endcsname}%
\expandafter\let\csname XINT_g60\endcsname\endcsname
\expandafter\edef\csname XINT_g61\endcsname
 {\noexpand\csname XINT_g58\expandafter\noexpand\csname XINT_g51\endcsname}%
\expandafter\edef\csname XINT_g62\endcsname
 {\noexpand\csname XINT_g61\expandafter\noexpand\csname XINT_g61\endcsname}%
\expandafter\edef\csname XINT_g63\endcsname
 {\noexpand\csname XINT_g62\expandafter\noexpand\csname XINT_g61\endcsname}%
\expandafter\edef\csname XINT_g64\endcsname
 {\noexpand\csname XINT_g63\expandafter\noexpand\csname XINT_g61\endcsname}%
\expandafter\edef\csname XINT_g65\endcsname
 {\noexpand\csname XINT_g64\expandafter\noexpand\csname XINT_g61\endcsname}%
\expandafter\edef\csname XINT_g66\endcsname
 {\noexpand\csname XINT_g65\expandafter\noexpand\csname XINT_g61\endcsname}%
\expandafter\edef\csname XINT_g67\endcsname
 {\noexpand\csname XINT_g66\expandafter\noexpand\csname XINT_g61\endcsname}%
\expandafter\edef\csname XINT_g68\endcsname
 {\noexpand\csname XINT_g67\expandafter\noexpand\csname XINT_g61\endcsname}%
```

19.27. Random number generation

Added at 1.3b (2018/05/18).

We provide a more random version of the (PDF)TEX `\pdfuniformdeviate`. I discusses the worries with the engine primitive with Bruno Le Floch in May 2018. Regarding `\pdfuniformdeviate x`:

1. with x=2^{29} or x=2^{30} the engine primitive produces only even numbers,
2. with x=3*2^{26} the integers produced by the RNG when taken modulo three obey the proportion 1:1:2, not 1:1:1,
3. with x=3*2^{14} there is analogous although weaker non-uniformity of the random integers when taken modulo 3,
4. generally speaking pure powers of two should generate uniform random integers, but when the range is divisible by large powers of two, the non-uniformity may be amplified in surprising ways by modulo operations.

Moreover, two seeds sharing the same low k bits generate sequences of 28-bits integers which are one-to-one identical modulo 2^k!

In order to mitigate the issues commented upon in the user manual, \xintUniformDeviate currently only uses the seven high bits from the underlying random stream, using multiple calls to \pdfuniformdeviate 128. From the Birthday Effect, after about 2^{11} seeds one will likely pick a new one sharing its 22 low bits with an earlier one.

1. but as the final random integer is obtained by additional operations involving the range x (currently a modulo operation), for odd ranges it is more difficult for bit correlations to be seen,
2. anyway as they are only 2^{28} seeds in total, after only 2^{14} seeds it is likely to encounter one already explored, and then random integers are identical, however complicated the RNG's raw output is malaxed, and whatever the target range x. And 2^{14} is only eight times as large as 2^{11}.

It would be nice if the engine provided some user interface for letting its RNG execute a given number of iterations without the overhead of replicated executions of \pdfuniformdeviate. This could help gain entropy and would reduce correlations across series from distinct seeds.

Modified at 1.4n (2025/09/05). Print a warning immediately if no uniformdeviate is available.

Modified at 1.4p (2026/03/01). Do s/\xint_undefined/\XINT_undefined/ for codebase coherence.

Modified at 1.4p (2026/03/01). Rename \xint_texuniformdeviate into \xinttexuniformdeviate, so that it can be used in some examples of the user documentation.

19.27.1. \xinttexuniformdeviate

ATTENTION! with ConTEXt, it will be redefined below as a *macro with an argument*. So one can NOT make code working both with the usual engines/formats and with ConTEXt. The definition we will use with ConTEXt is purely for our internal usage. Saying this because at 1.4p the user manual shows some usages of this \xinttexuniformdeviate, *but this is not a user level command for ConTEXt usage*.

```
\ifdefined\pdfuniformdeviate  \let\xinttexuniformdeviate\pdfuniformdeviate\fi
\ifdefined\uniformdeviate     \let\xinttexuniformdeviate\uniformdeviate   \fi
\ifx\xinttexuniformdeviate\relax\let\xinttexuniformdeviate\XINT_undefined\fi
```

19.27.2. \xinttexuniformdeviate_dgts

Added at 1.4n (2025/09/05). Needed for compatibility with ConTEXt.

One finds a macro \randomnumber (/usr/local/texlive/2025/texmf-dist/tex/context/base/mkxl/supp-ran.mkxl), which takes two arguments. Fortunately it seems that throughout the xint code base, we use \xinttexuniformdeviate always with a chardef or count argument with one sole exception where it is followed with digits terminated by \xint:. In order not to change anything to

current code and to support ConTEXt-LMTX we make here a suitable definition compatible with these use cases.
Warning: I have no idea and will not check now if the RNG has the same issues for the less significant decimal digits as in PDFTeX, thus I don't know if the overhead in the definition below of \xintUniformeDeviate has any rationale in the LMTX context.

MEMO: In the code below, I have not checked yet and do not remember (at time of preparing 1.4n) if it is possible that the #1 fetched by \xinttexuniformdeviate_dgts has some chance to be zero (as I don't remember the details of the construction). Attention that \randomnumber{0}{-1} may output -1.

Modified at 1.4o (2025/09/06). Remove the test whether some uniformdeviate already exists with ConTEXt and use unconditionally \randomnumber without checking if it exists.

```
\ifdefined\contextversion
  \def\xinttexuniformdeviate#1{\randomnumber{0}{#1-1}}%
  \def\xinttexuniformdeviate_dgts#1\xint:{\randomnumber{0}{#1-1}\xint:}%
\else
  \let\xinttexuniformdeviate_dgts\xinttexuniformdeviate
\fi
```

19.27.3. \xintUniformDeviate

For negative #1 the output (as per the doc I wrote in the user manual some years ago and that I trust) ranges from #1+1 to zero inclusive. This is same behaviour as original primitive and the input must be within the TEX bounds. Note that \the\numexpr-0\relax gives 0.

We make 5 calls to the primitive and the user doc says that when I tested the cost was about 13 times the one of the primitive.

```
\ifdefined\xinttexuniformdeviate
    \expandafter\xint_firstoftwo
\else\expandafter\xint_secondoftwo
\fi
{%
  \def\xintUniformDeviate#1%
     {\the\numexpr\expandafter\XINT_uniformdeviate_sgnfork\the\numexpr#1\xint:}%
  \def\XINT_uniformdeviate_sgnfork#1%
  {%
     \if-#1\XINT_uniformdeviate_neg\fi \XINT_uniformdeviate{}#1%
  }%
  \def\XINT_uniformdeviate_neg\fi\XINT_uniformdeviate#1-%
  {%
      \fi-\numexpr\XINT_uniformdeviate\relax
  }%
  \def\XINT_uniformdeviate#1#2\xint:
  {%(
      \expandafter\XINT_uniformdeviate_a\the\numexpr%
                    -\xinttexuniformdeviate\xint_c_ii^vii%
                    -\xint_c_ii^vii*\xinttexuniformdeviate\xint_c_ii^vii%
                    -\xint_c_ii^xiv*\xinttexuniformdeviate\xint_c_ii^vii%
                    -\xint_c_ii^xxi*\xinttexuniformdeviate\xint_c_ii^vii%
                    +\xinttexuniformdeviate_dgts#2\xint:/#2)*#2\xint:+#2\fi\relax#1%
  }%
  \def\XINT_uniformdeviate_a #1\xint:
  {%
      \expandafter\XINT_uniformdeviate_b\the\numexpr#1-(#1%
```

```
  }%
  \def\XINT_uniformdeviate_b#1#2\xint:{#1#2\if-#1}%
}%
{%
  \xintMessage{xintkernel}{Warning}%
  {No \string\uniformdeviate like primitive identified, macros producing^^J%
  \XINT_fourspaces "random numbers" will raise (expandable) errors.}%
  \def\xintUniformDeviate#1%
  {%
      \the\numexpr
      \XINT_expandableerror{(xintkernel) No uniformdeviate primitive!}%
      0\relax
  }%
}%
```

19.27.4. \xintsetrandomseed

Added at 1.4p (2026/03/01). This will remain undocumented at user level, apart from the fact thet some examples in the documentation use it. Contrarily to the timing primitives, it seems LuaTEX (even without LATEX format) has it, so we only need to alias it, idem with XETEX and OpTEX (which even has it under the two names). Regarding ConTEXt, I do not have time to learn to use it and the following appears to work but anyhow its RNG seems to be some altogther different one than the PDFTEX derived from MetaPost one. And using \count255 is surely bad, but well.

```
\ifdefined\contextversion
   \def\xintsetrandomseed{\afterassignment\XINT_ctx_setrandomseed
                          \count255 }%
   \def\XINT_ctx_setrandomseed{\ctxlua{math.randomseed(\the\count255)}}%
\else
\ifdefined\pdfsetrandomseed   \let\xintsetrandomseed\pdfsetrandomseed\fi
\ifdefined\setrandomseed      \let\xintsetrandomseed\setrandomseed\fi
\fi
\ifx\xintsetrandomseed\relax  \let\xintsetrandomseed\XINT_undefined\fi
\XINTrestorecatcodesendinput%
```

20. Package xinttools implementation

Contents

.1 Introduction . . . 272
.2 Catcodes, ε-TEX and reload detection . . 272
.3 Package identification . . . 273
.4 \xintgodef, \xintgoodef, \xintgfdef . 273
.5 \xintRevWithBraces . . . 273
.6 \xintZapFirstSpaces . . . 274
.7 \xintZapLastSpaces . . . 275
.8 \xintZapSpaces . . . 276
.9 \xintZapSpacesB . . . 276
.10 \xintCSVtoList, \xintCSVtoListNonStripped . . . 276
.11 \xintListWithSep . . . 278
.12 \xintNthElt . . . 279
.13 \xintNthOnePy . . . 280
.14 \xintKeep . . . 281
.15 \xintKeepUnbraced . . . 282
.16 \xintTrim . . . 283
.17 \xintTrimUnbraced . . . 285
.18 \xintApply . . . 286
.19 \xintApply:x (WIP, commented-out) . . 286
.20 \xintApplyUnbraced . . . 287
.21 \xintApplyUnbraced:x (WIP, commented-out) . . . 288
.22 \xintZip (WIP, not public) . . . 289
.23 \xintSeq . . . 291
.24 \xintloop, \xintbreakloop, \xintbreakloopanddo, \xintloopskiptonext . . . 294
.25 \xintiloop, \xintiloopindex, \xintbracediloopindex, \xintouteriloopindex, \xintbracedouteriloopindex, \xintbreakiloop, \xintbreakiloopanddo, \xintiloopskiptonext, \xintiloopskipandredo . . 294
.26 \XINT_xflet . . . 295
.27 \xintApplyInline . . . 295
.28 \xintFor, \xintFor*, \xintBreakFor, \xintBreakForAndDo . . . 296
.29 \XINT_forever, \xintintegers, \xintdimensions, \xintrationals . . . 298
.30 \xintForpair, \xintForthree, \xintForfour . . . 300
.31 \xintAssign . . . 302
.32 \xintAssignArray, \xintDigitsOf . . . 303
.33 CSV (non user documented) variants of Length, Keep, Trim, NthElt, Reverse . . 305
.33.1 \xintLength:f:csv . . . 306
.33.2 \xintLengthUpTo:f:csv . . . 306
.33.3 \xintKeep:f:csv . . . 307
.33.4 \xintTrim:f:csv . . . 309
.33.5 \xintNthEltPy:f:csv . . . 311
.33.6 \xintReverse:f:csv . . . 312
.33.7 \xintFirstItem:f:csv . . . 312
.33.8 \xintLastItem:f:csv . . . 313
.33.9 \xintKeep:x:csv . . . 313
.33.10 Public names for the undocumented csv macros: \xintCSVLength, \xintCSVKeep, \xintCSVKeepx, \xintCSVTrim, \xintCSVNthEltPy, \xintCSVReverse, \xintCSVFirstItem, \xintCSVLastItem 314

20.1. Introduction

The package xinttools loads only xintkernel. There is one feature (\xintrationals syntax with \xintFor) which requires the user to explicitly load xintfrac.

Note that xintexpr loads automatically both oxinttools and xintfrac.

20.2. Catcodes, ε-TEX and reload detection

The code for reload detection was initially copied from Heiko Oberdiek's packages, then modified.

The method for catcodes was also initially directly inspired by these packages.

```
\begingroup\catcode61\catcode48\catcode32=10\relax%
  \catcode13=5    % ^^M
  \endlinechar=13 %
  \catcode123=1   % {
  \catcode125=2   % }
  \catcode64=11   % @
  \catcode44=12   % ,
  \catcode46=12   % .
  \catcode58=12   % :
  \catcode94=7    % ^
```

```
  \def\empty{}\def\space{ }\newlinechar10
  \def\z{\endgroup}%
  \expandafter\let\expandafter\x\csname ver@xinttools.sty\endcsname
  \expandafter\let\expandafter\w\csname ver@xintkernel.sty\endcsname
  \expandafter\ifx\csname numexpr\endcsname\relax
    \expandafter\ifx\csname PackageWarningNoLine\endcsname\relax
      \immediate\write128{^^JPackage xinttools Warning:^^J%
                          \space\space\space\space
                          \numexpr not available, aborting input.^^J}%
    \else
      \PackageWarningNoLine{xinttools}{\numexpr not available, aborting input}%
    \fi
    \def\z{\endgroup\endinput}%
  \else
    \ifx\x\relax   % plain-TeX, first loading of xinttools.sty
      \ifx\w\relax % but xintkernel.sty not yet loaded.
         \def\z{\endgroup\input xintkernel.sty\relax}%
      \fi
    \else
      \ifx\x\empty % LaTeX, first loading,
      % variable is initialized, but \ProvidesPackage not yet seen
          \ifx\w\relax % xintkernel.sty not yet loaded.
            \def\z{\endgroup\RequirePackage{xintkernel}}%
          \fi
      \else
        \def\z{\endgroup\endinput}% xinttools already loaded.
      \fi
    \fi
  \fi
\z%
\XINTsetupcatcodes% defined in xintkernel.sty
```

20.3. Package identification

```
\XINT_providespackage
\ProvidesPackage{xinttools}%
  [2026/11/26 v2.0a2 Expandable and non-expandable utilities (JFB)]%
```

\XINT_toks is used in macros such as \xintFor. It is not used elsewhere in the xint bundle.

```
\newtoks\XINT_toks
\xint_firstofone{\let\XINT_sptoken= } %<- space here!
```

20.4. \xintgodef, \xintgoodef, \xintgfdef

Added at 1.09i (2013/12/18). For use in \xintAssign.

```
\def\xintgodef  {\global\xintodef }%
\def\xintgoodef {\global\xintoodef }%
\def\xintgfdef  {\global\xintfdef }%
```

20.5. \xintRevWithBraces

Added at 1.06 (2013/05/07). Makes the expansion of its argument and then reverses the resulting tokens or braced tokens, adding a pair of braces to each (thus, maintaining it when it was already

there.) The reason for \xint:, here and in other locations, is in case #1 expands to nothing, the \romannumeral-`0 must be stopped.

```
\def\xintRevWithBraces          {\romannumeral0\xintrevwithbraces }%
\def\xintRevWithBracesNoExpand {\romannumeral0\xintrevwithbracesnoexpand }%
\long\def\xintrevwithbraces #1%
{%
    \expandafter\XINT_revwbr_loop\expandafter{\expandafter}%
    \romannumeral`&&@#1\xint:\xint:\xint:\xint:%
                       \xint:\xint:\xint:\xint:\xint_bye
}%
\long\def\xintrevwithbracesnoexpand #1%
{%
    \XINT_revwbr_loop {}%
    #1\xint:\xint:\xint:\xint:%
      \xint:\xint:\xint:\xint:\xint_bye
}%
\long\def\XINT_revwbr_loop #1#2#3#4#5#6#7#8#9%
{%
    \xint_gob_til_xint: #9\XINT_revwbr_finish_a\xint:%
    \XINT_revwbr_loop {{#9}{#8}{#7}{#6}{#5}{#4}{#3}{#2}#1}%
}%
\long\def\XINT_revwbr_finish_a\xint:\XINT_revwbr_loop #1#2\xint_bye
{%
    \XINT_revwbr_finish_b #2\R\R\R\R\R\R\R\Z #1%
}%
\def\XINT_revwbr_finish_b #1#2#3#4#5#6#7#8\Z
{%
    \xint_gob_til_R
            #1\XINT_revwbr_finish_c \xint_gobble_viii
            #2\XINT_revwbr_finish_c \xint_gobble_vii
            #3\XINT_revwbr_finish_c \xint_gobble_vi
            #4\XINT_revwbr_finish_c \xint_gobble_v
            #5\XINT_revwbr_finish_c \xint_gobble_iv
            #6\XINT_revwbr_finish_c \xint_gobble_iii
            #7\XINT_revwbr_finish_c \xint_gobble_ii
            \R\XINT_revwbr_finish_c \xint_gobble_i\Z
}%
```

1.1c revisited this old code and improved upon the earlier endings.

```
\def\XINT_revwbr_finish_c#1{%
\def\XINT_revwbr_finish_c##1##2\Z{\expandafter#1##1}%
}\XINT_revwbr_finish_c{ }%
```

20.6. \xintZapFirstSpaces

Added at 1.09f (2013/11/04) [on 2013/11/01].

Modified at 1.1 (2014/10/28). To correct the bug in case of an empty argument, or argument containing only spaces, which had been forgotten in first version. New version is simpler than the initial one. This macro does NOT expand its argument.

```
\def\xintZapFirstSpaces {\romannumeral0\xintzapfirstspaces }%
\def\xintzapfirstspaces#1{\long
\def\xintzapfirstspaces ##1{\XINT_zapbsp_a #1##1\xint:#1#1\xint:}%
}\xintzapfirstspaces{ }%
```

If the original #1 started with a space, the grabbed #1 is empty. Thus _again? will see #1=\xint_bye, and hand over control to _again which will loop back into \XINT_zapbsp_a, with one initial space less. If the original #1 did not start with a space, or was empty, then the #1 below will be a <sptoken>, then an extract of the original #1, not empty and not starting with a space, which contains what was up to the first <sp><sp> present in original #1, or, if none pre-existed, <sptoken> and all of #1 (possibly empty) plus an ending \xint:. The added initial space will stop later the \romannumeral0. No brace stripping is possible. Control is handed over to \XINT_zapbsp_b which strips out the ending \xint:<sp><sp>\xint:

```
\def\XINT_zapbsp_a#1{\long\def\XINT_zapbsp_a ##1#1#1{%
  \XINT_zapbsp_again?##1\xint_bye\XINT_zapbsp_b ##1#1#1}%
}\XINT_zapbsp_a{ }%
\long\def\XINT_zapbsp_again? #1{\xint_bye #1\XINT_zapbsp_again }%
\xint_firstofone{\def\XINT_zapbsp_again\XINT_zapbsp_b} {\XINT_zapbsp_a }%
\long\def\XINT_zapbsp_b #1\xint:#2\xint:{#1}%
```

20.7. \xintZapLastSpaces

Added at 1.09f (2013/11/04) [on 2013/11/01].

```
\def\xintZapLastSpaces {\romannumeral0\xintzaplastspaces }%
\def\xintzaplastspaces#1{\long
\def\xintzaplastspaces ##1{\XINT_zapesp_a {}\empty##1#1#1\xint_bye\xint:}%
}\xintzaplastspaces{ }%
```

The \empty from \xintzaplastspaces is to prevent brace removal in the #2 below. The \expandafter chain removes it.

```
\xint_firstofone {\long\def\XINT_zapesp_a #1#2 } %<- second space here
    {\expandafter\XINT_zapesp_b\expandafter{#2}{#1}}%
```

Notice again an \empty added here. This is in preparation for possibly looping back to \XINT_zapesp_a. If the initial #1 had no <sp><sp>, the stuff however will not loop, because #3 will already be <some spaces>\xint_bye. Notice that this macro fetches all way to the ending \xint:. This looks not very efficient, but how often do we have to strip ending spaces from something which also has inner stretches of _multiple_ space tokens ?;-).

```
\long\def\XINT_zapesp_b #1#2#3\xint:%
    {\XINT_zapesp_end? #3\XINT_zapesp_e {#2#1}\empty #3\xint:}%
```

When we have been over all possible <sp><sp> things, we reach the ending space tokens, and #3 will be a bunch of spaces (possibly none) followed by \xint_bye. So the #1 in _end? will be \xint_bye. In all other cases #1 can not be \xint_bye (assuming naturally this token does nor arise in original input), hence control falls back to \XINT_zapesp_e which will loop back to \XINT_zapesp_a.

```
\long\def\XINT_zapesp_end? #1{\xint_bye #1\XINT_zapesp_end }%
```

We are done. The #1 here has accumulated all the previous material, and is stripped of its ending spaces, if any.

```
\long\def\XINT_zapesp_end\XINT_zapesp_e #1#2\xint:{ #1}%
```

We haven't yet reached the end, so we need to re-inject two space tokens after what we have gotten so far. Then we loop.

```
\def\XINT_zapesp_e#1{%
\long\def\XINT_zapesp_e ##1{\XINT_zapesp_a {##1#1#1}}%
}\XINT_zapesp_e{ }%
```

20.8. \xintZapSpaces

Added at 1.09f (2013/11/04) [on 2013/11/01].

Modified at 1.1 (2014/10/28). It had the same bug as \xintZapFirstSpaces. We in effect do first \xintZapFirstSpaces, then \xintZapLastSpaces.

```
\def\xintZapSpaces {\romannumeral0\xintzapspaces }%
\def\xintzapspaces#1{%
\long\def\xintzapspaces ##1% like \xintZapFirstSpaces.
        {\XINT_zapsp_a #1##1\xint:#1#1\xint:}%
}\xintzapspaces{ }%
\def\XINT_zapsp_a#1{%
\long\def\XINT_zapsp_a ##1#1#1%
        {\XINT_zapsp_again?##1\xint_bye\XINT_zapsp_b##1#1#1}%
}\XINT_zapsp_a{ }%
\long\def\XINT_zapsp_again? #1{\xint_bye #1\XINT_zapsp_again }%
\xint_firstofone{\def\XINT_zapsp_again\XINT_zapsp_b} {\XINT_zapsp_a }%
\xint_firstofone{\def\XINT_zapsp_b} {\XINT_zapsp_c }%
\def\XINT_zapsp_c#1{%
\long\def\XINT_zapsp_c ##1\xint:##2\xint:%
        {\XINT_zapesp_a{}\empty ##1#1#1\xint_bye\xint:}%
}\XINT_zapsp_c{ }%
```

20.9. \xintZapSpacesB

Added at 1.09f (2013/11/04) [on 2013/11/01]. Strips up to one pair of braces (but then does not strip spaces inside).

```
\def\xintZapSpacesB {\romannumeral0\xintzapspacesb }%
\long\def\xintzapspacesb #1{\XINT_zapspb_one? #1\xint:\xint:%
                            \xint_bye\xintzapspaces {#1}}%
\long\def\XINT_zapspb_one? #1#2%
   {\xint_gob_til_xint: #1\XINT_zapspb_onlyspaces\xint:%
    \xint_gob_til_xint: #2\XINT_zapspb_bracedorone\xint:%
    \xint_bye {#1}}%
\def\XINT_zapspb_onlyspaces\xint:%
    \xint_gob_til_xint:\xint:\XINT_zapspb_bracedorone\xint:%
    \xint_bye #1\xint_bye\xintzapspaces #2{ }%
\long\def\XINT_zapspb_bracedorone\xint:%
    \xint_bye #1\xint:\xint_bye\xintzapspaces #2{ #1}%
```

20.10. \xintCSVtoList, \xintCSVtoListNonStripped

Added at 1.06 (2013/05/07). \xintCSVtoList transforms a,b,..,z into {a}{b}...{z}. The comma separated list may be a macro which is first f-expanded. Here, use of \Z (and \R) perfectly safe.

Modified at 1.09f (2013/11/04). Automatically filters items with \xintZapSpacesB to strip away all spaces around commas, and spaces at the start and end of the list. The original is kept as \xintCSVtoListNonStripped, and is faster. But ... it doesn't strip spaces.

ATTENTION: if the input is empty the output contains one item (empty, of course). This means an \xintFor loop always executes at least once the iteration, contrarily to \xintFor*.

```
\def\xintCSVtoList {\romannumeral0\xintcsvtolist }%
\long\def\xintcsvtolist #1{\expandafter\xintApply
```

```
            \expandafter\xintzapspacesb
            \expandafter{\romannumeral0\xintcsvtolistnonstripped{#1}}}%
\def\xintCSVtoListNoExpand {\romannumeral0\xintcsvtolistnoexpand }%
\long\def\xintcsvtolistnoexpand #1{\expandafter\xintApply
            \expandafter\xintzapspacesb
            \expandafter{\romannumeral0\xintcsvtolistnonstrippednoexpand{#1}}}%
\def\xintCSVtoListNonStripped {\romannumeral0\xintcsvtolistnonstripped }%
\def\xintCSVtoListNonStrippedNoExpand
          {\romannumeral0\xintcsvtolistnonstrippednoexpand }%
\long\def\xintcsvtolistnonstripped #1%
{%
    \expandafter\XINT_csvtol_loop_a\expandafter
    {\expandafter}\romannumeral`&&@#1%
        ,\xint_bye,\xint_bye,\xint_bye,\xint_bye
        ,\xint_bye,\xint_bye,\xint_bye,\xint_bye,\Z
}%
\long\def\xintcsvtolistnonstrippednoexpand #1%
{%
    \XINT_csvtol_loop_a
    {}#1,\xint_bye,\xint_bye,\xint_bye,\xint_bye
        ,\xint_bye,\xint_bye,\xint_bye,\xint_bye,\Z
}%
\long\def\XINT_csvtol_loop_a #1#2,#3,#4,#5,#6,#7,#8,#9,%
{%
    \xint_bye #9\XINT_csvtol_finish_a\xint_bye
    \XINT_csvtol_loop_b {#1}{{#2}{#3}{#4}{#5}{#6}{#7}{#8}{#9}}%
}%
\long\def\XINT_csvtol_loop_b #1#2{\XINT_csvtol_loop_a {#1#2}}%
\long\def\XINT_csvtol_finish_a\xint_bye\XINT_csvtol_loop_b #1#2#3\Z
{%
    \XINT_csvtol_finish_b #3\R,\R,\R,\R,\R,\R,\Z #2{#1}%
}%
```

1.1c revisits this old code and improves upon the earlier endings. But as the _d.. macros have already nine parameters, I needed the \expandafter and \xint_gob_til_Z in finish_b (compare \XINT_keep_endb, or also \XINT_RQ_end_b).

```
\def\XINT_csvtol_finish_b #1,#2,#3,#4,#5,#6,#7,#8\Z
{%
    \xint_gob_til_R
            #1\expandafter\XINT_csvtol_finish_dviii\xint_gob_til_Z
            #2\expandafter\XINT_csvtol_finish_dvii \xint_gob_til_Z
            #3\expandafter\XINT_csvtol_finish_dvi  \xint_gob_til_Z
            #4\expandafter\XINT_csvtol_finish_dv   \xint_gob_til_Z
            #5\expandafter\XINT_csvtol_finish_div  \xint_gob_til_Z
            #6\expandafter\XINT_csvtol_finish_diii \xint_gob_til_Z
            #7\expandafter\XINT_csvtol_finish_dii  \xint_gob_til_Z
            \R\XINT_csvtol_finish_di \Z
}%
\long\def\XINT_csvtol_finish_dviii #1#2#3#4#5#6#7#8#9{ #9}%
\long\def\XINT_csvtol_finish_dvii  #1#2#3#4#5#6#7#8#9{ #9{#1}}%
\long\def\XINT_csvtol_finish_dvi   #1#2#3#4#5#6#7#8#9{ #9{#1}{#2}}%
\long\def\XINT_csvtol_finish_dv    #1#2#3#4#5#6#7#8#9{ #9{#1}{#2}{#3}}%
\long\def\XINT_csvtol_finish_div   #1#2#3#4#5#6#7#8#9{ #9{#1}{#2}{#3}{#4}}%
```

```
\long\def\XINT_csvtol_finish_diii  #1#2#3#4#5#6#7#8#9{ #9{#1}{#2}{#3}{#4}{#5}}%
\long\def\XINT_csvtol_finish_dii   #1#2#3#4#5#6#7#8#9%
                                          { #9{#1}{#2}{#3}{#4}{#5}{#6}}%
\long\def\XINT_csvtol_finish_di\Z  #1#2#3#4#5#6#7#8#9%
                                          { #9{#1}{#2}{#3}{#4}{#5}{#6}{#7}}%
```

20.11. \xintListWithSep

Added at 1.04 (2013/04/25). \xintListWithSep {\sep}{{a}{b}...{z}} returns a \sep b \sep ....\sep z. It f-expands its second argument. The 'sep' may be \par's: the macro \xintlistwithsep etc... are all declared long. 'sep' does not have to be a single token. It is not expanded. The "list" argument may be empty.
\xintListWithSepNoExpand does not f-expand its second argument.

Modified at 1.2p (2017/12/05). This venerable macro from 1.04 remained unchanged for a long time and was finally refactored at 1.2p for increased speed. Tests done with a list of identical {\x} items and a sep of \z demonstrated a speed increase of about:

- 3x for 30 items,
- 4.5x for 100 items,
- 7.5x--8x for 1000 items.

```
\def\xintListWithSep         {\romannumeral0\xintlistwithsep }%
\def\xintListWithSepNoExpand {\romannumeral0\xintlistwithsepnoexpand }%
\long\def\xintlistwithsep #1#2%
    {\expandafter\XINT_lws\expandafter {\romannumeral`&&@#2}{#1}}%
\long\def\xintlistwithsepnoexpand #1#2%
{%
    \XINT_lws_loop_a {#1}#2{\xint_bye\XINT_lws_e_vi}%
       {\xint_bye\XINT_lws_e_v}{\xint_bye\XINT_lws_e_iv}%
       {\xint_bye\XINT_lws_e_iii}{\xint_bye\XINT_lws_e_ii}%
       {\xint_bye\XINT_lws_e_i}{\xint_bye\XINT_lws_e}%
       {\xint_bye\expandafter\space}\xint_bye
}%
\long\def\XINT_lws #1#2%
{%
    \XINT_lws_loop_a {#2}#1{\xint_bye\XINT_lws_e_vi}%
       {\xint_bye\XINT_lws_e_v}{\xint_bye\XINT_lws_e_iv}%
       {\xint_bye\XINT_lws_e_iii}{\xint_bye\XINT_lws_e_ii}%
       {\xint_bye\XINT_lws_e_i}{\xint_bye\XINT_lws_e}%
       {\xint_bye\expandafter\space}\xint_bye
}%
\long\def\XINT_lws_loop_a #1#2#3#4#5#6#7#8#9%
{%
    \xint_bye #9\xint_bye
    \XINT_lws_loop_b {#1}{#2}{#3}{#4}{#5}{#6}{#7}{#8}{#9}%
}%
\long\def\XINT_lws_loop_b #1#2#3#4#5#6#7#8#9%
{%
    \XINT_lws_loop_a {#1}{#2#1#3#1#4#1#5#1#6#1#7#1#8#1#9}%
}%
\long\def\XINT_lws_e_vi\xint_bye\XINT_lws_loop_b #1#2#3#4#5#6#7#8#9\xint_bye
    { #2#1#3#1#4#1#5#1#6#1#7#1#8}%
\long\def\XINT_lws_e_v\xint_bye\XINT_lws_loop_b #1#2#3#4#5#6#7#8\xint_bye
    { #2#1#3#1#4#1#5#1#6#1#7}%
```

```
\long\def\XINT_lws_e_iv\xint_bye\XINT_lws_loop_b #1#2#3#4#5#6#7\xint_bye
    { #2#1#3#1#4#1#5#1#6}%
\long\def\XINT_lws_e_iii\xint_bye\XINT_lws_loop_b #1#2#3#4#5#6\xint_bye
    { #2#1#3#1#4#1#5}%
\long\def\XINT_lws_e_ii\xint_bye\XINT_lws_loop_b #1#2#3#4#5\xint_bye
    { #2#1#3#1#4}%
\long\def\XINT_lws_e_i\xint_bye\XINT_lws_loop_b #1#2#3#4\xint_bye
    { #2#1#3}%
\long\def\XINT_lws_e\xint_bye\XINT_lws_loop_b #1#2#3\xint_bye
    { #2}%
```

20.12. \xintNthElt

Added at 1.06 (2013/05/07).

Modified at 1.2j (2016/12/22). Last refactored in 1.2j.

\xintNthElt {i}{List} returns the i th item from List (one pair of braces removed). The list is first f-expanded. The \xintNthEltNoExpand does no expansion of its second argument. Both variants expand i inside \numexpr.

With i = 0, the number of items is returned using \xintLength but with the List argument f-expanded first.

Negative values return the |i|th element from the end.

When i is out of range, an empty value is returned.

```
\def\xintNthElt          {\romannumeral0\xintnthelt }%
\def\xintNthEltNoExpand {\romannumeral0\xintntheltnoexpand }%
\long\def\xintnthelt #1#2{\expandafter\XINT_nthelt_a\the\numexpr #1\expandafter.%
                         \expandafter{\romannumeral`&&@#2}}%
\def\xintntheltnoexpand #1{\expandafter\XINT_nthelt_a\the\numexpr #1.}%
\def\XINT_nthelt_a #1%
{%
    \xint_UDzerominusfork
        #1-\XINT_nthelt_zero
        0#1\XINT_nthelt_neg
         0-{\XINT_nthelt_pos #1}%
    \krof
}%
\def\XINT_nthelt_zero #1.{\xintlength }%
\long\def\XINT_nthelt_neg #1.#2%
{%
    \expandafter\XINT_nthelt_neg_a\the\numexpr\xint_c_i+\XINT_length_loop
    #2\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:
      \xint_c_viii\xint_c_vii\xint_c_vi\xint_c_v
      \xint_c_iv\xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye
    -#1.#2\xint_bye
}%
\def\XINT_nthelt_neg_a #1%
{%
    \xint_UDzerominusfork
        #1-\xint_stop_afterbye
        0#1\xint_stop_afterbye
         0-{}%
    \krof
    \expandafter\XINT_nthelt_neg_b
```

```
    \romannumeral\expandafter\XINT_gobble\the\numexpr-\xint_c_i+#1%
}%
\long\def\XINT_nthelt_neg_b #1#2\xint_bye{ #1}%
\long\def\XINT_nthelt_pos #1.#2%
{%
    \expandafter\XINT_nthelt_pos_done
    \romannumeral0\expandafter\XINT_trim_loop\the\numexpr#1-\xint_c_x.%
     #2\xint:\xint:\xint:\xint:\xint:%
       \xint:\xint:\xint:\xint:\xint:%
    \xint_bye
}%
\def\XINT_nthelt_pos_done #1{%
\long\def\XINT_nthelt_pos_done ##1##2\xint_bye{%
  \xint_gob_til_xint:##1\expandafter#1\xint_gobble_ii\xint:#1##1}%
}\XINT_nthelt_pos_done{ }%
```

20.13. \xintNthOnePy

Added at 1.4 (2020/01/31). See relevant code comments in xintexpr.

```
\def\xintNthOnePy          {\romannumeral0\xintnthonepy }%
\def\xintNthOnePyNoExpand {\romannumeral0\xintnthonepynoexpand }%
\long\def\xintnthonepy #1#2{\expandafter\XINT_nthonepy_a\the\numexpr #1\expandafter.%
                           \expandafter{\romannumeral`&&@#2}}%
\def\xintnthonepynoexpand #1{\expandafter\XINT_nthonepy_a\the\numexpr #1.}%
\def\XINT_nthonepy_a #1%
{%
    \xint_UDsignfork
        #1\XINT_nthonepy_neg
         -{\XINT_nthonepy_nonneg #1}%
    \krof
}%
\long\def\XINT_nthonepy_neg #1.#2%
{%
    \expandafter\XINT_nthonepy_neg_a\the\numexpr\xint_c_i+\XINT_length_loop
    #2\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:
      \xint_c_viii\xint_c_vii\xint_c_vi\xint_c_v
      \xint_c_iv\xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye
    -#1.#2\xint_bye
}%
\def\XINT_nthonepy_neg_a #1%
{%
    \xint_UDzerominusfork
        #1-\xint_stop_afterbye
        0#1\xint_stop_afterbye
         0-{}%
    \krof
    \expandafter\XINT_nthonepy_neg_b
    \romannumeral\expandafter\XINT_gobble\the\numexpr-\xint_c_i+#1%
}%
\long\def\XINT_nthonepy_neg_b #1#2\xint_bye{{#1}}%
\long\def\XINT_nthonepy_nonneg #1.#2%
{%
```

```
    \expandafter\XINT_nthonepy_nonneg_done
    \romannumeral0\expandafter\XINT_trim_loop\the\numexpr#1-\xint_c_ix.%
     #2\xint:\xint:\xint:\xint:\xint:%
       \xint:\xint:\xint:\xint:\xint:%
    \xint_bye
}%
\def\XINT_nthonepy_nonneg_done #1{%
\long\def\XINT_nthonepy_nonneg_done ##1##2\xint_bye{%
  \xint_gob_til_xint:##1\expandafter#1\xint_gobble_ii\xint:{##1}}%
}\XINT_nthonepy_nonneg_done{ }%
```

20.14. \xintKeep

Added at 1.09m (2014/02/26). \xintKeep{i}{L} f-expands its second argument L. It then grabs the first i items from L and discards the rest.

ATTENTION: **each such kept item is returned inside a brace pair** Use \xintKeepUnbraced to avoid that.

For i equal or larger to the number N of items in (expanded) L, the full L is returned (with braced items). For i=0, the macro returns an empty output. For i<0, the macro discards the first N-|i| items. No brace pairs added to the remaining items. For i is less or equal to -N, the full L is returned (with no braces added.)

\xintKeepNoExpand does not expand the L argument.

Modified at 1.2i (2016/12/13). Prior to 1.2i the code proceeded along a loop with no pre-computation of the length of L, for the i>0 case. The faster 1.2i version takes advantage of novel \xintLengthUpTo from xintkernel.sty.

```
\def\xintKeep          {\romannumeral0\xintkeep }%
\def\xintKeepNoExpand {\romannumeral0\xintkeepnoexpand }%
\long\def\xintkeep #1#2{\expandafter\XINT_keep_a\the\numexpr #1\expandafter.%
                        \expandafter{\romannumeral`&&@#2}}%
\def\xintkeepnoexpand #1{\expandafter\XINT_keep_a\the\numexpr #1.}%
\def\XINT_keep_a #1%
{%
    \xint_UDzerominusfork
        #1-\XINT_keep_keepnone
        0#1\XINT_keep_neg
         0-{\XINT_keep_pos #1}%
    \krof
}%
\long\def\XINT_keep_keepnone .#1{ }%
\long\def\XINT_keep_neg #1.#2%
{%
    \expandafter\XINT_keep_neg_a\the\numexpr
    #1-\numexpr\XINT_length_loop
    #2\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:
      \xint_c_viii\xint_c_vii\xint_c_vi\xint_c_v
      \xint_c_iv\xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye.#2%
}%
\def\XINT_keep_neg_a #1%
{%
    \xint_UDsignfork
        #1{\expandafter\space\romannumeral\XINT_gobble}%
         -\XINT_keep_keepall
```

```
    \krof
}%
\def\XINT_keep_keepall #1.{ }%
\long\def\XINT_keep_pos #1.#2%
{%
    \expandafter\XINT_keep_loop
    \the\numexpr#1-\XINT_lengthupto_loop
    #1.#2\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:
         \xint_c_vii\xint_c_vi\xint_c_v\xint_c_iv
         \xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye.%
    -\xint_c_viii.{}#2\xint_bye%
}%
\def\XINT_keep_loop #1#2.%
{%
    \xint_gob_til_minus#1\XINT_keep_loop_end-%
    \expandafter\XINT_keep_loop
    \the\numexpr#1#2-\xint_c_viii\expandafter.\XINT_keep_loop_pickeight
}%
\long\def\XINT_keep_loop_pickeight
     #1#2#3#4#5#6#7#8#9{{#1{#2}{#3}{#4}{#5}{#6}{#7}{#8}{#9}}}%
\def\XINT_keep_loop_end-\expandafter\XINT_keep_loop
    \the\numexpr-#1-\xint_c_viii\expandafter.\XINT_keep_loop_pickeight
    {\csname XINT_keep_end#1\endcsname}%
\long\expandafter\def\csname XINT_keep_end1\endcsname
   #1#2#3#4#5#6#7#8#9\xint_bye { #1{#2}{#3}{#4}{#5}{#6}{#7}{#8}}%
\long\expandafter\def\csname XINT_keep_end2\endcsname
   #1#2#3#4#5#6#7#8\xint_bye { #1{#2}{#3}{#4}{#5}{#6}{#7}}%
\long\expandafter\def\csname XINT_keep_end3\endcsname
   #1#2#3#4#5#6#7\xint_bye { #1{#2}{#3}{#4}{#5}{#6}}%
\long\expandafter\def\csname XINT_keep_end4\endcsname
   #1#2#3#4#5#6\xint_bye { #1{#2}{#3}{#4}{#5}}%
\long\expandafter\def\csname XINT_keep_end5\endcsname
   #1#2#3#4#5\xint_bye { #1{#2}{#3}{#4}}%
\long\expandafter\def\csname XINT_keep_end6\endcsname
   #1#2#3#4\xint_bye { #1{#2}{#3}}%
\long\expandafter\def\csname XINT_keep_end7\endcsname
   #1#2#3\xint_bye { #1{#2}}%
\long\expandafter\def\csname XINT_keep_end8\endcsname
   #1#2\xint_bye { #1}%
```

20.15. \xintKeepUnbraced

Added at 1.2a (2015/10/19). Same as \xintKeep but will *not* add (or maintain) brace pairs around the kept items when length(L)>i>0.

The name may cause a mis-understanding: for i<0, (i.e. keeping only trailing items), there is no brace removal at all happening.

Modified at 1.2i (2016/12/13). As \xintKeep.

```
\def\xintKeepUnbraced         {\romannumeral0\xintkeepunbraced }%
\def\xintKeepUnbracedNoExpand {\romannumeral0\xintkeepunbracednoexpand }%
\long\def\xintkeepunbraced #1#2%
    {\expandafter\XINT_keepunbr_a\the\numexpr #1\expandafter.%
                          \expandafter{\romannumeral`&&@#2}}%
```

```
\def\xintkeepunbracednoexpand #1%
   {\expandafter\XINT_keepunbr_a\the\numexpr #1.}%
\def\XINT_keepunbr_a #1%
{%
    \xint_UDzerominusfork
        #1-\XINT_keep_keepnone
        0#1\XINT_keep_neg
         0-{\XINT_keepunbr_pos #1}%
    \krof
}%
\long\def\XINT_keepunbr_pos #1.#2%
{%
    \expandafter\XINT_keepunbr_loop
    \the\numexpr#1-\XINT_lengthupto_loop
    #1.#2\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:
         \xint_c_vii\xint_c_vi\xint_c_v\xint_c_iv
         \xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye.%
    -\xint_c_viii.{}#2\xint_bye%
}%
\def\XINT_keepunbr_loop #1#2.%
{%
    \xint_gob_til_minus#1\XINT_keepunbr_loop_end-%
    \expandafter\XINT_keepunbr_loop
    \the\numexpr#1#2-\xint_c_viii\expandafter.\XINT_keepunbr_loop_pickeight
}%
\long\def\XINT_keepunbr_loop_pickeight
     #1#2#3#4#5#6#7#8#9{{#1#2#3#4#5#6#7#8#9}}%
\def\XINT_keepunbr_loop_end-\expandafter\XINT_keepunbr_loop
    \the\numexpr-#1-\xint_c_viii\expandafter.\XINT_keepunbr_loop_pickeight
    {\csname XINT_keepunbr_end#1\endcsname}%
\long\expandafter\def\csname XINT_keepunbr_end1\endcsname
   #1#2#3#4#5#6#7#8#9\xint_bye { #1#2#3#4#5#6#7#8}%
\long\expandafter\def\csname XINT_keepunbr_end2\endcsname
   #1#2#3#4#5#6#7#8\xint_bye { #1#2#3#4#5#6#7}%
\long\expandafter\def\csname XINT_keepunbr_end3\endcsname
   #1#2#3#4#5#6#7\xint_bye { #1#2#3#4#5#6}%
\long\expandafter\def\csname XINT_keepunbr_end4\endcsname
   #1#2#3#4#5#6\xint_bye { #1#2#3#4#5}%
\long\expandafter\def\csname XINT_keepunbr_end5\endcsname
   #1#2#3#4#5\xint_bye { #1#2#3#4}%
\long\expandafter\def\csname XINT_keepunbr_end6\endcsname
   #1#2#3#4\xint_bye { #1#2#3}%
\long\expandafter\def\csname XINT_keepunbr_end7\endcsname
   #1#2#3\xint_bye { #1#2}%
\long\expandafter\def\csname XINT_keepunbr_end8\endcsname
   #1#2\xint_bye { #1}%
```

20.16. \xintTrim

Added at 1.09m (2014/02/26). `\xintTrim{i}{L}` f-expands its second argument L. It then removes the first i items from L and keeps the rest. For i equal or larger to the number N of items in (expanded) L, the macro returns an empty output. For i=0, the original (expanded) L is returned.

For i<0, the macro proceeds from the tail. It thus removes the last |i| items, i.e. it keeps the first N-|i| items. For |i|>= N, the empty list is returned.
\xintTrimNoExpand does not expand the L argument.

Modified at 1.2i (2016/12/13). Speed improvements for i<0 branch (which hands over to \xintKeep).
Speed improvements with 1.2j for i>0 branch which gobbles items nine by nine despite not knowing in advance if it will go too far.

```
\def\xintTrim          {\romannumeral0\xinttrim }%
\def\xintTrimNoExpand {\romannumeral0\xinttrimnoexpand }%
\long\def\xinttrim #1#2{\expandafter\XINT_trim_a\the\numexpr #1\expandafter.%
                        \expandafter{\romannumeral`&&@#2}}%
\def\xinttrimnoexpand #1{\expandafter\XINT_trim_a\the\numexpr #1.}%
\def\XINT_trim_a #1%
{%
    \xint_UDzerominusfork
        #1-\XINT_trim_trimnone
        0#1\XINT_trim_neg
         0-{\XINT_trim_pos #1}%
    \krof
}%
\long\def\XINT_trim_trimnone .#1{ #1}%
\long\def\XINT_trim_neg #1.#2%
{%
    \expandafter\XINT_trim_neg_a\the\numexpr
    #1-\numexpr\XINT_length_loop
    #2\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:
      \xint_c_viii\xint_c_vii\xint_c_vi\xint_c_v
      \xint_c_iv\xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye
    .{}#2\xint_bye
}%
\def\XINT_trim_neg_a #1%
{%
    \xint_UDsignfork
        #1{\expandafter\XINT_keep_loop\the\numexpr-\xint_c_viii+}%
         -\XINT_trim_trimall
    \krof
}%
\def\XINT_trim_trimall#1{%
\def\XINT_trim_trimall {\expandafter#1\xint_bye}%
}\XINT_trim_trimall{ }%
```

This branch doesn't pre-evaluate the length of the list argument. Redone again for 1.2j, manages to trim nine by nine. Some non optimal looking aspect of the code is for allowing sharing with \xintNthElt.

```
\long\def\XINT_trim_pos #1.#2%
{%
    \expandafter\XINT_trim_pos_done\expandafter\space
    \romannumeral0\expandafter\XINT_trim_loop\the\numexpr#1-\xint_c_ix.%
     #2\xint:\xint:\xint:\xint:\xint:%
       \xint:\xint:\xint:\xint:\xint:%
    \xint_bye
}%
\def\XINT_trim_loop #1#2.%
{%
```

```
    \xint_gob_til_minus#1\XINT_trim_finish-%
    \expandafter\XINT_trim_loop\the\numexpr#1#2\XINT_trim_loop_trimnine
}%
\long\def\XINT_trim_loop_trimnine #1#2#3#4#5#6#7#8#9%
{%
    \xint_gob_til_xint: #9\XINT_trim_toofew\xint:-\xint_c_ix.%
}%
\def\XINT_trim_toofew\xint:{*\xint_c_}%
\def\XINT_trim_finish#1{%
\def\XINT_trim_finish-%
    \expandafter\XINT_trim_loop\the\numexpr-##1\XINT_trim_loop_trimnine
{%
    \expandafter\expandafter\expandafter#1%
    \csname xint_gobble_\romannumeral\numexpr\xint_c_ix-##1\endcsname
}}\XINT_trim_finish{ }%
\long\def\XINT_trim_pos_done #1\xint:#2\xint_bye {#1}%
```

20.17. \xintTrimUnbraced

Added at 1.2a (2015/10/19).

Modified at 1.2i (2016/12/13). As \xintTrim.

```
\def\xintTrimUnbraced          {\romannumeral0\xinttrimunbraced }%
\def\xintTrimUnbracedNoExpand {\romannumeral0\xinttrimunbracednoexpand }%
\long\def\xinttrimunbraced #1#2%
    {\expandafter\XINT_trimunbr_a\the\numexpr #1\expandafter.%
                          \expandafter{\romannumeral`&&@#2}}%
\def\xinttrimunbracednoexpand #1%
    {\expandafter\XINT_trimunbr_a\the\numexpr #1.}%
\def\XINT_trimunbr_a #1%
{%
    \xint_UDzerominusfork
        #1-\XINT_trim_trimnone
        0#1\XINT_trimunbr_neg
         0-{\XINT_trim_pos #1}%
    \krof
}%
\long\def\XINT_trimunbr_neg #1.#2%
{%
    \expandafter\XINT_trimunbr_neg_a\the\numexpr
    #1-\numexpr\XINT_length_loop
    #2\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:
      \xint_c_viii\xint_c_vii\xint_c_vi\xint_c_v
      \xint_c_iv\xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye
    .{}#2\xint_bye
}%
\def\XINT_trimunbr_neg_a #1%
{%
    \xint_UDsignfork
        #1{\expandafter\XINT_keepunbr_loop\the\numexpr-\xint_c_viii+}%
         -\XINT_trim_trimall
    \krof
}%
```

20.18. \xintApply

Added at 1.04 (2013/04/25). \xintApply {\macro}{{a}{b}...{z}} returns {\macro{a}}...{\macro{b}} where each instance of \macro is f-expanded. The list itself is first f-expanded and may thus be a macro.

```
\def\xintApply          {\romannumeral0\xintapply }%
\def\xintApplyNoExpand {\romannumeral0\xintapplynoexpand }%
\long\def\xintapply #1#2%
{%
    \expandafter\XINT_apply\expandafter {\romannumeral`&&@#2}%
    {#1}%
}%
\long\def\XINT_apply #1#2{\XINT_apply_loop_a {}{#2}#1\xint_bye }%
\long\def\xintapplynoexpand #1#2{\XINT_apply_loop_a {}{#1}#2\xint_bye }%
\long\def\XINT_apply_loop_a #1#2#3%
{%
    \xint_bye #3\XINT_apply_end\xint_bye
    \expandafter
    \XINT_apply_loop_b
    \expandafter {\romannumeral`&&@#2{#3}}{#1}{#2}%
}%
\long\def\XINT_apply_loop_b #1#2{\XINT_apply_loop_a {#2{#1}}}%
\long\def\XINT_apply_end\xint_bye\expandafter\XINT_apply_loop_b
    \expandafter #1#2#3{ #2}%
```

20.19. \xintApply:x (WIP, commented-out)

Added at 1.4 (2020/01/31) [on 2020/01/27]. For usage in the NumPy-like slicing routines. Well, actually, in the end I sticked with old-fashioned (quadratic cost) \xintApply for 1.4 2020/01/31 release. See comments there.

(Comments mainly from 2020/01/27, but on 2020/02/24 I comment out the code and add an alternative)

To expand in \expanded context, and does not need to do any expansion of its second argument.

This uses techniques I had developed for 1.2i/1.2j Keep, Trim, Length, LastItem like macros, and I should revamp venerable \xintApply probably too. But the latter f-expandability (if it does not have \expanded at disposal) complicates significantly matters as it has to store material and release at very end.

Here it is simpler and I am doing it quickly as I really want to release 1.4. The \xint: token should not be located in looped over items. I could use something more exotic like the null char with catcode 3...

```
\long\def\xintApply:x #1#2%
{%
    \XINT_apply:x_loop {#1}#2%
    {\xint:\XINT_apply:x_loop_enda}{\xint:\XINT_apply:x_loop_endb}%
    {\xint:\XINT_apply:x_loop_endc}{\xint:\XINT_apply:x_loop_endd}%
    {\xint:\XINT_apply:x_loop_ende}{\xint:\XINT_apply:x_loop_endf}%
    {\xint:\XINT_apply:x_loop_endg}{\xint:\XINT_apply:x_loop_endh}\xint_bye
}%
\long\def\XINT_apply:x_loop #1#2#3#4#5#6#7#8#9%
{%
    \xint_gob_til_xint: #9\xint:
    {#1{#2}}{#1{#3}}{#1{#4}}{#1{#5}}{#1{#6}}{#1{#7}}{#1{#8}}{#1{#9}}%
```

```
        \XINT_apply:x_loop {#1}%
    }%
    \long\def\XINT_apply:x_loop_endh\xint: #1\xint_bye{}%
    \long\def\XINT_apply:x_loop_endg\xint: #1#2\xint_bye{{#1}}%
    \long\def\XINT_apply:x_loop_endf\xint: #1#2#3\xint_bye{{#1}{#2}}%
    \long\def\XINT_apply:x_loop_ende\xint: #1#2#3#4\xint_bye{{#1}{#2}{#3}}%
    \long\def\XINT_apply:x_loop_endd\xint: #1#2#3#4#5\xint_bye{{#1}{#2}{#3}{#4}}%
    \long\def\XINT_apply:x_loop_endc\xint: #1#2#3#4#5#6\xint_bye{{#1}{#2}{#3}{#4}{#5}}%
    \long\def\XINT_apply:x_loop_endb\xint: #1#2#3#4#5#6#7\xint_bye{{#1}{#2}{#3}{#4}{#5}{#6}}%
    \long\def\XINT_apply:x_loop_enda\xint: #1#2#3#4#5#6#7#8\xint_bye{{#1}{#2}{#3}{#4}{#5}{#6}{#7}}%
For small number of items gain with respect to \xintApply is little if any (might even be a loss).
  Picking one by one is possibly better for small number of items.  Like this for example, the
natural simple minded thing:
  \long\def\xintApply:x #1#2%
    {%
        \XINT_apply:x_loop {#1}#2\xint_bye\xint_bye
    }%
    \long\def\XINT_apply:x_loop #1#2%
    {%
        \xint_bye #2\xint_bye {#1{#2}}%
        \XINT_apply:x_loop {#1}%
    }%
Some variant on 2020/02/24
    \long\def\xint_Bbye#1\xint_Bye{}%
    \long\def\xintApply:x #1#2%
    {%
        \XINT_apply:x_loop {#1}#2%
        {\xint_bye}{\xint_bye}{\xint_bye}{\xint_bye}%
        {\xint_bye}{\xint_bye}{\xint_bye}{\xint_bye}\xint_bye
    }%
    \long\def\XINT_apply:x_loop #1#2#3#4#5#6#7#8#9%
    {%
        \xint_Bye #2\xint_bye {#1{#2}}%
        \xint_Bye #3\xint_bye {#1{#3}}%
        \xint_Bye #4\xint_bye {#1{#4}}%
        \xint_Bye #5\xint_bye {#1{#5}}%
        \xint_Bye #6\xint_bye {#1{#6}}%
        \xint_Bye #7\xint_bye {#1{#7}}%
        \xint_Bye #8\xint_bye {#1{#8}}%
        \xint_Bye #9\xint_bye {#1{#9}}%
        \XINT_apply:x_loop {#1}%
    }%
```

20.20. \xintApplyUnbraced

Added at 1.06b (2013/05/14). \xintApplyUnbraced {\macro}{{a}{b}...{z}} returns \macro{a}...\macro{z} where each instance of \macro is f-expanded using \romannumeral-`0. The second argument may be a macro as it is itself also f-expanded. No braces are added: this allows for example a non-expandable \def in \macro, without having to do \gdef.

```
\def\xintApplyUnbraced {\romannumeral0\xintapplyunbraced }%
\def\xintApplyUnbracedNoExpand {\romannumeral0\xintapplyunbracednoexpand }%
\long\def\xintapplyunbraced #1#2%
```

```
{%
    \expandafter\XINT_applyunbr\expandafter {\romannumeral`&&@#2}%
    {#1}%
}%
\long\def\XINT_applyunbr #1#2{\XINT_applyunbr_loop_a {}{#2}#1\xint_bye }%
\long\def\xintapplyunbracednoexpand #1#2%
   {\XINT_applyunbr_loop_a {}{#1}#2\xint_bye }%
\long\def\XINT_applyunbr_loop_a #1#2#3%
{%
    \xint_bye #3\XINT_applyunbr_end\xint_bye
    \expandafter\XINT_applyunbr_loop_b
    \expandafter {\romannumeral`&&@#2{#3}}{#1}{#2}%
}%
\long\def\XINT_applyunbr_loop_b #1#2{\XINT_applyunbr_loop_a {#2#1}}%
\long\def\XINT_applyunbr_end\xint_bye\expandafter\XINT_applyunbr_loop_b
    \expandafter #1#2#3{ #2}%
```

20.21. \xintApplyUnbraced:x (WIP, commented-out)

Added at 1.4 (2020/01/31) [on 2020/01/27]. For usage in the NumPy-like slicing routines.
The items should not contain \xint: and the applied macro should not contain \empty.
Finally, xintexpr.sty 1.4 code did not use this macro but the f-expandable one \xintApplyUnbraced.

Modified at 1.4b (2020/02/25). For 1.4b I prefer to keep the \xintApplyUnbraced:x code commented out, and classify it as WIP.

```
 \long\def\xintApplyUnbraced:x #1#2%
 {%
     \XINT_applyunbraced:x_loop {#1}#2%
     {\xint:\XINT_applyunbraced:x_loop_enda}{\xint:\XINT_applyunbraced:x_loop_endb}%
     {\xint:\XINT_applyunbraced:x_loop_endc}{\xint:\XINT_applyunbraced:x_loop_endd}%
     {\xint:\XINT_applyunbraced:x_loop_ende}{\xint:\XINT_applyunbraced:x_loop_endf}%
     {\xint:\XINT_applyunbraced:x_loop_endg}{\xint:\XINT_applyunbraced:x_loop_endh}\xint_bye
 }%
 \long\def\XINT_applyunbraced:x_loop #1#2#3#4#5#6#7#8#9%
 {%
     \xint_gob_til_xint: #9\xint:
             #1{#2}%
       \empty#1{#3}%
       \empty#1{#4}%
       \empty#1{#5}%
       \empty#1{#6}%
       \empty#1{#7}%
       \empty#1{#8}%
       \empty#1{#9}%
     \XINT_applyunbraced:x_loop {#1}%
 }%
 \long\def\XINT_applyunbraced:x_loop_endh\xint: #1\xint_bye{}%
 \long\def\XINT_applyunbraced:x_loop_endg\xint: #1\empty#2\xint_bye{#1}%
 \long\def\XINT_applyunbraced:x_loop_endf\xint: #1\empty
                                                #2\empty#3\xint_bye{#1#2}%
 \long\def\XINT_applyunbraced:x_loop_ende\xint: #1\empty
                                                #2\empty
                                                #3\empty#4\xint_bye{#1#2#3}%
```

```
\long\def\XINT_applyunbraced:x_loop_endd\xint: #1\empty
                                               #2\empty
                                               #3\empty
                                               #4\empty#5\xint_bye{#1#2#3#4}%
\long\def\XINT_applyunbraced:x_loop_endc\xint: #1\empty
                                               #2\empty
                                               #3\empty
                                               #4\empty
                                               #5\empty#6\xint_bye{#1#2#3#4#5}%
\long\def\XINT_applyunbraced:x_loop_endb\xint: #1\empty
                                               #2\empty
                                               #3\empty
                                               #4\empty
                                               #5\empty
                                               #6\empty#7\xint_bye{#1#2#3#4#5#6}%
\long\def\XINT_applyunbraced:x_loop_enda\xint: #1\empty
                                               #2\empty
                                               #3\empty
                                               #4\empty
                                               #5\empty
                                               #6\empty
                                               #7\empty#8\xint_bye{#1#2#3#4#5#6#7}%
```

20.22. \xintZip (WIP, not public)

Added at 1.4b (2020/02/25) [on 2020/02/25]. Support for zip(). Requires \expanded.

The implementation here thus considers the argument is already completely expanded and is a sequence of nut-ples. I will come back at later date for more generic macros.

Consider even the name of the function zip() as WIP.

As per what this does, it imitates the zip() function. See xint-manual.pdf.

I use lame terminators. Will think again later on this. I have to be careful with the used terminators, in particular with the NE context in mind.

Generally speaking I will think another day about efficiency else I will never start this.

OK, done. More compact than I initially thought. Various things should be commented upon here. Well, actually not so compact in the end as I basically had to double the whole thing simply to avoid the overhead of having to grab the final result delimited by some \xint_bye\xint_bye\xint_bye\xint_bye\empty terminator. Now actually rather \xint_bye\xint_bye\xint_bye\xint_bye\xint:

```
\def\xintZip #1{\expanded\XINT_zip_A#1\xint_bye\xint_bye}%
\def\XINT_zip_A#1%
{%
    \xint_bye#1{\expandafter}\xint_bye
    \expanded{\xint_noxpd{\XINT_ziptwo_A
        #1\xint_bye\xint_bye\xint_bye\xint_bye\xint:}\expandafter}%
    \expanded\XINT_zip_a
}%
\def\XINT_zip_a#1%
{%
    \xint_bye#1\XINT_zip_terminator\xint_bye
    \expanded{\xint_noxpd{\XINT_ziptwo_a
        #1\xint_bye\xint_bye\xint_bye\xint_bye\xint:}\expandafter}%
    \expanded\XINT_zip_a
}%
```

```
\def\XINT_zip_terminator\xint_bye#1\xint_bye{{}\empty\empty\empty\empty\xint:}%
\def\XINT_ziptwo_a #1#2#3#4#5\xint:#6#7#8#9%
{%
    \bgroup
    \xint_bye #1\XINT_ziptwo_e \xint_bye
    \xint_bye #6\XINT_ziptwo_e \xint_bye {{#1}#6}%
    \xint_bye #2\XINT_ziptwo_e \xint_bye
    \xint_bye #7\XINT_ziptwo_e \xint_bye {{#2}#7}%
    \xint_bye #3\XINT_ziptwo_e \xint_bye
    \xint_bye #8\XINT_ziptwo_e \xint_bye {{#3}#8}%
    \xint_bye #4\XINT_ziptwo_e \xint_bye
    \xint_bye #9\XINT_ziptwo_e \xint_bye {{#4}#9}%
```

Attention here that #6 can very well deliver no tokens at all. But the \ifx will then do the expected thing. Only mentioning!

By the way, the \xint_bye method means TeX needs to look into tokens but skipping braced groups. A conditional based method lets TeX look only at the start but then it has to find \else or \fi so here also it must looks at tokens, and actually goes into braced groups. But (written 2020/02/26) I never did serious testing comparing the two, and in xint I have usually preferred \xint_bye/\xint_gob_til_foo types of methods (they proved superior than \ifnum to check for 0000 in numerical core context for example, at the early days when xint used blocks of 4 digits, not 8), or usage of \if/\ifx only on single tokens, combined with some \xint_dothis/\xint_orthat syntax.

```
    \ifx \empty#6\expandafter\XINT_zipone_a\fi
    \XINT_ziptwo_b #5\xint:
}%
\def\XINT_zipone_a\XINT_ziptwo_b{\XINT_zipone_b}%
\def\XINT_ziptwo_b #1#2#3#4#5\xint:#6#7#8#9%
{%
    \xint_bye #1\XINT_ziptwo_e \xint_bye
    \xint_bye #6\XINT_ziptwo_e \xint_bye {{#1}#6}%
    \xint_bye #2\XINT_ziptwo_e \xint_bye
    \xint_bye #7\XINT_ziptwo_e \xint_bye {{#2}#7}%
    \xint_bye #3\XINT_ziptwo_e \xint_bye
    \xint_bye #8\XINT_ziptwo_e \xint_bye {{#3}#8}%
    \xint_bye #4\XINT_ziptwo_e \xint_bye
    \xint_bye #9\XINT_ziptwo_e \xint_bye {{#4}#9}%
    \XINT_ziptwo_b #5\xint:
}%
\def\XINT_ziptwo_e #1\XINT_ziptwo_b #2\xint:#3\xint:
    {\iffalse{\fi}\xint_bye\xint_bye\xint_bye\xint_bye\xint:}%
\def\XINT_zipone_b #1#2#3#4%
{%
    \xint_bye #1\XINT_zipone_e \xint_bye {{#1}}%
    \xint_bye #2\XINT_zipone_e \xint_bye {{#2}}%
    \xint_bye #3\XINT_zipone_e \xint_bye {{#3}}%
    \xint_bye #4\XINT_zipone_e \xint_bye {{#4}}%
    \XINT_zipone_b
}%
\def\XINT_zipone_e #1\XINT_zipone_b #2\xint:
    {\iffalse{\fi}\xint_bye\xint_bye\xint_bye\xint_bye\empty}%
\def\XINT_ziptwo_A #1#2#3#4#5\xint:#6#7#8#9%
{%
    \bgroup
```

```
    \xint_bye #1\XINT_ziptwo_end \xint_bye
    \xint_bye #6\XINT_ziptwo_end \xint_bye {{#1}#6}%
    \xint_bye #2\XINT_ziptwo_end \xint_bye
    \xint_bye #7\XINT_ziptwo_end \xint_bye {{#2}#7}%
    \xint_bye #3\XINT_ziptwo_end \xint_bye
    \xint_bye #8\XINT_ziptwo_end \xint_bye {{#3}#8}%
    \xint_bye #4\XINT_ziptwo_end \xint_bye
    \xint_bye #9\XINT_ziptwo_end \xint_bye {{#4}#9}%
    \ifx \empty#6\expandafter\XINT_zipone_A\fi
    \XINT_ziptwo_B #5\xint:
}%
\def\XINT_zipone_A\XINT_ziptwo_B{\XINT_zipone_B}%
\def\XINT_ziptwo_B #1#2#3#4#5\xint:#6#7#8#9%
{%
    \xint_bye #1\XINT_ziptwo_end \xint_bye
    \xint_bye #6\XINT_ziptwo_end \xint_bye {{#1}#6}%
    \xint_bye #2\XINT_ziptwo_end \xint_bye
    \xint_bye #7\XINT_ziptwo_end \xint_bye {{#2}#7}%
    \xint_bye #3\XINT_ziptwo_end \xint_bye
    \xint_bye #8\XINT_ziptwo_end \xint_bye {{#3}#8}%
    \xint_bye #4\XINT_ziptwo_end \xint_bye
    \xint_bye #9\XINT_ziptwo_end \xint_bye {{#4}#9}%
    \XINT_ziptwo_B #5\xint:
}%
\def\XINT_ziptwo_end #1\XINT_ziptwo_B #2\xint:#3\xint:{\iffalse{\fi}}%
\def\XINT_zipone_B #1#2#3#4%
{%
    \xint_bye #1\XINT_zipone_end \xint_bye {{#1}}%
    \xint_bye #2\XINT_zipone_end \xint_bye {{#2}}%
    \xint_bye #3\XINT_zipone_end \xint_bye {{#3}}%
    \xint_bye #4\XINT_zipone_end \xint_bye {{#4}}%
    \XINT_zipone_B
}%
\def\XINT_zipone_end #1\XINT_zipone_B #2\xint:#3\xint:{\iffalse{\fi}}%
```

20.23. \xintSeq

Added at 1.09c (2013/10/09). Without the optional argument puts stress on the input stack, should not be used to generated thousands of terms then.

Modified at 1.4j (2021/07/13). This venerable macro had a brace removal bug in case it produced a single number: `\xintSeq{10}{10}` expanded to `10` not `{10}`. When I looked at the code the bug looked almost deliberate to me, but reading the documentation (which I have not modified), the behaviour is really unexpected. And the variant with step parameter `\xintSeq[1]{10}{10}` did produce `{10}`, so yes, definitely it was a bug!

I take this occasion to do some style (and perhaps efficiency) refactoring in the coding. I feel there is room for improvement, no time this time. And I don't touch the variant with step parameter.

Memo: xintexpr has some variants, a priori on ultra quick look they do not look like having similar bug as this one had.

```
\def\xintSeq {\romannumeral0\xintseq }%
\def\xintseq #1{\XINT_seq_chkopt  #1\xint_bye }%
\def\XINT_seq_chkopt #1%
```

```
{%
    \ifx [#1\expandafter\XINT_seq_opt
       \else\expandafter\XINT_seq_noopt
    \fi  #1%
}%
\def\XINT_seq_noopt #1\xint_bye #2%
{%
    \expandafter\XINT_seq
    \the\numexpr#1\expandafter.\the\numexpr #2.%
}%
\def\XINT_seq #1.#2.%
{%
   \ifnum #1=#2 \xint_dothis\XINT_seq_e\fi
   \ifnum #2>#1 \xint_dothis\XINT_seq_pa\fi
                \xint_orthat\XINT_seq_na
   #2.{#1}{#2}%
}%
\def\XINT_seq_e#1.#2{}%
\def\XINT_seq_pa {\expandafter\XINT_seq_p\the\numexpr-\xint_c_i+}%
\def\XINT_seq_na {\expandafter\XINT_seq_n\the\numexpr\xint_c_i+}%
\def\XINT_seq_p #1.#2%
{%
    \ifnum #1>#2
      \expandafter\XINT_seq_p\the
    \else
      \expandafter\XINT_seq_e
    \fi
    \numexpr #1-\xint_c_i.{#2}{#1}%
}%
\def\XINT_seq_n #1.#2%
{%
    \ifnum #1<#2
      \expandafter\XINT_seq_n\the
    \else
      \expandafter\XINT_seq_e
    \fi
    \numexpr #1+\xint_c_i.{#2}{#1}%
}%
```

Note at time of the 1.4j bug fix : I definitely should improve this branch and diminish the number of expandafter's but no time this time.

```
\def\XINT_seq_opt [\xint_bye #1]#2#3%
{%
    \expandafter\XINT_seqo\expandafter
    {\the\numexpr #2\expandafter}\expandafter
    {\the\numexpr #3\expandafter}\expandafter
    {\the\numexpr #1}%
}%
\def\XINT_seqo #1#2%
{%
   \ifcase\ifnum #1=#2 0\else\ifnum #2>#1 1\else -1\fi\fi\space
      \expandafter\XINT_seqo_a
   \or
```

```
716       \expandafter\XINT_seqo_pa
717    \else
718       \expandafter\XINT_seqo_na
719    \fi
720    {#1}{#2}%
721 }%
722 \def\XINT_seqo_a #1#2#3{ {#1}}%
723 \def\XINT_seqo_o #1#2#3#4{ #4}%
724 \def\XINT_seqo_pa #1#2#3%
725 {%
726     \ifcase\ifnum #3=\xint_c_ 0\else\ifnum #3>\xint_c_ 1\else -1\fi\fi\space
727            \expandafter\XINT_seqo_o
728     \or
729            \expandafter\XINT_seqo_pb
730     \else
731            \xint_afterfi{\expandafter\space\xint_gobble_iv}%
732     \fi
733     {#1}{#2}{#3}{{#1}}%
734 }%
735 \def\XINT_seqo_pb #1#2#3%
736 {%
737     \expandafter\XINT_seqo_pc\expandafter{\the\numexpr #1+#3}{#2}{#3}%
738 }%
739 \def\XINT_seqo_pc #1#2%
740 {%
741     \ifnum #1>#2
742         \expandafter\XINT_seqo_o
743     \else
744         \expandafter\XINT_seqo_pd
745     \fi
746     {#1}{#2}%
747 }%
748 \def\XINT_seqo_pd #1#2#3#4{\XINT_seqo_pb {#1}{#2}{#3}{#4{#1}}}%
749 \def\XINT_seqo_na #1#2#3%
750 {%
751     \ifcase\ifnum #3=\xint_c_ 0\else\ifnum #3>\xint_c_ 1\else -1\fi\fi\space
752         \expandafter\XINT_seqo_o
753     \or
754         \xint_afterfi{\expandafter\space\xint_gobble_iv}%
755     \else
756         \expandafter\XINT_seqo_nb
757     \fi
758     {#1}{#2}{#3}{{#1}}%
759 }%
760 \def\XINT_seqo_nb #1#2#3%
761 {%
762     \expandafter\XINT_seqo_nc\expandafter{\the\numexpr #1+#3}{#2}{#3}%
763 }%
764 \def\XINT_seqo_nc #1#2%
765 {%
766     \ifnum #1<#2
767         \expandafter\XINT_seqo_o
```

```
    \else
        \expandafter\XINT_seqo_nd
    \fi
    {#1}{#2}%
}%
\def\XINT_seqo_nd #1#2#3#4{\XINT_seqo_nb {#1}{#2}{#3}{#4{#1}}}%
```

20.24. \xintloop, \xintbreakloop, \xintbreakloopanddo, \xintloopskiptonext

Added at 1.09g (2013/11/22) [on 2013/11/22].

Modified at 1.09h (2013/11/28). Made \long.

```
\long\def\xintloop #1#2\repeat {#1#2\xintloop_again\fi\xint_gobble_i {#1#2}}%
\long\def\xintloop_again\fi\xint_gobble_i #1{\fi
                              #1\xintloop_again\fi\xint_gobble_i {#1}}%
\long\def\xintbreakloop #1\xintloop_again\fi\xint_gobble_i #2{}%
\long\def\xintbreakloopanddo #1#2\xintloop_again\fi\xint_gobble_i #3{#1}%
\long\def\xintloopskiptonext #1\xintloop_again\fi\xint_gobble_i #2{%
                         #2\xintloop_again\fi\xint_gobble_i {#2}}%
```

20.25. \xintiloop, \xintiloopindex, \xintbracediloopindex, \xintouteriloopindex, \xintbracedouteriloopindex, \xintbreakiloop, \xintbreakiloopanddo, \xintiloopskiptonext, \xintiloopskipandredo

Added at 1.09g (2013/11/22) [on 2013/11/22].

Modified at 1.09h (2013/11/28). Made \long.

Added at 1.3b (2018/05/18) [on 2018/04/24]. "braced" variants.

```
\def\xintiloop [#1+#2]{%
    \expandafter\xintiloop_a\the\numexpr #1\expandafter.\the\numexpr #2.}%
\long\def\xintiloop_a #1.#2.#3#4\repeat{%
    #3#4\xintiloop_again\fi\xint_gobble_iii {#1}{#2}{#3#4}}%
\def\xintiloop_again\fi\xint_gobble_iii #1#2{%
    \fi\expandafter\xintiloop_again_b\the\numexpr#1+#2.#2.}%
\long\def\xintiloop_again_b #1.#2.#3{%
    #3\xintiloop_again\fi\xint_gobble_iii {#1}{#2}{#3}}%
\long\def\xintbreakiloop #1\xintiloop_again\fi\xint_gobble_iii #2#3#4{}%
\long\def\xintbreakiloopanddo
     #1.#2\xintiloop_again\fi\xint_gobble_iii #3#4#5{#1}%
\long\def\xintiloopindex #1\xintiloop_again\fi\xint_gobble_iii #2%
                {#2#1\xintiloop_again\fi\xint_gobble_iii {#2}}%
\long\def\xintbracediloopindex #1\xintiloop_again\fi\xint_gobble_iii #2%
     {{#2}#1\xintiloop_again\fi\xint_gobble_iii {#2}}%
\long\def\xintouteriloopindex #1\xintiloop_again
                         #2\xintiloop_again\fi\xint_gobble_iii #3%
   {#3#1\xintiloop_again #2\xintiloop_again\fi\xint_gobble_iii {#3}}%
\long\def\xintbracedouteriloopindex #1\xintiloop_again
                         #2\xintiloop_again\fi\xint_gobble_iii #3%
   {{#3}#1\xintiloop_again #2\xintiloop_again\fi\xint_gobble_iii {#3}}%
\long\def\xintiloopskiptonext #1\xintiloop_again\fi\xint_gobble_iii #2#3{%
    \expandafter\xintiloop_again_b \the\numexpr#2+#3.#3.}%
\long\def\xintiloopskipandredo #1\xintiloop_again\fi\xint_gobble_iii #2#3#4{%
    #4\xintiloop_again\fi\xint_gobble_iii {#2}{#3}{#4}}%
```

20.26. \XINT_xflet

Added at 1.09e (2013/10/29) [on 2013/10/29]. We f-expand unbraced tokens and swallow arising space tokens until the dust settles.

```
\def\XINT_xflet #1%
{%
    \def\XINT_xflet_macro {#1}\XINT_xflet_zapsp
}%
\def\XINT_xflet_zapsp
{%
    \expandafter\futurelet\expandafter\XINT_token
    \expandafter\XINT_xflet_sp?\romannumeral`&&@%
}%
\def\XINT_xflet_sp?
{%
    \ifx\XINT_token\XINT_sptoken
         \expandafter\XINT_xflet_zapsp
    \else\expandafter\XINT_xflet_zapspB
    \fi
}%
\def\XINT_xflet_zapspB
{%
    \expandafter\futurelet\expandafter\XINT_tokenB
    \expandafter\XINT_xflet_spB?\romannumeral`&&@%
}%
\def\XINT_xflet_spB?
{%
    \ifx\XINT_tokenB\XINT_sptoken
         \expandafter\XINT_xflet_zapspB
    \else\expandafter\XINT_xflet_eq?
    \fi
}%
\def\XINT_xflet_eq?
{%
    \ifx\XINT_token\XINT_tokenB
         \expandafter\XINT_xflet_macro
    \else\expandafter\XINT_xflet_zapsp
    \fi
}%
```

20.27. \xintApplyInline

Added at 1.09a (2013/09/24). \xintApplyInline\macro{{a}{b}...{z}} has the same effect as executing \macro{a} and then applying again \xintApplyInline to the shortened list {{b}...{z}} until nothing is left. This is a non-expandable command which will result in quicker code than using \xintApplyUnbraced. It f-expands its second (list) argument first, which may thus be encapsulated in a macro.

Modified at 1.09c (2013/10/09). Rewritten. Nota bene: uses catcode 3 Z as privated list terminator.

```
\catcode`Z 3
\long\def\xintApplyInline #1#2%
{%
```

```
  \long\expandafter\def\expandafter\XINT_inline_macro
  \expandafter ##\expandafter 1\expandafter {#1{##1}}%
  \XINT_xflet\XINT_inline_b #2Z% this Z has catcode 3
}%
\def\XINT_inline_b
{%
    \ifx\XINT_token Z\expandafter\xint_gobble_i
    \else\expandafter\XINT_inline_d\fi
}%
\long\def\XINT_inline_d #1%
{%
  \long\def\XINT_item{{#1}}\XINT_xflet\XINT_inline_e
}%
\def\XINT_inline_e
{%
    \ifx\XINT_token Z\expandafter\XINT_inline_w
    \else\expandafter\XINT_inline_f\fi
}%
\def\XINT_inline_f
{%
  \expandafter\XINT_inline_g\expandafter{\XINT_inline_macro {##1}}%
}%
\long\def\XINT_inline_g #1%
{%
   \expandafter\XINT_inline_macro\XINT_item
   \long\def\XINT_inline_macro ##1{#1}\XINT_inline_d
}%
\def\XINT_inline_w #1%
{%
   \expandafter\XINT_inline_macro\XINT_item
}%
```

20.28. \xintFor, \xintFor*, \xintBreakFor, \xintBreakForAndDo

Added at 1.09c (2013/10/09) [on 2013/10/09]. A new kind of loop which uses macro parameters #1, #2, #3, #4 rather than macros; while not expandable it survives executing code closing groups, like what happens in an alignment with the & character. When inserted in a macro for later use, the # character must be doubled.

The non-star variant works on a csv list, which it expands once, the star variant works on a token list, which it (repeatedly) f-expands.

Modified at 1.09e (2013/10/29). Adds \XINT_forever with \xintintegers, \xintdimensions, \xintrationals and \xintBreakFor, \xintBreakForAndDo, \xintifForFirst, \xintifForLast. On this occasion \xi
↪ nt_firstoftwo and \xint_secondoftwo are made long.

Modified at 1.09f (2013/11/04). Rewrites large parts of \xintFor code in order to filter the comma separated list via \xintCSVtoList which gets rid of spaces. The #1 in \XINT_for_forever? has an initial space token which serves two purposes: preventing brace stripping, and stopping the expansion made by \xintcsvtolist. If the \XINT_forever branch is taken, the added space will not be a problem there.

Now allows all macro parameters from #1 to #9 in \xintFor, \xintFor*, and \XINT_forever.

Modified at 1.2i (2016/12/13). Slightly more robust \xintifForFirst/Last in case of nesting.

```
\def\XINT_tmpa #1#2{\ifnum #2<#1 \xint_afterfi {{#########2}}\fi}%
\def\XINT_tmpb #1#2{\ifnum #1<#2 \xint_afterfi {{#########2}}\fi}%
```

```
\def\XINT_tmpc #1%
{%
    \expandafter\edef \csname XINT_for_left#1\endcsname
               {\xintApplyUnbraced {\XINT_tmpa #1}{123456789}}%
    \expandafter\edef \csname XINT_for_right#1\endcsname
               {\xintApplyUnbraced {\XINT_tmpb #1}{123456789}}%
}%
\xintApplyInline \XINT_tmpc {123456789}%
\long\def\xintBreakFor      #1Z{}%
\long\def\xintBreakForAndDo #1#2Z{#1}%
\def\xintFor {\let\xintifForFirst\xint_firstoftwo
              \let\xintifForLast\xint_secondoftwo
              \futurelet\XINT_token\XINT_for_ifstar }%
\def\XINT_for_ifstar {\ifx\XINT_token*\expandafter\XINT_forx
                                \else\expandafter\XINT_for \fi }%
\catcode`U 3 % with numexpr
\catcode`V 3 % with xintfrac.sty (xint.sty not enough)
\catcode`D 3 % with dimexpr
\def\XINT_flet_zapsp
{%
    \futurelet\XINT_token\XINT_flet_sp?
}%
\def\XINT_flet_sp?
{%
    \ifx\XINT_token\XINT_sptoken
         \xint_afterfi{\expandafter\XINT_flet_zapsp\romannumeral0}%
    \else\expandafter\XINT_flet_macro
    \fi
}%
\long\def\XINT_for #1#2in#3#4#5%
{%
    \expandafter\XINT_toks\expandafter
        {\expandafter\XINT_for_d\the\numexpr #2\relax {#5}}%
    \def\XINT_flet_macro {\expandafter\XINT_for_forever?\space}%
    \expandafter\XINT_flet_zapsp #3Z%
}%
\def\XINT_for_forever? #1Z%
{%
    \ifx\XINT_token U\XINT_to_forever\fi
    \ifx\XINT_token V\XINT_to_forever\fi
    \ifx\XINT_token D\XINT_to_forever\fi
    \expandafter\the\expandafter\XINT_toks\romannumeral0\xintcsvtolist {#1}Z%
}%
\def\XINT_to_forever\fi #1\xintcsvtolist #2{\fi \XINT_forever #2}%
\long\def\XINT_forx *#1#2in#3#4#5%
{%
    \expandafter\XINT_toks\expandafter
       {\expandafter\XINT_forx_d\the\numexpr #2\relax {#5}}%
    \XINT_xflet\XINT_forx_forever? #3Z%
}%
\def\XINT_forx_forever?
{%
```

```
    \ifx\XINT_token U\XINT_to_forxever\fi
    \ifx\XINT_token V\XINT_to_forxever\fi
    \ifx\XINT_token D\XINT_to_forxever\fi
    \XINT_forx_empty?
}%
\def\XINT_to_forxever\fi #1\XINT_forx_empty? {\fi \XINT_forever }%
\catcode`U 11
\catcode`D 11
\catcode`V 11
\def\XINT_forx_empty?
{%
    \ifx\XINT_token Z\expandafter\xintBreakFor\fi
    \the\XINT_toks
}%
\long\def\XINT_for_d #1#2#3%
{%
  \long\def\XINT_y ##1##2##3##4##5##6##7##8##9{#2}%
  \XINT_toks {{#3}}%
  \long\edef\XINT_x {\noexpand\XINT_y \csname XINT_for_left#1\endcsname
                     \the\XINT_toks   \csname XINT_for_right#1\endcsname }%
  \XINT_toks {\XINT_x\let\xintifForFirst\xint_secondoftwo
                     \let\xintifForLast\xint_secondoftwo\XINT_for_d #1{#2}}%
  \futurelet\XINT_token\XINT_for_last?
}%
\long\def\XINT_forx_d #1#2#3%
{%
  \long\def\XINT_y ##1##2##3##4##5##6##7##8##9{#2}%
  \XINT_toks {{#3}}%
  \long\edef\XINT_x {\noexpand\XINT_y \csname XINT_for_left#1\endcsname
                     \the\XINT_toks   \csname XINT_for_right#1\endcsname }%
  \XINT_toks {\XINT_x\let\xintifForFirst\xint_secondoftwo
                     \let\xintifForLast\xint_secondoftwo\XINT_forx_d #1{#2}}%
  \XINT_xflet\XINT_for_last?
}%
\def\XINT_for_last?
{%
  \ifx\XINT_token Z\expandafter\XINT_for_last?yes\fi
  \the\XINT_toks
}%
\def\XINT_for_last?yes
{%
 \let\xintifForLast\xint_firstoftwo
 \xintBreakForAndDo{\XINT_x\xint_gobble_i Z}%
}%
```

20.29. \XINT_forever, \xintintegers, \xintdimensions, \xintrationals

Added at 1.09e (2013/10/29).

Modified at 1.09f (2013/11/04). The \xintrationals feature uses \xintiiadd/\xintiimul to avoid the overhead of the full-blown frac version \xintAdd and \xintMul. Also 1.09f uses \xintZapSpacesB in this \xintrationals case to get rid of leading and ending spaces in the #4 and #5 delimited parameters of \XINT_forever_opt_a. For \xintintegers and \xintdimensions this is

not necessary, due to the use of \numexpr resp. \dimexpr in \XINT_?expr_Ua, resp.\XINT_?expr_Da.

```
\catcode`U 3
\catcode`D 3
\catcode`V 3
\let\xintegers       U%
\let\xintintegers    U%
\let\xintdimensions  D%
\let\xintrationals   V%
\def\XINT_forever #1%
{%
  \expandafter\XINT_forever_a
  \csname XINT_?expr_\ifx#1UU\else\ifx#1DD\else V\fi\fi a\expandafter\endcsname
  \csname XINT_?expr_\ifx#1UU\else\ifx#1DD\else V\fi\fi i\expandafter\endcsname
  \csname XINT_?expr_\ifx#1UU\else\ifx#1DD\else V\fi\fi \endcsname
}%
\catcode`U 11
\catcode`D 11
\catcode`V 11
\def\XINT_?expr_Ua #1#2%
   {\expandafter{\expandafter\numexpr\the\numexpr #1\expandafter\relax
                               \expandafter\relax\expandafter}%
    \expandafter{\the\numexpr #2}}%
\def\XINT_?expr_Da #1#2%
   {\expandafter{\expandafter\dimexpr\number\dimexpr #1\expandafter\relax
                  \expandafter s\expandafter p\expandafter\relax\expandafter}%
    \expandafter{\number\dimexpr #2}}%
\catcode`Z 11
\def\XINT_?expr_Va #1#2%
{%
    \expandafter\XINT_?expr_Vb\expandafter
          {\romannumeral`&&@\xintrawwithzeros{\xintZapSpacesB{#2}}}%
          {\romannumeral`&&@\xintrawwithzeros{\xintZapSpacesB{#1}}}%
}%
\catcode`Z 3
\def\XINT_?expr_Vb #1#2{\expandafter\XINT_?expr_Vc #2.#1.}%
\def\XINT_?expr_Vc #1/#2.#3/#4.%
{%
     \xintifEq {#2}{#4}%
       {\XINT_?expr_Vf {#3}{#1}{#2}}%
       {\expandafter\XINT_?expr_Vd\expandafter
        {\romannumeral0\xintiimul {#2}{#4}}%
        {\romannumeral0\xintiimul {#1}{#4}}%
        {\romannumeral0\xintiimul {#2}{#3}}%
       }%
}%
\def\XINT_?expr_Vd #1#2#3{\expandafter\XINT_?expr_Ve\expandafter {#2}{#3}{#1}}%
\def\XINT_?expr_Ve #1#2{\expandafter\XINT_?expr_Vf\expandafter {#2}{#1}}%
\def\XINT_?expr_Vf #1#2#3{{#2/#3}{{0}{#1}{#2}{#3}}}%
\def\XINT_?expr_Ui {{\numexpr 1\relax}{1}}%
\def\XINT_?expr_Di {{\dimexpr 0pt\relax}{65536}}%
\def\XINT_?expr_Vi {{1/1}{0111}}%
```

```
\def\XINT_?expr_U #1#2%
   {\expandafter{\expandafter\numexpr\the\numexpr #1+#2\relax\relax}{#2}}%
\def\XINT_?expr_D #1#2%
   {\expandafter{\expandafter\dimexpr\the\numexpr #1+#2\relax sp\relax}{#2}}%
\def\XINT_?expr_V #1#2{\XINT_?expr_Vx #2}%
\def\XINT_?expr_Vx #1#2%
{%
     \expandafter\XINT_?expr_Vy\expandafter
        {\romannumeral0\xintiiadd {#1}{#2}}{#2}%
}%
\def\XINT_?expr_Vy #1#2#3#4%
{%
     \expandafter{\romannumeral0\xintiiadd {#3}{#1}/#4}{{#1}{#2}{#3}{#4}}%
}%
\def\XINT_forever_a #1#2#3#4%
{%
    \ifx #4[\expandafter\XINT_forever_opt_a
       \else\expandafter\XINT_forever_b
    \fi #1#2#3#4%
}%
\def\XINT_forever_b #1#2#3Z{\expandafter\XINT_forever_c\the\XINT_toks #2#3}%
\long\def\XINT_forever_c #1#2#3#4#5%
    {\expandafter\XINT_forever_d\expandafter #2#4#5{#3}Z}%
\def\XINT_forever_opt_a #1#2#3[#4+#5]#6Z%
{%
    \expandafter\expandafter\expandafter
    \XINT_forever_opt_c\expandafter\the\expandafter\XINT_toks
    \romannumeral`&&@#1{#4}{#5}#3%
}%
\long\def\XINT_forever_opt_c #1#2#3#4#5#6{\XINT_forever_d #2{#4}{#5}#6{#3}Z}%
\long\def\XINT_forever_d #1#2#3#4#5%
{%
  \long\def\XINT_y ##1##2##3##4##5##6##7##8##9{#5}%
  \XINT_toks {{#2}}%
  \long\edef\XINT_x {\noexpand\XINT_y \csname XINT_for_left#1\endcsname
                     \the\XINT_toks   \csname XINT_for_right#1\endcsname }%
  \XINT_x
  \let\xintifForFirst\xint_secondoftwo
  \let\xintifForLast\xint_secondoftwo
  \expandafter\XINT_forever_d\expandafter #1\romannumeral`&&@#4{#2}{#3}#4{#5}%
}%
```

20.30. \xintForpair, \xintForthree, \xintForfour

Added at 1.09c (2013/10/09).

Modified at 1.09f (2013/11/04). \xintForpair delegate to \xintCSVtoList and its \xintZapSpacesB the handling of spaces. Does not share code with \xintFor anymore.

\xintForpair extended to accept #1#2, #2#3 etc... up to #8#9, \xintForthree, #1#2#3 up to #7#8#9, \xintForfour id.

Modified at 1.2i (2016/12/13). Slightly more robust \xintifForFirst/\xintifForLast in case of nesting.

Modified at 1.4n (2025/09/05). Allow one final extraneous comma (formerly, such input caused a

crash). This goes via testing an extra token #6, which should be an opening parenthesis. The potential brace removal changes behavior for illegal inputs. Successive commas in the input list are still not allowed.

```
\catcode`j 3
\long\def\xintForpair #1#2#3in#4#5#6%
{%
    \let\xintifForFirst\xint_firstoftwo
    \let\xintifForLast\xint_secondoftwo
    \XINT_toks  {\XINT_forpair_d #2{#6}}%
    \expandafter\the\expandafter\XINT_toks #4jZ%
}%
\long\def\XINT_forpair_d #1#2#3(#4)#5#6%
{%
  \long\def\XINT_y ##1##2##3##4##5##6##7##8##9{#2}%
  \XINT_toks \expandafter{\romannumeral0\xintcsvtolist{ #4}}%
  \long\edef\XINT_x {\noexpand\XINT_y \csname XINT_for_left#1\endcsname
      \the\XINT_toks \csname XINT_for_right\the\numexpr#1+\xint_c_i\endcsname}%
  \if1\ifx #5j1\else\ifx#6j1\else0\fi\fi\expandafter\XINT_for_last?yes\fi
  \XINT_x
  \let\xintifForFirst\xint_secondoftwo
  \let\xintifForLast\xint_secondoftwo
  \XINT_forpair_d #1{#2}#6%
}%
\long\def\xintForthree #1#2#3in#4#5#6%
{%
    \let\xintifForFirst\xint_firstoftwo
    \let\xintifForLast\xint_secondoftwo
    \XINT_toks  {\XINT_forthree_d #2{#6}}%
    \expandafter\the\expandafter\XINT_toks #4jZ%
}%
\long\def\XINT_forthree_d #1#2#3(#4)#5#6%
{%
  \long\def\XINT_y ##1##2##3##4##5##6##7##8##9{#2}%
  \XINT_toks \expandafter{\romannumeral0\xintcsvtolist{ #4}}%
  \long\edef\XINT_x {\noexpand\XINT_y \csname XINT_for_left#1\endcsname
      \the\XINT_toks \csname XINT_for_right\the\numexpr#1+\xint_c_ii\endcsname}%
  \if1\ifx #5j1\else\ifx#6j1\else0\fi\fi\expandafter\XINT_for_last?yes\fi
  \XINT_x
  \let\xintifForFirst\xint_secondoftwo
  \let\xintifForLast\xint_secondoftwo
  \XINT_forthree_d #1{#2}#6%
}%
\long\def\xintForfour #1#2#3in#4#5#6%
{%
    \let\xintifForFirst\xint_firstoftwo
    \let\xintifForLast\xint_secondoftwo
    \XINT_toks  {\XINT_forfour_d #2{#6}}%
    \expandafter\the\expandafter\XINT_toks #4jZ%
}%
\long\def\XINT_forfour_d #1#2#3(#4)#5#6%
{%
  \long\def\XINT_y ##1##2##3##4##5##6##7##8##9{#2}%
```

```
  \XINT_toks \expandafter{\romannumeral0\xintcsvtolist{ #4}}%
  \long\edef\XINT_x {\noexpand\XINT_y \csname XINT_for_left#1\endcsname
     \the\XINT_toks \csname XINT_for_right\the\numexpr#1+\xint_c_iii\endcsname}%
  \if1\ifx #5j1\else\ifx#6j1\else0\fi\fi\expandafter\XINT_for_last?yes\fi
  \XINT_x
  \let\xintifForFirst\xint_secondoftwo
  \let\xintifForLast\xint_secondoftwo
  \XINT_forfour_d #1{#2}#6%
}%
\catcode`Z 11
\catcode`j 11
```

20.31. \xintAssign

\xintAssign {a}{b}..{z}\to\A\B...\Z

Modified at 1.1c (2015/09/12). Belatedly corrects some "features" of \xintAssign which didn't like the case of a space right before the "\to", or the case with the first token not an opening brace and the subsequent material containing brace groups. The new code handles gracefully these situations.

```
\def\xintAssign{\def\XINT_flet_macro {\XINT_assign_fork}\XINT_flet_zapsp }%
\def\XINT_assign_fork
{%
    \let\XINT_assign_def\def
    \ifx\XINT_token[\expandafter\XINT_assign_opt
                \else\expandafter\XINT_assign_a
    \fi
}%
\def\XINT_assign_opt [#1]%
{%
    \ifcsname #1def\endcsname
      \expandafter\let\expandafter\XINT_assign_def \csname #1def\endcsname
    \else
      \expandafter\let\expandafter\XINT_assign_def \csname xint#1def\endcsname
    \fi
    \XINT_assign_a
}%
\long\def\XINT_assign_a #1\to
{%
    \def\XINT_flet_macro{\XINT_assign_b}%
    \expandafter\XINT_flet_zapsp\romannumeral`&&@#1\xint:\to
}%
\long\def\XINT_assign_b
{%
    \ifx\XINT_token\bgroup
         \expandafter\XINT_assign_c
    \else\expandafter\XINT_assign_f
    \fi
}%
\long\def\XINT_assign_f #1\xint:\to #2%
{%
    \XINT_assign_def #2{#1}%
}%
```

```
\long\def\XINT_assign_c #1%
{%
    \def\XINT_assign_tmp {#1}%
    \ifx\XINT_assign_tmp\xint_bracedstopper
        \expandafter\XINT_assign_e
    \else
        \expandafter\XINT_assign_d
    \fi
}%
\long\def\XINT_assign_d #1\to #2%
{%
    \expandafter\XINT_assign_def\expandafter #2\expandafter{\XINT_assign_tmp}%
    \XINT_assign_c #1\to
}%
\def\XINT_assign_e #1\to {}%
\def\xintRelaxArray #1%
{%
    \edef\XINT_restoreescapechar {\escapechar\the\escapechar\relax}%
    \escapechar -1
    \expandafter\def\expandafter\xint_arrayname\expandafter {\string #1}%
    \XINT_restoreescapechar
    \xintiloop [\csname\xint_arrayname 0\endcsname+-1]
      \global
      \expandafter\let\csname\xint_arrayname\xintiloopindex\endcsname\relax
      \ifnum \xintiloopindex > \xint_c_
    \repeat
    \global\expandafter\let\csname\xint_arrayname 00\endcsname\relax
    \global\let #1\relax
}%
```

20.32. \xintAssignArray, \xintDigitsOf

```
\xintAssignArray {a}{b}..{z}\to\U.
  \xintDigitsOf=\xintAssignArray.
```

Modified at 1.4p (2026/03/01). Allow usage of the array macro with a negative index.

```
\def\xintAssignArray{\def\XINT_flet_macro {\XINT_assignarray_fork}%
                     \XINT_flet_zapsp }%
\def\XINT_assignarray_fork
{%
    \let\XINT_assignarray_def\def
    \ifx\XINT_token[\expandafter\XINT_assignarray_opt
                \else\expandafter\XINT_assignarray
    \fi
}%
\def\XINT_assignarray_opt [#1]%
{%
    \ifcsname #1def\endcsname
      \expandafter\let\expandafter\XINT_assignarray_def \csname #1def\endcsname
    \else
      \expandafter\let\expandafter\XINT_assignarray_def
                                  \csname xint#1def\endcsname
    \fi
```

```
    \XINT_assignarray
}%
```

Very old code got refactored at 1.4p. Hesitation about using \XINT_tmpa or like here potential ganer is used by caller.

```
\long\def\XINT_assignarray #1\to #2%
{%
    \edef\XINT_restoreescapechar {\escapechar\the\escapechar\relax }%
    \escapechar -1
    \expandafter\def\expandafter\XINT_arrayname\expandafter {\string #2}%
    \XINT_restoreescapechar
    \def\XINT_itemcount {0}%
    \expandafter\XINT_assignarray_loop \romannumeral`&&@#1\xint:#2%
}%
\long\def\XINT_assignarray_loop #1%
{%
    \def\XINT_assign_tmp {#1}%
    \ifx\XINT_assign_tmp\xint_bracedstopper
       \expandafter\let\csname\XINT_arrayname0\endcsname\XINT_itemcount
       \expandafter\XINT_assignarray_end
    \else
       \edef\XINT_itemcount{\the\numexpr\XINT_itemcount+\xint_c_i}%
       \expandafter\XINT_assignarray_def
       \csname\XINT_arrayname\XINT_itemcount\expandafter\endcsname
                                            \expandafter{\XINT_assign_tmp }%
       \expandafter\XINT_assignarray_loop
    \fi
}%
```

Modified at 1.4p (2026/03/01). Allow negative index, handled as in Python (positive index retains former meaning, i.e.starts at 1).

This very old code from 2013 got refactored on this occasion. Code style hackier. A bit annoying though that \expandafter\expandafter\expandafter, because of \csname. Would have been easier to incorporate the terminator in the \foo<N> macro itself but I feared this would be too unexpected in case of needs of some direct hacking.

On this occasion the number of items, which is used by \foo to test if the index is out of range is now present via its explicit tokens rather than via encapsulation \foo0.

#4 is \foo, #3 is foo, #2 is the number of items, and #1 is \foo00. The Nth item is in \foo<N>.

```
\def\XINT_assignarray_end
{%
    \expandafter\XINT_assignarray_endend
    \csname\XINT_arrayname00\expandafter\endcsname
    \expanded{{\XINT_itemcount}{\XINT_arrayname}}%
}%
\def\XINT_assignarray_endend #1#2#3#4%
{%
    \def #4##1%
    {%
        \romannumeral\expandafter\expandafter\expandafter
        \xint_c_\csname\expandafter#1\the\numexpr ##1.%
    }%
    \def #1##1##2.%
    {%
```

```
        \ifnum\if-##1\else##1\fi##2>#2
            \expanded{\noexpand\XINT_expandableerror
        {Illegal index ##1##2 (\if-##1< -\else> \fi #2) for \string#4}}%
            empty%
        \else
            #3\if-##1\the\numexpr\xint_c_i+#2\fi
            ##1##2%
        \fi
        \endcsname
    }%
}%
\let\xintDigitsOf\xintAssignArray
```

20.33. CSV (non user documented) variants of Length, Keep, Trim, NthElt, Reverse

Modified at 1.2j (2016/12/22). These routines are for use by \xintListSel:x:csv and \xintListSel:f:csv from xintexpr, and also for the reversed and len functions. Refactored for 1.2j release, following 1.2i updates to \xintKeep, \xintTrim, ...

These macros will remain undocumented in the user manual:

-- they exist primarily for internal use by the xintexpr parsers, hence don't have to be general purpose; for example, they a priori need to handle only catcode 12 tokens (not true in \xintNewExpr, though) hence they are not really worried about controlling brace stripping (nevertheless 1.2j has paid some secondary attention to it, see below.) They are not worried about normalizing leading spaces either, because none will be encountered when the macros are used as auxiliaries to the expression parsers.

-- crucial design elements may change in future:

1. whether the handled lists must have or not have a final comma. Currently, the model is the one of comma separated lists with **no** final comma. But this means that there can not be a distinction of principle between a truly empty list and a list which contains one item which turns out to be empty. More importantly it makes the coding more complicated as it is needed to distinguish the empty list from the single-item list, both lacking commas.

   For the internal use of xintexpr, it would be ok to require all list items to be terminated by a comma, and this would bring quite some simplications here, but as initially I started with non-terminated lists, I have left it this way in the 1.2j refactoring.

2. the way to represent the empty list. I was tempted for matter of optimization and synchronization with xintexpr context to require the empty list to be always represented by a space token and to not let the macros admit a completely empty input. But there were complications so for the time being 1.2j does accept truly empty output (it is not distinguished from an input equal to a space token) and produces empty output for empty list. This means that the status of the «nil» object for the xintexpr parsers is not completely clarified (currently it is represented by a space token).

The original Python slicing code in xintexpr 1.1 used \xintCSVtoList and \xintListWithSep{,} to convert back and forth to token lists and apply \xintKeep/\xintTrim. Release 1.2g switched to devoted f-expandable macros added to xinttools. Release 1.2j refactored all these macros as a follow-up to 1.2i improvements to \xintKeep/\xintTrim. They were made \long on this occasion and auxiliary \xintLengthUpTo:f:csv was added.

Leading spaces in items are currently maintained as is by the 1.2j macros, even by \xintNthEltPy:f:csv, with the exception of the first item, as the list is f-expanded. Perhaps \xintNthEltPy:f:csv should remove a leading space if present in the picked item; anyway, there are no spaces for the lists handled internally by the Python slicer of xintexpr, except the «nil» object currently represented by exactly one space.

Kept items (with no leading spaces; but first item special as it will have lost a leading space due to f-expansion) will lose a brace pair under \xintKeep:f:csv if the first argument was positive and strictly less than the length of the list. This differs of course from \xintKeep (which always braces items it outputs when used with positive first argument) and also from \xintKeepUnbraced in the case when the whole list is kept. Actually the case of singleton list is special, and brace removal will happen then.

This behaviour was otherwise for releases earlier than 1.2j and may change again.

Directly usable names are provided, but these macros (and the behaviour as described above) are to be considered *unstable* for the time being.

20.33.1. \xintLength:f:csv

Added at 1.2g (2016/03/19).

Modified at 1.2j (2016/12/22). Contrarily to \xintLength from xintkernel this one expands its argument.

```
\def\xintLength:f:csv {\romannumeral0\xintlength:f:csv}%
\def\xintlength:f:csv #1%
{\long\def\xintlength:f:csv ##1{%
    \expandafter#1\the\numexpr\expandafter\XINT_length:f:csv_a
    \romannumeral`&&@##1\xint:,\xint:,\xint:,\xint:,%
      \xint:,\xint:,\xint:,\xint:,\xint:,%
      \xint_c_ix,\xint_c_viii,\xint_c_vii,\xint_c_vi,%
      \xint_c_v,\xint_c_iv,\xint_c_iii,\xint_c_ii,\xint_c_i,\xint_bye
    \relax
}}\xintlength:f:csv { }%
```

Must first check if empty list.

```
\long\def\XINT_length:f:csv_a #1%
{%
    \xint_gob_til_xint: #1\xint_c_\xint_bye\xint:%
    \XINT_length:f:csv_loop #1%
}%
\long\def\XINT_length:f:csv_loop #1,#2,#3,#4,#5,#6,#7,#8,#9,%
{%
    \xint_gob_til_xint: #9\XINT_length:f:csv_finish\xint:%
    \xint_c_ix+\XINT_length:f:csv_loop
}%
\def\XINT_length:f:csv_finish\xint:\xint_c_ix+\XINT_length:f:csv_loop
    #1,#2,#3,#4,#5,#6,#7,#8,#9,{#9\xint_bye}%
```

20.33.2. \xintLengthUpTo:f:csv

Added at 1.2j (2016/12/22). \added{1.2j}\xintLengthUpTo:f:csv{N}{comma-list}. No ending comma. Returns -0 if length>N, else returns difference N-length. **N must be non-negative!!** Attention to the dot after \xint_bye for the loop interface.

```
\def\xintLengthUpTo:f:csv {\romannumeral0\xintlengthupto:f:csv}%
\long\def\xintlengthupto:f:csv #1#2%
{%
    \expandafter\XINT_lengthupto:f:csv_a
    \the\numexpr#1\expandafter.%
    \romannumeral`&&@#2\xint:,\xint:,\xint:,\xint:,%
         \xint:,\xint:,\xint:,\xint:,%
         \xint_c_viii,\xint_c_vii,\xint_c_vi,\xint_c_v,%
```

```
          \xint_c_iv,\xint_c_iii,\xint_c_ii,\xint_c_i,\xint_bye.%
}%
```

Must first recognize if empty list. If this is the case, return N.

```
\long\def\XINT_lengthupto:f:csv_a #1.#2%
{%
    \xint_gob_til_xint: #2\XINT_lengthupto:f:csv_empty\xint:%
    \XINT_lengthupto:f:csv_loop_b #1.#2%
}%
\def\XINT_lengthupto:f:csv_empty\xint:%
    \XINT_lengthupto:f:csv_loop_b #1.#2\xint_bye.{ #1}%
\def\XINT_lengthupto:f:csv_loop_a #1%
{%
    \xint_UDsignfork
      #1\XINT_lengthupto:f:csv_gt
       -\XINT_lengthupto:f:csv_loop_b
    \krof #1%
}%
\long\def\XINT_lengthupto:f:csv_gt #1\xint_bye.{-0}%
\long\def\XINT_lengthupto:f:csv_loop_b #1.#2,#3,#4,#5,#6,#7,#8,#9,%
{%
    \xint_gob_til_xint: #9\XINT_lengthupto:f:csv_finish_a\xint:%
    \expandafter\XINT_lengthupto:f:csv_loop_a\the\numexpr #1-\xint_c_viii.%
}%
\def\XINT_lengthupto:f:csv_finish_a\xint:
    \expandafter\XINT_lengthupto:f:csv_loop_a
    \the\numexpr #1-\xint_c_viii.#2,#3,#4,#5,#6,#7,#8,#9,%
{%
    \expandafter\XINT_lengthupto:f:csv_finish_b\the\numexpr #1-#9\xint_bye
}%
\def\XINT_lengthupto:f:csv_finish_b #1#2.%
{%
    \xint_UDsignfork
       #1{-0}%
        -{ #1#2}%
    \krof
}%
```

20.33.3. \xintKeep:f:csv

Added at 1.2g (2016/03/19) [on 2016/03/17].

Modified at 1.2j (2016/12/22). Redone with use of \xintLengthUpTo:f:csv. Same code skeleton as \xintKeep but handling comma separated but non terminated lists has complications. The \xintKeep in case of a negative #1 uses \xintgobble, we don't have that for comma delimited items, hence we do a special loop here (this style of loop is surely competitive with xintgobble for a few dozens items and even more). The loop knows before starting that it will not go too far.

```
\def\xintKeep:f:csv {\romannumeral0\xintkeep:f:csv }%
\long\def\xintkeep:f:csv #1#2%
{%
    \expandafter\xint_stop_aftergobble
    \romannumeral0\expandafter\XINT_keep:f:csv_a
    \the\numexpr #1\expandafter.\expandafter{\romannumeral`&&@#2}%
```

```
}%
\def\XINT_keep:f:csv_a #1%
{%
    \xint_UDzerominusfork
        #1-\XINT_keep:f:csv_keepnone
        0#1\XINT_keep:f:csv_neg
         0-{\XINT_keep:f:csv_pos #1}%
    \krof
}%
\long\def\XINT_keep:f:csv_keepnone .#1{,}%
\long\def\XINT_keep:f:csv_neg #1.#2%
{%
    \expandafter\XINT_keep:f:csv_neg_done\expandafter,%
    \romannumeral0%
    \expandafter\XINT_keep:f:csv_neg_a\the\numexpr
    #1-\numexpr\XINT_length:f:csv_a
    #2\xint:,\xint:,\xint:,\xint:,%
      \xint:,\xint:,\xint:,\xint:,\xint:,%
      \xint_c_ix,\xint_c_viii,\xint_c_vii,\xint_c_vi,%
      \xint_c_v,\xint_c_iv,\xint_c_iii,\xint_c_ii,\xint_c_i,\xint_bye
    .#2\xint_bye
}%
\def\XINT_keep:f:csv_neg_a #1%
{%
    \xint_UDsignfork
        #1{\expandafter\XINT_keep:f:csv_trimloop\the\numexpr-\xint_c_ix+}%
         -\XINT_keep:f:csv_keepall
    \krof
}%
\def\XINT_keep:f:csv_keepall #1.{ }%
\long\def\XINT_keep:f:csv_neg_done #1\xint_bye{#1}%
\def\XINT_keep:f:csv_trimloop #1#2.%
{%
    \xint_gob_til_minus#1\XINT_keep:f:csv_trimloop_finish-%
    \expandafter\XINT_keep:f:csv_trimloop
    \the\numexpr#1#2-\xint_c_ix\expandafter.\XINT_keep:f:csv_trimloop_trimnine
}%
\long\def\XINT_keep:f:csv_trimloop_trimnine #1,#2,#3,#4,#5,#6,#7,#8,#9,{}%
\def\XINT_keep:f:csv_trimloop_finish-%
    \expandafter\XINT_keep:f:csv_trimloop
    \the\numexpr-#1-\xint_c_ix\expandafter.\XINT_keep:f:csv_trimloop_trimnine
    {\csname XINT_trim:f:csv_finish#1\endcsname}%
\long\def\XINT_keep:f:csv_pos #1.#2%
{%
    \expandafter\XINT_keep:f:csv_pos_fork
    \romannumeral0\XINT_lengthupto:f:csv_a
    #1.#2\xint:,\xint:,\xint:,\xint:,%
         \xint:,\xint:,\xint:,\xint:,%
         \xint_c_viii,\xint_c_vii,\xint_c_vi,\xint_c_v,%
         \xint_c_iv,\xint_c_iii,\xint_c_ii,\xint_c_i,\xint_bye.%
    .#1.{}#2\xint_bye%
}%
```

```
\def\XINT_keep:f:csv_pos_fork #1#2.%
{%
    \xint_UDsignfork
      #1{\expandafter\XINT_keep:f:csv_loop\the\numexpr-\xint_c_viii+}%
       -\XINT_keep:f:csv_pos_keepall
    \krof
}%
\long\def\XINT_keep:f:csv_pos_keepall #1.#2#3\xint_bye{,#3}%
\def\XINT_keep:f:csv_loop #1#2.%
{%
    \xint_gob_til_minus#1\XINT_keep:f:csv_loop_end-%
    \expandafter\XINT_keep:f:csv_loop
    \the\numexpr#1#2-\xint_c_viii\expandafter.\XINT_keep:f:csv_loop_pickeight
}%
\long\def\XINT_keep:f:csv_loop_pickeight
     #1#2,#3,#4,#5,#6,#7,#8,#9,{{#1,#2,#3,#4,#5,#6,#7,#8,#9}}%
\def\XINT_keep:f:csv_loop_end-\expandafter\XINT_keep:f:csv_loop
    \the\numexpr-#1-\xint_c_viii\expandafter.\XINT_keep:f:csv_loop_pickeight
    {\csname XINT_keep:f:csv_end#1\endcsname}%
\long\expandafter\def\csname XINT_keep:f:csv_end1\endcsname
   #1#2,#3,#4,#5,#6,#7,#8,#9\xint_bye {#1,#2,#3,#4,#5,#6,#7,#8}%
\long\expandafter\def\csname XINT_keep:f:csv_end2\endcsname
   #1#2,#3,#4,#5,#6,#7,#8\xint_bye {#1,#2,#3,#4,#5,#6,#7}%
\long\expandafter\def\csname XINT_keep:f:csv_end3\endcsname
   #1#2,#3,#4,#5,#6,#7\xint_bye {#1,#2,#3,#4,#5,#6}%
\long\expandafter\def\csname XINT_keep:f:csv_end4\endcsname
   #1#2,#3,#4,#5,#6\xint_bye {#1,#2,#3,#4,#5}%
\long\expandafter\def\csname XINT_keep:f:csv_end5\endcsname
   #1#2,#3,#4,#5\xint_bye {#1,#2,#3,#4}%
\long\expandafter\def\csname XINT_keep:f:csv_end6\endcsname
   #1#2,#3,#4\xint_bye {#1,#2,#3}%
\long\expandafter\def\csname XINT_keep:f:csv_end7\endcsname
   #1#2,#3\xint_bye {#1,#2}%
\long\expandafter\def\csname XINT_keep:f:csv_end8\endcsname
   #1#2\xint_bye {#1}%
```

20.33.4. \xintTrim:f:csv

Added at 1.2g (2016/03/19) [on 2016/03/17].

Modified at 1.2j (2016/12/22). Redone on the basis of new \xintTrim.

```
\def\xintTrim:f:csv {\romannumeral0\xinttrim:f:csv }%
\long\def\xinttrim:f:csv #1#2%
{%
    \expandafter\xint_stop_aftergobble
    \romannumeral0\expandafter\XINT_trim:f:csv_a
    \the\numexpr #1\expandafter.\expandafter{\romannumeral`&&@#2}%
}%
\def\XINT_trim:f:csv_a #1%
{%
    \xint_UDzerominusfork
        #1-\XINT_trim:f:csv_trimnone
        0#1\XINT_trim:f:csv_neg
```

```
          0-{\XINT_trim:f:csv_pos #1}%
    \krof
}%
\long\def\XINT_trim:f:csv_trimnone .#1{,#1}%
\long\def\XINT_trim:f:csv_neg #1.#2%
{%
    \expandafter\XINT_trim:f:csv_neg_a\the\numexpr
    #1-\numexpr\XINT_length:f:csv_a
    #2\xint:,\xint:,\xint:,\xint:,%
      \xint:,\xint:,\xint:,\xint:,\xint:,%
      \xint_c_ix,\xint_c_viii,\xint_c_vii,\xint_c_vi,%
      \xint_c_v,\xint_c_iv,\xint_c_iii,\xint_c_ii,\xint_c_i,\xint_bye
    .{}#2\xint_bye
}%
\def\XINT_trim:f:csv_neg_a #1%
{%
    \xint_UDsignfork
        #1{\expandafter\XINT_keep:f:csv_loop\the\numexpr-\xint_c_viii+}%
         -\XINT_trim:f:csv_trimall
    \krof
}%
\def\XINT_trim:f:csv_trimall {\expandafter,\xint_bye}%
\long\def\XINT_trim:f:csv_pos #1.#2%
{%
    \expandafter\XINT_trim:f:csv_pos_done\expandafter,%
    \romannumeral0%
    \expandafter\XINT_trim:f:csv_loop\the\numexpr#1-\xint_c_ix.%
     #2\xint:,\xint:,\xint:,\xint:,\xint:,%
       \xint:,\xint:,\xint:,\xint:,\xint:\xint_bye
}%
\def\XINT_trim:f:csv_loop #1#2.%
{%
    \xint_gob_til_minus#1\XINT_trim:f:csv_finish-%
    \expandafter\XINT_trim:f:csv_loop\the\numexpr#1#2\XINT_trim:f:csv_loop_trimnine
}%
\long\def\XINT_trim:f:csv_loop_trimnine #1,#2,#3,#4,#5,#6,#7,#8,#9,%
{%
    \xint_gob_til_xint: #9\XINT_trim:f:csv_toofew\xint:-\xint_c_ix.%
}%
\def\XINT_trim:f:csv_toofew\xint:{*\xint_c_}%
\def\XINT_trim:f:csv_finish-%
    \expandafter\XINT_trim:f:csv_loop\the\numexpr-#1\XINT_trim:f:csv_loop_trimnine
{%
    \csname XINT_trim:f:csv_finish#1\endcsname
}%
\long\expandafter\def\csname XINT_trim:f:csv_finish1\endcsname
  #1,#2,#3,#4,#5,#6,#7,#8,{ }%
\long\expandafter\def\csname XINT_trim:f:csv_finish2\endcsname
  #1,#2,#3,#4,#5,#6,#7,{ }%
\long\expandafter\def\csname XINT_trim:f:csv_finish3\endcsname
  #1,#2,#3,#4,#5,#6,{ }%
\long\expandafter\def\csname XINT_trim:f:csv_finish4\endcsname
```

```
  #1,#2,#3,#4,#5,{ }%
\long\expandafter\def\csname XINT_trim:f:csv_finish5\endcsname
  #1,#2,#3,#4,{ }%
\long\expandafter\def\csname XINT_trim:f:csv_finish6\endcsname
  #1,#2,#3,{ }%
\long\expandafter\def\csname XINT_trim:f:csv_finish7\endcsname
  #1,#2,{ }%
\long\expandafter\def\csname XINT_trim:f:csv_finish8\endcsname
  #1,{ }%
\expandafter\let\csname XINT_trim:f:csv_finish9\endcsname\space
\long\def\XINT_trim:f:csv_pos_done #1\xint:#2\xint_bye{#1}%
```

20.33.5. \xintNthEltPy:f:csv

Counts like Python starting at zero. Last refactored with 1.2j. Attention, makes currently no effort at removing leading spaces in the picked item.

```
\def\xintNthEltPy:f:csv {\romannumeral0\xintntheltpy:f:csv }%
\long\def\xintntheltpy:f:csv #1#2%
{%
   \expandafter\XINT_nthelt:f:csv_a
   \the\numexpr #1\expandafter.\expandafter{\romannumeral`&&@#2}%
}%
\def\XINT_nthelt:f:csv_a #1%
{%
    \xint_UDsignfork
        #1\XINT_nthelt:f:csv_neg
         -\XINT_nthelt:f:csv_pos
    \krof #1%
}%
\long\def\XINT_nthelt:f:csv_neg -#1.#2%
{%
    \expandafter\XINT_nthelt:f:csv_neg_fork
    \the\numexpr\XINT_length:f:csv_a
    #2\xint:,\xint:,\xint:,\xint:,%
      \xint:,\xint:,\xint:,\xint:,\xint:,%
      \xint_c_ix,\xint_c_viii,\xint_c_vii,\xint_c_vi,%
      \xint_c_v,\xint_c_iv,\xint_c_iii,\xint_c_ii,\xint_c_i,\xint_bye
    -#1.#2,\xint_bye
}%
\def\XINT_nthelt:f:csv_neg_fork #1%
{%
    \if#1-\expandafter\xint_stop_afterbye\fi
    \expandafter\XINT_nthelt:f:csv_neg_done
    \romannumeral0%
    \expandafter\XINT_keep:f:csv_trimloop\the\numexpr-\xint_c_ix+#1%
}%
\long\def\XINT_nthelt:f:csv_neg_done#1,#2\xint_bye{ #1}%
\long\def\XINT_nthelt:f:csv_pos #1.#2%
{%
    \expandafter\XINT_nthelt:f:csv_pos_done
    \romannumeral0%
    \expandafter\XINT_trim:f:csv_loop\the\numexpr#1-\xint_c_ix.%
    #2\xint:,\xint:,\xint:,\xint:,\xint:,%
```

```
      \xint:,\xint:,\xint:,\xint:,\xint:,\xint_bye
}%
\def\XINT_nthelt:f:csv_pos_done #1{%
\long\def\XINT_nthelt:f:csv_pos_done ##1,##2\xint_bye{%
  \xint_gob_til_xint:##1\XINT_nthelt:f:csv_pos_cleanup\xint:#1##1}%
}\XINT_nthelt:f:csv_pos_done{ }%
```

This strange thing is in case the picked item was the last one, hence there was an ending `\xint:` (we could not put a comma earlier for matters of not confusing empty list with a singleton list), and we do this here to activate brace-stripping of item as all other items may be brace-stripped if picked. This is done for coherence. Of course, in the context of the xintexpr.sty parsers, there are no braces in list items...

```
\xint_firstofone{\long\def\XINT_nthelt:f:csv_pos_cleanup\xint:} %
   #1\xint:{ #1}%
```

20.33.6. \xintReverse:f:csv

Added at 1.2g (2016/03/19) [on 2016/03/17]. Contrarily to `\xintReverseOrder` from xintkernel.sty, this one expands its argument. Handles empty list too. .

Modified at 1.2j (2016/12/22). Made `\long`.

```
\def\xintReverse:f:csv {\romannumeral0\xintreverse:f:csv }%
\long\def\xintreverse:f:csv #1%
{%
    \expandafter\XINT_reverse:f:csv_loop
    \expandafter{\expandafter}\romannumeral`&&@#1,%
      \xint:,%
        \xint_bye,\xint_bye,\xint_bye,\xint_bye,%
        \xint_bye,\xint_bye,\xint_bye,\xint_bye,%
      \xint:
}%
\long\def\XINT_reverse:f:csv_loop #1#2,#3,#4,#5,#6,#7,#8,#9,%
{%
    \xint_bye #9\XINT_reverse:f:csv_cleanup\xint_bye
    \XINT_reverse:f:csv_loop {,#9,#8,#7,#6,#5,#4,#3,#2#1}%
}%
\long\def\XINT_reverse:f:csv_cleanup\xint_bye\XINT_reverse:f:csv_loop #1#2\xint:
{%
    \XINT_reverse:f:csv_finish #1%
}%
\long\def\XINT_reverse:f:csv_finish #1\xint:,{ }%
```

20.33.7. \xintFirstItem:f:csv

Added at 1.2k (2017/01/06). For use by first() in `\xintexpr`-essions, and some amount of compatibility with `\xintNewExpr`.

```
\def\xintFirstItem:f:csv {\romannumeral0\xintfirstitem:f:csv}%
\long\def\xintfirstitem:f:csv #1%
{%
    \expandafter\XINT_first:f:csv_a\romannumeral`&&@#1,\xint_bye
}%
\long\def\XINT_first:f:csv_a #1,#2\xint_bye{ #1}%
```

20.33.8. \xintLastItem:f:csv

Added at 1.2k (2017/01/06). Based on and sharing code with xintkernel's \xintLastItem from 1.2i. Output empty if input empty. f-expands its argument (hence first item, if not protected.) For use by last() in \xintexpr-essions with to some extent \xintNewExpr compatibility.

```
\def\xintLastItem:f:csv {\romannumeral0\xintlastitem:f:csv}%
\long\def\xintlastitem:f:csv #1%
{%
    \expandafter\XINT_last:f:csv_loop\expandafter{\expandafter}\expandafter.%
    \romannumeral`&&@#1,%
    \xint:\XINT_last_loop_enda,\xint:\XINT_last_loop_endb,%
    \xint:\XINT_last_loop_endc,\xint:\XINT_last_loop_endd,%
    \xint:\XINT_last_loop_ende,\xint:\XINT_last_loop_endf,%
    \xint:\XINT_last_loop_endg,\xint:\XINT_last_loop_endh,\xint_bye
}%
\long\def\XINT_last:f:csv_loop #1.#2,#3,#4,#5,#6,#7,#8,#9,%
{%
    \xint_gob_til_xint: #9%
        {#8}{#7}{#6}{#5}{#4}{#3}{#2}{#1}\xint:
    \XINT_last:f:csv_loop {#9}.%
}%
```

20.33.9. \xintKeep:x:csv

Added at 1.2j (2016/12/22). To xintexpr. Moved here at 1.4. Not part of publicly supported macros, may be removed at any time.

```
\def\xintKeep:x:csv #1#2%
{%
    \expandafter\xint_gobble_i
    \romannumeral0\expandafter\XINT_keep:x:csv_pos
    \the\numexpr #1\expandafter.\expandafter{\romannumeral`&&@#2}%
}%
\def\XINT_keep:x:csv_pos #1.#2%
{%
    \expandafter\XINT_keep:x:csv_loop\the\numexpr#1-\xint_c_viii.%
    #2\xint_Bye,\xint_Bye,\xint_Bye,\xint_Bye,%
       \xint_Bye,\xint_Bye,\xint_Bye,\xint_Bye,\xint_bye
}%
\def\XINT_keep:x:csv_loop #1%
{%
    \xint_gob_til_minus#1\XINT_keep:x:csv_finish-%
    \XINT_keep:x:csv_loop_pickeight #1%
}%
\def\XINT_keep:x:csv_loop_pickeight #1.#2,#3,#4,#5,#6,#7,#8,#9,%
{%
    ,#2,#3,#4,#5,#6,#7,#8,#9%
    \expandafter\XINT_keep:x:csv_loop\the\numexpr#1-\xint_c_viii.%
}%
\def\XINT_keep:x:csv_finish-\XINT_keep:x:csv_loop_pickeight -#1.%
{%
    \csname XINT_keep:x:csv_finish#1\endcsname
}%
```

```
\expandafter\def\csname XINT_keep:x:csv_finish1\endcsname
  #1,#2,#3,#4,#5,#6,#7,{,#1,#2,#3,#4,#5,#6,#7\xint_Bye}%
\expandafter\def\csname XINT_keep:x:csv_finish2\endcsname
  #1,#2,#3,#4,#5,#6,{,#1,#2,#3,#4,#5,#6\xint_Bye}%
\expandafter\def\csname XINT_keep:x:csv_finish3\endcsname
  #1,#2,#3,#4,#5,{,#1,#2,#3,#4,#5\xint_Bye}%
\expandafter\def\csname XINT_keep:x:csv_finish4\endcsname
  #1,#2,#3,#4,{,#1,#2,#3,#4\xint_Bye}%
\expandafter\def\csname XINT_keep:x:csv_finish5\endcsname
  #1,#2,#3,{,#1,#2,#3\xint_Bye}%
\expandafter\def\csname XINT_keep:x:csv_finish6\endcsname
  #1,#2,{,#1,#2\xint_Bye}%
\expandafter\def\csname XINT_keep:x:csv_finish7\endcsname
  #1,{,#1\xint_Bye}%
\expandafter\let\csname XINT_keep:x:csv_finish8\endcsname\xint_Bye
```

20.33.10. Public names for the undocumented csv macros: \xintCSVLength, \xintCSVKeep, \xintCSVKeepx, \xintCSVTrim, \xintCSVNthEltPy, \xintCSVReverse, \xintCSVFirstItem, \xintCSVLastItem

Completely unstable macros: currently they expand the list argument and want no final comma. But for matters of xintexpr.sty I could as well decide to require a final comma, and then I could simplify implementation but of course this would break the macros if used with current functionalities.

```
\let\xintCSVLength   \xintLength:f:csv
\let\xintCSVKeep     \xintKeep:f:csv
\let\xintCSVKeepx    \xintKeep:x:csv
\let\xintCSVTrim     \xintTrim:f:csv
\let\xintCSVNthEltPy \xintNthEltPy:f:csv
\let\xintCSVReverse  \xintReverse:f:csv
\let\xintCSVFirstItem\xintFirstItem:f:csv
\let\xintCSVLastItem \xintLastItem:f:csv
\let\XINT_tmpa\relax \let\XINT_tmpb\relax \let\XINT_tmpc\relax
\XINTrestorecatcodesendinput%
```

21. Package xintcore implementation

Contents

.1 Introduction . . . 315
.2 Catcodes, ε-TEX and reload detection . . . 315
.3 Package identification . . . 316
.4 Legacy error reporting . . . 316
Routines handling integers as lists of token digits 317
.5 \xintiiSgn . . . 317
.6 \xintiiOpp . . . 317
.7 \xintiiAbs . . . 317
.8 \xintFDg . . . 318
.9 \xintLDg . . . 318
.10 \xintDouble . . . 318
.11 \xintHalf . . . 319
.12 \xintInc . . . 319
.13 \xintDec . . . 320
.14 \xintDSL . . . 320
.15 \xintDSR . . . 321
.16 \xintDSRr . . . 321
Blocks of eight digits . . . 322
.17 \XINT_cuz . . . 322
.18 \XINT_cuz_byviii . . . 322
.19 \XINT_sepbyviii_Z . . . 322
.20 \XINT_sepbyviii_andcount . . . 323
.21 \XINT_sepandrev . . . 323
Core arithmetic . . . 324
.22 \xintiiAdd . . . 324
.23 \xintiiCmp . . . 324
.24 \xintiiSub . . . 325
.25 \xintiiMul . . . 326
.26 Reversed eight-digit block output . . . 326
.27 \xintiiDivision . . . 326
.28 \xintiiSqr . . . 327
.29 \xintiiPow . . . 327
.30 \xintiiFac . . . 328
.31 \XINT_useiimessage . . . 329

21.1. Introduction

Got split off from xint with release 1.1.

The core arithmetic routines have been entirely rewritten for release 1.2. The 1.2i and 1.2l brought again some improvements.

The commenting continues (2026/11/26) to be very sparse: actually it got worse than ever with release 1.2. I will possibly add comments at a later date, but for the time being the new routines are not commented at all.

1.3 removes all macros which were deprecated at 1.2o.

21.2. Catcodes, ε-TEX and reload detection

The code for reload detection was initially copied from Heiko Oberdiek's packages, then modified.

The method for catcodes was also initially directly inspired by these packages.

```
\begingroup\catcode61\catcode48\catcode32=10\relax%
  \catcode13=5    % ^^M
  \endlinechar=13 %
  \catcode123=1   % {
  \catcode125=2   % }
  \catcode64=11   % @
  \catcode44=12   % ,
  \catcode46=12   % .
  \catcode58=12   % :
  \catcode94=7    % ^
  \def\empty{}\def\space{ }\newlinechar10
  \def\z{\endgroup}%
  \expandafter\let\expandafter\x\csname ver@xintcore.sty\endcsname
  \expandafter\let\expandafter\c\csname ver@calxcore.sty\endcsname
  \expandafter\ifx\csname numexpr\endcsname\relax
    \expandafter\ifx\csname PackageWarningNoLine\endcsname\relax
      \immediate\write128{^^JPackage xintcore Warning:^^J%
```

```
                          \space\space\space\space
                          \numexpr not available, aborting input.^^J}%
    \else
      \PackageWarningNoLine{xintcore}{\numexpr not available, aborting input}%
    \fi
    \def\z{\endgroup\endinput}%
  \else
    \ifx\x\relax   % plain-TeX, first loading of xintcore.sty
      \ifx\c\relax % but calxcore.sty not yet loaded.
         \def\z{\endgroup\input calxcore.sty\relax}%
      \fi
    \else
      \ifx\x\empty % LaTeX, first loading,
      % variable is initialized, but \ProvidesPackage not yet seen
          \ifx\c\relax % calxcore.sty not yet loaded.
            \def\z{\endgroup\RequirePackage{calxcore}}%
          \fi
      \else
        \def\z{\endgroup\endinput}% xintkernel already loaded.
      \fi
    \fi
  \fi
\z%
\XINTsetupcatcodes% defined in xintkernel.sty
```

21.3. Package identification

```
\XINT_providespackage
\ProvidesPackage{xintcore}%
  [2026/11/26 v2.0a2 Arithmetic on big integers (JFB)]%
```

21.4. Legacy error reporting

Legacy integer and rational routines cannot represent NaN or infinity. Their error path always raises the condition's signal flag, emits an expandable error, and returns the caller's fallback value. There are no configurable traps or handlers.

Syntax (f-expandable):
\XINT_legacysignalcondition{C}{M}{H}{V} -> <error>V
H is ignored, as in the original trapped-signal path.

```
\catcode`- 11
\def\XINT_ConversionSyntax-signal  {{InvalidOperation}}%
\let\XINT_DivisionImpossible-signal\XINT_ConversionSyntax-signal
\let\XINT_DivisionUndefined-signal \XINT_ConversionSyntax-signal
\let\XINT_InvalidContext-signal    \XINT_ConversionSyntax-signal
\catcode`- 12
\def\XINT_legacysignalcondition #1{\expandafter\XINT_legacysignalcondition_a
    \romannumeral0\ifcsname XINT_#1-signal\endcsname
                    \xint_dothis{\csname XINT_#1-signal\endcsname}%
                  \fi\xint_orthat{{#1}}{#1}}%
\def\XINT_legacysignalcondition_a #1#2#3#4#5{%
    \expandafter\xint_gobble_i\csname XINT_#1Flag_ON\endcsname
    \XINT_expandableerror{#3}%
    \xint_firstofone{#5}%
```

```
}%
\def\XINT_ifFlagRaised #1{%
    \ifcsname XINT_#1Flag_ON\endcsname
        \expandafter\xint_firstoftwo
    \else
        \expandafter\xint_secondoftwo
    \fi}%
\def\XINT_resetFlag #1%
    {\expandafter\let\csname XINT_#1Flag_ON\endcsname\XINT_undefined}%
\def\XINT_resetFlags {%
    \XINT_resetFlag{InvalidOperation}%
    \XINT_resetFlag{DivisionByZero}%
    \XINT_resetFlag{Underflow}%
    \XINT_resetFlag{Overflow}%
}%
\def\XINT_RaiseFlag #1{\expandafter\xint_gobble_i\csname XINT_#1Flag_ON\endcsname}%
```

Routines handling integers as lists of token digits

Routines handling big integers which are lists of digit tokens with no special additional structure.

Some routines do not accept non properly terminated inputs like "\the\numexpr1", or "\the\mathcode`\-", others do.

These routines or their sub-routines are mainly for internal usage.

21.5. \xintiiSgn

Moved to calxcore for STUPID reasons.

21.6. \xintiiOpp

Moved to calxcore for STUPID reasons.

21.7. \xintiiAbs

Attention \xintiiAbs non robust against non terminated input.

```
\def\xintiiAbs {\romannumeral0\xintiiabs }%
\def\xintiiabs #1%
{%
    \expandafter\XINT_abs \romannumeral`&&@#1%
}%
\def\XINT_abs #1%
{%
    \xint_UDsignfork
      #1{ }%
       -{ #1}%
    \krof
}%
\def\XINT_Abs #1%
{%
    \xint_UDsignfork
      #1{}%
       -{#1}%
```

```
    \krof
}%
```

21.8. \xintFDg

FIRST DIGIT.
  1.2l: \xintiiFDg made robust against non terminated input.
  1.2o deprecates \xintiiFDg, gives to \xintFDg former meaning of \xintiiFDg.

```
\def\xintFDg {\romannumeral0\xintfdg }%
\def\xintfdg #1{\expandafter\XINT_fdg \romannumeral`&&@#1\xint:\Z}%
\def\XINT_FDg #1%
   {\romannumeral0\expandafter\XINT_fdg\romannumeral`&&@\xintnum{#1}\xint:\Z }%
\def\XINT_fdg #1#2#3\Z
{%
    \xint_UDzerominusfork
      #1-{ 0}%   zero
      0#1{ #2}%  negative
       0-{ #1}%  positive
    \krof
}%
```

21.9. \xintLDg

LAST DIGIT.
  Rewritten for 1.2i (2016/12/10). Surprisingly perhaps, it is faster than \xintLastItem from xintkernel.sty despite the \numexpr operations.
  1.2o deprecates \xintiiLDg, gives to \xintLDg former meaning of \xintiiLDg.
  Attention \xintLDg non robust against non terminated input.

```
\def\xintLDg {\romannumeral0\xintldg }%
\def\xintldg #1{\expandafter\XINT_ldg_fork\romannumeral`&&@#1%
    \XINT_ldg_c{}{}{}{}{}{}{}{}\xint_bye\relax}%
\def\XINT_ldg_fork #1%
{%
    \xint_UDsignfork
      #1\XINT_ldg
       -{\XINT_ldg#1}%
    \krof
}%
\def\XINT_ldg #1{%
\def\XINT_ldg ##1##2##3##4##5##6##7##8##9%
   {\expandafter#1%
    \the\numexpr##9##8##7##6##5##4##3##2##1*\xint_c_+\XINT_ldg_a##9}%
}\XINT_ldg{ }%
\def\XINT_ldg_a#1#2{\XINT_ldg_cbye#2\XINT_ldg_d#1\XINT_ldg_c\XINT_ldg_b#2}%
\def\XINT_ldg_b#1#2#3#4#5#6#7#8#9{#9#8#7#6#5#4#3#2#1*\xint_c_+\XINT_ldg_a#9}%
\def\XINT_ldg_c    #1#2\xint_bye{#1}%
\def\XINT_ldg_cbye #1\XINT_ldg_c{}%
\def\XINT_ldg_d#1#2\xint_bye{#1}%
```

21.10. \xintDouble

Attention \xintDouble non robust against non terminated input.

```
\def\xintDouble {\romannumeral0\xintdouble}%
\def\xintdouble #1{\expandafter\XINT_dbl_fork\romannumeral`&&@#1%
                   \xint_bye2345678\xint_bye*\xint_c_ii\relax}%
\def\XINT_dbl_fork #1%
{%
    \xint_UDsignfork
     #1\XINT_dbl_neg
      -\XINT_dbl
    \krof #1%
}%
\def\XINT_dbl_neg-{\expandafter-\romannumeral0\XINT_dbl}%
\def\XINT_dbl #1{%
\def\XINT_dbl ##1##2##3##4##5##6##7##8%
   {\expandafter#1\the\numexpr##1##2##3##4##5##6##7##8\XINT_dbl_a}%
}\XINT_dbl{ }%
\def\XINT_dbl_a #1#2#3#4#5#6#7#8%
   {\expandafter\XINT_dbl_e\the\numexpr 1#1#2#3#4#5#6#7#8\XINT_dbl_a}%
\def\XINT_dbl_e#1{*\xint_c_ii\if#13+\xint_c_i\fi\relax}%
```

21.11. \xintHalf

Attention \xintHalf non robust against non terminated input.

```
\def\xintHalf {\romannumeral0\xinthalf}%
\def\xinthalf #1{\expandafter\XINT_half_fork\romannumeral`&&@#1%
    \xint_bye\xint_Bye345678\xint_bye
    *\xint_c_v+\xint_c_v)/\xint_c_x-\xint_c_i\relax}%
\def\XINT_half_fork #1%
{%
  \xint_UDsignfork
     #1\XINT_half_neg
      -\XINT_half
    \krof #1%
}%
\def\XINT_half_neg-{\xintiiopp\XINT_half}%
\def\XINT_half #1{%
\def\XINT_half ##1##2##3##4##5##6##7##8%
   {\expandafter#1\the\numexpr(##1##2##3##4##5##6##7##8\XINT_half_a}%
}\XINT_half{ }%
\def\XINT_half_a#1{\xint_Bye#1\xint_bye\XINT_half_b#1}%
\def\XINT_half_b #1#2#3#4#5#6#7#8%
   {\expandafter\XINT_half_e\the\numexpr(1#1#2#3#4#5#6#7#8\XINT_half_a}%
\def\XINT_half_e#1{*\xint_c_v+#1-\xint_c_v)\relax}%
```

21.12. \xintInc

1.2i much delayed complete rewrite in 1.2 style.

As we take 9 by 9 with the input save stack at 5000 this allows a bit less than 9 times 2500 = 22500 digits on input.

Attention \xintInc non robust against non terminated input.

```
\def\xintInc {\romannumeral0\xintinc}%
\def\xintinc #1{\expandafter\XINT_inc_fork\romannumeral`&&@#1%
                \xint_bye23456789\xint_bye+\xint_c_i\relax}%
```

```
\def\XINT_inc_fork #1%
{%
    \xint_UDsignfork
      #1\XINT_inc_neg
       -\XINT_inc
    \krof #1%
}%
\def\XINT_inc_neg-#1\xint_bye#2\relax
   {\xintiiopp\XINT_dec #1\XINT_dec_bye234567890\xint_bye}%
\def\XINT_inc #1{%
\def\XINT_inc ##1##2##3##4##5##6##7##8##9%
   {\expandafter#1\the\numexpr##1##2##3##4##5##6##7##8##9\XINT_inc_a}%
}\XINT_inc{ }%
\def\XINT_inc_a #1#2#3#4#5#6#7#8#9%
   {\expandafter\XINT_inc_e\the\numexpr 1#1#2#3#4#5#6#7#8#9\XINT_inc_a}%
\def\XINT_inc_e#1{\if#12+\xint_c_i\fi\relax}%
```

21.13. \xintDec

1.2i much delayed complete rewrite in the 1.2 style. Things are a bit more complicated than \xintInc because 2999999999 is too big for TeX.

Attention \xintDec non robust against non terminated input.

```
\def\xintDec {\romannumeral0\xintdec}%
\def\xintdec #1{\expandafter\XINT_dec_fork\romannumeral`&&@#1%
                \XINT_dec_bye234567890\xint_bye}%
\def\XINT_dec_fork #1%
{%
    \xint_UDsignfork
      #1\XINT_dec_neg
       -\XINT_dec
    \krof #1%
}%
\def\XINT_dec_neg-#1\XINT_dec_bye#2\xint_bye
   {\expandafter-%
    \romannumeral0\XINT_inc #1\xint_bye23456789\xint_bye+\xint_c_i\relax}%
\def\XINT_dec #1{%
\def\XINT_dec ##1##2##3##4##5##6##7##8##9%
   {\expandafter#1\the\numexpr##1##2##3##4##5##6##7##8##9\XINT_dec_a}%
}\XINT_dec{ }%
\def\XINT_dec_a #1#2#3#4#5#6#7#8#9%
   {\expandafter\XINT_dec_e\the\numexpr 1#1#2#3#4#5#6#7#8#9\XINT_dec_a}%
\def\XINT_dec_bye #1\XINT_dec_a#2#3\xint_bye
         {\if#20-\xint_c_ii\relax+\else-\fi\xint_c_i\relax}%
\def\XINT_dec_e#1{\unless\if#11\xint_dothis{-\xint_c_i#1}\fi\xint_orthat\relax}%
```

21.14. \xintDSL

DECIMAL SHIFT LEFT (=MULTIPLICATION PAR 10). Rewritten for 1.2i. This was very old code... I never came back to it, but I should have rewritten it long time ago.

Attention \xintDSL non robust against non terminated input.

```
\def\xintDSL {\romannumeral0\xintdsl }%
\def\xintdsl #1{\expandafter\XINT_dsl\romannumeral`&&@#10}%
```

```
\def\XINT_dsl#1{%
\def\XINT_dsl ##1{\xint_gob_til_zero ##1\xint_dsl_zero 0#1##1}%
}\XINT_dsl{ }%
\def\xint_dsl_zero 0 0{ }%
```

21.15. \xintDSR

Decimal shift right, truncates towards zero. Rewritten for 1.2i. Limited to 22483 digits on input.

Attention \xintDSR non robust against non terminated input.

```
\def\xintDSR{\romannumeral0\xintdsr}%
\def\xintdsr #1{\expandafter\XINT_dsr_fork\romannumeral`&&@#1%
    \xint_bye\xint_Bye3456789\xint_bye+\xint_c_v)/\xint_c_x-\xint_c_i\relax}%
\def\XINT_dsr_fork #1%
{%
    \xint_UDsignfork
      #1\XINT_dsr_neg
       -\XINT_dsr
    \krof #1%
}%
\def\XINT_dsr_neg-{\xintiiopp\XINT_dsr}%
\def\XINT_dsr #1{%
\def\XINT_dsr ##1##2##3##4##5##6##7##8##9%
   {\expandafter#1\the\numexpr(##1##2##3##4##5##6##7##8##9\XINT_dsr_a}%
}\XINT_dsr{ }%
\def\XINT_dsr_a#1{\xint_Bye#1\xint_bye\XINT_dsr_b#1}%
\def\XINT_dsr_b #1#2#3#4#5#6#7#8#9%
   {\expandafter\XINT_dsr_e\the\numexpr(1#1#2#3#4#5#6#7#8#9\XINT_dsr_a}%
\def\XINT_dsr_e #1{)\relax}%
```

21.16. \xintDSRr

New with 1.2i. Decimal shift right, rounds away from zero; done in the 1.2 spirit (with much delay, sorry). Used by \xintRound, \xintDivRound.

This is about the first time I am happy that the division in \numexpr rounds!

Attention \xintDSRr non robust against non terminated input.

```
\def\xintDSRr{\romannumeral0\xintdsrr}%
\def\xintdsrr #1{\expandafter\XINT_dsrr_fork\romannumeral`&&@#1%
                \xint_bye\xint_Bye3456789\xint_bye/\xint_c_x\relax}%
\def\XINT_dsrr_fork #1%
{%
    \xint_UDsignfork
      #1\XINT_dsrr_neg
       -\XINT_dsrr
    \krof #1%
}%
\def\XINT_dsrr_neg-{\xintiiopp\XINT_dsrr}%
\def\XINT_dsrr #1{%
\def\XINT_dsrr ##1##2##3##4##5##6##7##8##9%
   {\expandafter#1\the\numexpr##1##2##3##4##5##6##7##8##9\XINT_dsrr_a}%
}\XINT_dsrr{ }%
\def\XINT_dsrr_a#1{\xint_Bye#1\xint_bye\XINT_dsrr_b#1}%
```

```
\def\XINT_dsrr_b #1#2#3#4#5#6#7#8#9%
   {\expandafter\XINT_dsrr_e\the\numexpr1#1#2#3#4#5#6#7#8#9\XINT_dsrr_a}%
\let\XINT_dsrr_e\XINT_inc_e
```

Blocks of eight digits

The lingua of release 1.2. Six of them have been moved to calxcore at 2.0a1. An \XINT_smallunrev byviii was deleted then as it served only in the legacy addition macros, now replaced by those of calxcore.

21.17. \XINT_cuz

This (launched by \romannumeral0) iterately removes all leading zeroes from a sequence of 8N digits ended by \R.

Rewritten for 1.21, now uses \numexpr governed expansion and \ifnum test rather than delimited gobbling macros.

Note 2015/11/28: with only four digits the gob_til_fourzeroes had proved in some old testing faster than \ifnum test. But with eight digits, the execution times are much closer, as I tested back then.

```
\def\XINT_cuz #1{%
\def\XINT_cuz {\expandafter#1\the\numexpr\XINT_cuz_loop}%
}\XINT_cuz{ }%
\def\XINT_cuz_loop #1#2#3#4#5#6#7#8#9%
{%
    #1#2#3#4#5#6#7#8%
       \xint_gob_til_R #9\XINT_cuz_hitend\R
       \ifnum #1#2#3#4#5#6#7#8>\xint_c_
             \expandafter\XINT_cuz_cleantoend
       \else\expandafter\XINT_cuz_loop
       \fi #9%
}%
\def\XINT_cuz_hitend\R #1\R{\relax}%
\def\XINT_cuz_cleantoend #1\R{\relax #1}%
```

21.18. \XINT_cuz_byviii

This removes eight by eight leading zeroes from a sequence of 8N digits ended by \R. Thus, we still have 8N digits on output. Expansion started by \romannumeral0

```
\def\XINT_cuz_byviii #1#2#3#4#5#6#7#8#9%
{%
    \xint_gob_til_R #9\XINT_cuz_byviii_e \R
    \xint_gob_til_eightzeroes #1#2#3#4#5#6#7#8\XINT_cuz_byviii_z 00000000%
    \XINT_cuz_byviii_done #1#2#3#4#5#6#7#8#9%
}%
\def\XINT_cuz_byviii_z 00000000\XINT_cuz_byviii_done 00000000{\XINT_cuz_byviii}%
\def\XINT_cuz_byviii_done #1\R { #1}%
\def\XINT_cuz_byviii_e\R #1\XINT_cuz_byviii_done #2\R{ #2}%
```

21.19. \XINT_sepbyviii_Z

This is used as

```
   \the\numexpr\XINT_sepbyviii_Z <8Ndigits>\XINT_sepbyviii_Z_end 2345678\relax
```

It produces 1<8d>!...1<8d>!1;!

Prior to 1.2l it used \Z as terminator (hence the name). At 1.2l a switch to ; was done. This was at a time I thought perhaps I would use an internal format maintaining such 8 digits blocks, and this had to be compatible with the \csname...\endcsname encapsulation in \xintexpr parsers. That rationale is obsolete since 1.4 usage of \expanded in xintexpr, but if an internal format is one day used it would be nice to be able to externalize it easily, so catcode 12 tokens are the most convenient.

As the expansion is done via successive \numexpr, it is convenient to use something such as ; which terminates it and stays there. The ! itself would do. But it proved convenient to have unique ending pattern. I could use two ! perhaps.

Modified at 1.4n (2025/09/05). Unfortunately LuaMetaTeX has given to ; and : a meaning as operators inside its \numexpr. Let's hope ! does not acquire meaning too there. Anyway here I only need to use \relax;! in \XINT_sepbyviii_Z_end. First step in a tedious check of the entire codebase...

```
\def\XINT_sepbyviii_Z #1#2#3#4#5#6#7#8%
{%
    1#1#2#3#4#5#6#7#8\expandafter!\the\numexpr\XINT_sepbyviii_Z
}%
\def\XINT_sepbyviii_Z_end #1\relax {\relax;!}%
```

21.20. \XINT_sepbyviii_andcount

This is used as

```
\the\numexpr\XINT_sepbyviii_andcount <8Ndigits>%
    \XINT_sepbyviii_end 2345678\relax
    \xint_c_vii!\xint_c_vi!\xint_c_v!\xint_c_iv!%
    \xint_c_iii!\xint_c_ii!\xint_c_i!\xint_c_\W
```

It will produce

```
1<8d>!1<8d>!....1<8d>!1\xint:<count of blocks>\xint:
```

Used by \XINT_div_prepare_g for \XINT_div_prepare_h, and also by \xintiiCmp.

```
\def\XINT_sepbyviii_andcount
{%
    \expandafter\XINT_sepbyviii_andcount_a\the\numexpr\XINT_sepbyviii
}%
\def\XINT_sepbyviii #1#2#3#4#5#6#7#8%
{%
    1#1#2#3#4#5#6#7#8\expandafter!\the\numexpr\XINT_sepbyviii
}%
\def\XINT_sepbyviii_end #1\relax {\relax\XINT_sepbyviii_andcount_end!}%
\def\XINT_sepbyviii_andcount_a {\XINT_sepbyviii_andcount_b \xint_c_\xint:}%
\def\XINT_sepbyviii_andcount_b #1\xint:#2!#3!#4!#5!#6!#7!#8!#9!%
{%
    #2\expandafter!\the\numexpr#3\expandafter!\the\numexpr#4\expandafter
    !\the\numexpr#5\expandafter!\the\numexpr#6\expandafter!\the\numexpr
    #7\expandafter!\the\numexpr#8\expandafter!\the\numexpr#9\expandafter!\the\numexpr
    \expandafter\XINT_sepbyviii_andcount_b\the\numexpr #1+\xint_c_viii\xint:%
}%
\def\XINT_sepbyviii_andcount_end #1\XINT_sepbyviii_andcount_b\the\numexpr
    #2+\xint_c_viii\xint:#3#4\W {\expandafter\xint:\the\numexpr #2+#3\xint:}%
```

21.21. \XINT_sepandrev

This is used typically as

```
   \romannumeral0\XINT_sepandrev <8Ndigits>%
                 \XINT_rsepbyviii_end_A 2345678%
                 \XINT_rsepbyviii_end_B 2345678\relax UV%
                 \R\xint:\R\xint:\R\xint:\R\xint:\R\xint:\R\xint:\R\xint:\R\xint:\W
```

and will produce

```
   1<8digits>!1<8digits>!1<8digits>!...
```

where the blocks have been globally reversed. The UV here are only place holders (must be \numexpr r1 stoppers) to share same syntax as \XINT_sepandrev_andcount, they are gobbled (#2 in \XINT_sepandrev_done).

```
\def\XINT_sepandrev
{%
    \expandafter\XINT_sepandrev_a\the\numexpr 1\XINT_rsepbyviii
}%
\def\XINT_sepandrev_a {\XINT_sepandrev_b {}}%
\def\XINT_sepandrev_b #1#2\xint:#3\xint:#4\xint:#5\xint:#6\xint:#7\xint:#8\xint:#9\xint:%
{%
    \xint_gob_til_R #9\XINT_sepandrev_end\R
    \XINT_sepandrev_b {#9!#8!#7!#6!#5!#4!#3!#2!#1}%
}%
\def\XINT_sepandrev_end\R\XINT_sepandrev_b #1#2\W {\XINT_sepandrev_done #1}%
\def\XINT_sepandrev_done #11#2!{ }%
```

Core arithmetic

All three raw ii operations use the native calxcore implementations. Their callers continue to supply and receive raw decimal integers; representation conversion is confined to these interface macros.

21.22. \xintiiAdd

Input: \xintiiAdd{raw}{raw}; output: raw integer. Convert both operands to native, apply native addition, convert back.

```
\def\xintiiAdd {\romannumeral0\xintiiadd }%
\def\xintiiadd #1#2{\CALX_ntoraw{\CALX_nadd{\CALX_nfromraw{#1}}{\CALX_nfromraw{#2}}}}%
```

21.23. \xintiiCmp

Moved to calxcore at 2.0a1 for STUPID reasons.

Prior to 1.4m the «strcmp» primitive was not used by xintcore. Here is the old implementation:

```
\def\XINT_cmp_plusplus  #1#2#3\xint:
{%
    \expandafter\XINT_cmp_pp
    \the\numexpr\expandafter\XINT_sepbyviii_andcount
    \romannumeral0\XINT_zeroes_forviii #2#3\R\R\R\R\R\R\R{10}0000001\W
    #2#3\XINT_sepbyviii_end 2345678\relax
        \xint_c_vii!\xint_c_vi!\xint_c_v!\xint_c_iv!%
        \xint_c_iii!\xint_c_ii!\xint_c_i!\xint_c_\W
    #1%
}%
\def\XINT_cmp_pp #1\xint:#2\xint:#3\xint:
{%
    \expandafter\XINT_cmp_checklengths
```

```
        \the\numexpr #2\expandafter\xint:%
        \the\numexpr\expandafter\XINT_sepbyviii_andcount
        \romannumeral0\XINT_zeroes_forviii #3\R\R\R\R\R\R\R{10}0000001\W
        #3\XINT_sepbyviii_end 2345678\relax
            \xint_c_vii!\xint_c_vi!\xint_c_v!\xint_c_iv!%
            \xint_c_iii!\xint_c_ii!\xint_c_i!\xint_c_\W
        #1;!1;!1;!1;!\W
    }%
    \def\XINT_cmp_checklengths #1\xint:#2\xint:#3\xint:
    {%
        \ifnum #1=#3
           \expandafter\xint_firstoftwo
        \else
           \expandafter\xint_secondoftwo
        \fi
        \XINT_cmp_a {\XINT_cmp_distinctlengths {#1}{#3}}#2;!1;!1;!1;!\W
    }%
    \def\XINT_cmp_distinctlengths #1#2#3\W #4\W
    {%
        \ifnum #1>#2
            \expandafter\xint_firstoftwo
        \else
            \expandafter\xint_secondoftwo
        \fi
        { -1}{ 1}%
    }%
    \def\XINT_cmp_a 1#1!1#2!1#3!1#4!#5\W 1#6!1#7!1#8!1#9!%
    {%
        \xint_gob_til_sc #1\XINT_cmp_equal ;%
        \ifnum #1>#6 \XINT_cmp_gt\fi
        \ifnum #1<#6 \XINT_cmp_lt\fi
        \xint_gob_til_sc #2\XINT_cmp_equal ;%
        \ifnum #2>#7 \XINT_cmp_gt\fi
        \ifnum #2<#7 \XINT_cmp_lt\fi
        \xint_gob_til_sc #3\XINT_cmp_equal ;%
        \ifnum #3>#8 \XINT_cmp_gt\fi
        \ifnum #3<#8 \XINT_cmp_lt\fi
        \xint_gob_til_sc #4\XINT_cmp_equal ;%
        \ifnum #4>#9 \XINT_cmp_gt\fi
        \ifnum #4<#9 \XINT_cmp_lt\fi
        \XINT_cmp_a #5\W
    }%
    \def\XINT_cmp_lt#1{\def\XINT_cmp_lt\fi ##1\W ##2\W {\fi#1-1}}\XINT_cmp_lt{ }%
    \def\XINT_cmp_gt#1{\def\XINT_cmp_gt\fi ##1\W ##2\W {\fi#11}}\XINT_cmp_gt{ }%
    \def\XINT_cmp_equal #1\W #2\W { 0}%
```

21.24. \xintiiSub

Input: \xintiiSub{raw}{raw}; output: raw integer. Convert both operands to native, apply native subtraction, convert back.

```
\def\xintiiSub {\romannumeral0\xintiisub }%
\def\xintiisub #1#2{\CALX_ntoraw{\CALX_nsub{\CALX_nfromraw{#1}}{\CALX_nfromraw{#2}}}}%
```

21.25. \xintiiMul

Input: \xintiiMul{raw}{raw}; output: raw integer. Convert both operands to native, apply native multiplication, convert back.

```
\def\xintiiMul {\romannumeral0\xintiimul }%
\def\xintiimul #1#2{\CALX_ntoraw{\CALX_nmul{\CALX_nfromraw{#1}}{\CALX_nfromraw{#2}}}}%
```

21.26. Reversed eight-digit block output

Converts a block list, least significant first, terminated by 1;! as produced by the small multiplication kernels, to an untagged decimal integer (used by xintfrac).

```
\def\XINT_mul_out
   {\expandafter\XINT_cuz_small\romannumeral0\XINT_unrevbyviii {}}%
```

21.27. \xintiiDivision

The public ii entry points accept unexpanded, f-expandable material yielding raw ii integers, and return raw integers. Each operation delegates to the corresponding native calxcore division operation. Internal division callers use \XINT_ndivrawpair with divisor first and dividend second; it returns raw {quotient}{remainder}.

```
\def\XINT_ndivrawpair #1#2{%
    \expandafter\XINT_ndivrawpair_a\romannumeral0\CALX_ndivision
        {\CALX_nfromraw{#2}}{\CALX_nfromraw{#1}}}%
\def\XINT_ndivrawpair_a #1#2{%
    \expandafter\XINT_ndivrawpair_b
    \romannumeral0\CALX_ntoraw{#2}\xint:{#1}}%
\def\XINT_ndivrawpair_b #1\xint:#2{%
    \expandafter\XINT_ndivrawpair_c
    \romannumeral0\CALX_ntoraw{#2}\xint:{#1}}%
\def\XINT_ndivrawpair_c #1\xint:#2{{#1}{#2}}%
\def\xintiiDivision   {\romannumeral0\xintiidivision }%
\def\xintiidivision #1#2{\XINT_ndivrawpair{#2}{#1}}%
\def\xintiiQuo {\romannumeral0\xintiiquo }%
\def\xintiiquo #1#2{\CALX_ntoraw{\CALX_nquo{\CALX_nfromraw{#1}}{\CALX_nfromraw{#2}}}}%
\def\xintiiRem {\romannumeral0\xintiirem }%
\def\xintiirem #1#2{\CALX_ntoraw{\CALX_nrem{\CALX_nfromraw{#1}}{\CALX_nfromraw{#2}}}}%
\def\xintiiDivRound   {\romannumeral0\xintiidivround }%
\def\xintiidivround #1#2{\CALX_ntoraw{\CALX_ndivround{\CALX_nfromraw{#1}}{\CALX_nfromraw{#2}}}}%
\def\xintiiDivTrunc   {\romannumeral0\xintiidivtrunc }%
\def\xintiidivtrunc #1#2{\CALX_ntoraw{\CALX_ndivtrunc{\CALX_nfromraw{#1}}{\CALX_nfromraw{#2}}}}%
\def\xintiiModTrunc {\romannumeral0\xintiimodtrunc }%
\def\xintiimodtrunc #1#2{\CALX_ntoraw{\CALX_nmodtrunc{\CALX_nfromraw{#1}}{\CALX_nfromraw{#2}}}}%
\def\xintiiDivMod   {\romannumeral0\xintiidivmod }%
\def\xintiidivmod #1#2{%
    \expandafter\XINT_ndivrawmod_a\romannumeral0\CALX_ndivmod
      {\CALX_nfromraw{#1}}{\CALX_nfromraw{#2}}}%
\def\XINT_ndivrawmod_a #1#2{%
    \expandafter\XINT_ndivrawpair_b
    \romannumeral0\CALX_ntoraw{#2}\xint:{#1}}%
\def\xintiiDivFloor {\romannumeral0\xintiidivfloor}%
\def\xintiidivfloor #1#2{\CALX_ntoraw{\CALX_ndivfloor{\CALX_nfromraw{#1}}{\CALX_nfromraw{#2}}}}%
\def\xintiiMod {\romannumeral0\xintiimod}%
\def\xintiimod #1#2{\CALX_ntoraw{\CALX_nmod{\CALX_nfromraw{#1}}{\CALX_nfromraw{#2}}}}%
```

21.28. \xintiiSqr

1.2l: \xintiiSqr made robust against non terminated input.

```
\def\xintiiSqr {\romannumeral0\xintiisqr }%
\def\xintiisqr #1{\CALX_ntoraw{\XINT_nsquare{\CALX_nfromraw{#1}}}}%
\def\XINT_nsquare #1{\expandafter\XINT_nsquare_a\romannumeral`&&@#1\xint:}%
\def\XINT_nsquare_a #1\xint:{\CALX_nmul{#1}{#1}}%
```

21.29. \xintiiPow

1.2f Modifies the initial steps: 1) in order to be able to let more easily \xintiPow use \xintNum on the exponent once xintfrac.sty is loaded; 2) also because I noticed it was not very well coded. And it did only a \numexpr on the exponent, contradicting the documentation related to the "i" convention in names.

1.2l: \xintiiPow made robust against non terminated input.

The macro makes no a priori test on whether computation has a chance to complete succesfully, as this depends on TeX memory parameters. But roughly, the size of the output should be less than the maximal size for addition, i.e. with TeXLive 2025 settings, have less than about 26600 decimal digits.

```
\def\xintiiPow {\romannumeral0\xintiipow }%
\def\xintiipow #1#2%
{%
    \expandafter\xint_pow\the\numexpr #2\expandafter
    .\romannumeral`&&@#1\xint:
}%
\def\xint_pow #1.#2%#3\xint:
{%
    \xint_UDzerominusfork
      #2-\XINT_pow_AisZero
      0#2\XINT_pow_Aneg
      0-{\XINT_pow_Apos #2}%
    \krof {#1}%
}%
\def\XINT_pow_AisZero #1#2\xint:
{%
     \ifcase\XINT_cntSgn #1\xint:
         \xint_afterfi { 1}%
     \or
         \xint_afterfi { 0}%
     \else
         \xint_afterfi
        {\XINT_legacysignalcondition{DivisionByZero}{0 raised to power #1.}{}{ 0}}%
     \fi
}%
\def\XINT_pow_Aneg #1%
{%
   \ifodd #1
       \expandafter\XINT_opp\romannumeral0%
   \fi
   \XINT_pow_Apos {}{#1}%
}%
\def\XINT_pow_Apos #1#2{\XINT_pow_Apos_a {#2}#1}%
\def\XINT_pow_Apos_a #1#2#3%
```

```
{%
    \xint_gob_til_xint: #3\XINT_pow_Apos_short\xint:
    \XINT_pow_AatleastTwo {#1}#2#3%
}%
\def\XINT_pow_Apos_short\xint:\XINT_pow_AatleastTwo #1#2\xint:
{%
    \ifcase #2
         \xintError:thiscannothappen
    \or  \expandafter\XINT_pow_AisOne
    \else\expandafter\XINT_pow_AatleastTwo
    \fi {#1}#2\xint:
}%
\def\XINT_pow_AisOne #1\xint:{ 1}%
\def\XINT_pow_AatleastTwo #1%
{%
    \ifcase\XINT_cntSgn #1\xint:
        \expandafter\XINT_pow_BisZero
    \or
        \expandafter\XINT_pow_I_in
    \else
        \expandafter\XINT_pow_BisNegative
    \fi
    {#1}%
}%
\def\XINT_pow_BisNegative #1\xint:{\XINT_legacysignalcondition{Underflow}%
    {Inverse power is not an integer.}{}{ 0}}%
\def\XINT_pow_BisZero #1\xint:{ 1}%
```

B = #1 > 0, A = #2 > 1. Earlier code checked if size of B did not exceed a given limit (for example 131000).

```
\def\XINT_pow_I_in #1#2\xint:%
   {\CALX_ntoraw{\CALX_npow{#1}{\CALX_nfromraw{#2}}}}%
```

21.30. \xintiiFac

\xintiiFac{k}: k a \numexpr expression; k! as a raw integer, computed by calxcore's \CALX_nfac. Negative k or k > 10^4: InvalidOperation, 0.

```
\def\xintiiFac {\romannumeral0\xintiifac }%
\def\xintiifac #1{\expandafter\XINT_fac_fork\the\numexpr#1.}%
\def\XINT_fac_fork #1#2.%
{%
    \xint_UDzerominusfork
     #1-\XINT_fac_zero
     0#1\XINT_fac_neg
      0-\XINT_fac_checksize
    \krof #1#2.%
}%
\def\XINT_fac_zero #1.{ 1}%
\def\XINT_fac_neg  #1.{\XINT_legacysignalcondition{InvalidOperation}{Factorial of
    negative argument: #1.}{}{ 0}}%
\def\XINT_fac_checksize #1.%
{%
    \ifnum #1>\xint_c_e_iv \xint_dothis{\XINT_legacysignalcondition
```

```
      {InvalidOperation}{Factorial argument is too large: #1 > 10^4.}%
      {}{ 0}}\fi
    \xint_orthat{\CALX_ntoraw{\CALX_nfac{#1}}}%
}%
```

21.31. \XINT_useiimessage

1.2o

```
\def\XINT_useiimessage #1% used in LaTeX only
{%
    \XINT_ifFlagRaised {#1}%
    {\@backslashchar#1
     (load xintfrac or use \@backslashchar xintii\xint_gobble_iv#1!)\MessageBreak}%
    {}%
}%
\XINTrestorecatcodesendinput%
```

22. Package xint implementation

Contents

.1 Introduction 330
.2 Catcodes, ε-TEX and reload detection . . 330
.3 Package identification 331
.4 \xintLen, \xintiLen 331
.5 \xintiiLogTen 332
.6 \xintReverseDigits 332
.7 \xintiiE 333
.8 \xintDecSplit 333
.9 \xintDecSplitL 334
.10 \xintDecSplitR 335
.11 \xintDSHr 335
.12 \xintDSH 335
.13 \xintDSx 336
.14 \xintiiEq 337
.15 \xintiiNotEq 337
.16 \xintiiGeq 338
.17 \xintiiGt 338
.18 \xintiiLt 338
.19 \xintiiGtorEq 338
.20 \xintiiLtorEq 339
.21 \xintiiIsZero 339
.22 \xintiiIsNotZero 339
.23 \xintiiIsOne 339
.24 \xintiiOdd 339
.25 \xintiiEven 340
.26 \xintiiMON 340
.27 \xintiiMMON 340
.28 \xintSgnFork 340
.29 \xintiiifSgn 341
.30 \xintiiifCmp 341
.31 \xintiiifEq 341
.32 \xintiiifGt 341
.33 \xintiiifLt 342
.34 \xintiiifZero 342
.35 \xintiiifNotZero 342
.36 \xintiiifOne 342
.37 \xintiiifOdd 342
.38 \xintifTrueAelseB, \xintifFalseAelseB 343
.39 \xintIsTrue, \xintIsFalse 343
.40 \xintNOT 343
.41 \xintAND, \xintOR, \xintXOR 343
.42 \xintANDof 343
.43 \xintORof 344
.44 \xintXORof 344
.45 \xintiiMax 345
.46 \xintiiMin 346
.47 \xintiiMaxof 347
.48 \xintiiMinof 347
.49 \xintiiSum 347
.50 \xintiiPrd 348
.51 \xintiiSquareRoot 348
.52 \xintiiSqrt, \xintiiSqrtR 350
.53 \xintiiBinomial 350
.54 \xintiiPFactorial 351
.55 \xintiiGCD 351
.56 \xintiiGCDof 351
.57 \xintiiLCM 352
.58 \xintiiLCMof 352

22.1. Introduction

With release 1.1 the core arithmetic routines \xintiiAdd, \xintiiSub, \xintiiMul, \xintiiQuo, \xintiiPow were separated to be the main component of the then new xintcore.

1.3b adds randomness related macros.

22.2. Catcodes, ε-TEX and reload detection

```
\begingroup\catcode61\catcode48\catcode32=10\relax%
  \catcode13=5    % ^^M
  \endlinechar=13 %
  \catcode123=1   % {
  \catcode125=2   % }
  \catcode64=11   % @
  \catcode44=12   % ,
  \catcode46=12   % .
  \catcode58=12   % :
  \catcode94=7    % ^
  \catcode35=6    % #
```

```
  \def\odef#1{\expandafter\def\expandafter#1\expandafter}%
  \def\empty{}\def\space{ }\newlinechar10
  \def\z{\endgroup}%
  \def\odef#1{\expandafter\def\expandafter#1\expandafter}%
  \expandafter\let\expandafter\x\csname ver@xint.sty\endcsname
  \expandafter\let\expandafter\c\csname ver@calxint.sty\endcsname
  \expandafter\let\expandafter\w\csname ver@xintcore.sty\endcsname
  \expandafter\ifx\csname numexpr\endcsname\relax
    \expandafter\ifx\csname PackageWarningNoLine\endcsname\relax
      \immediate\write128{^^JPackage xint Warning:^^J%
                          \space\space\space\space
                          \numexpr not available, aborting input.^^J}%
    \else
      \PackageWarningNoLine{xint}{\numexpr not available, aborting input}%
    \fi
    \def\z{\endgroup\endinput}%
  \else
    \ifx\x\relax   % plain-TeX, first loading of xint.sty
      \ifx\w\relax % but xintcore.sty not yet loaded.
         \odef\z{\z\input xintcore.sty\relax}%
      \fi
      \ifx\c\relax % but calxint.sty not yet loaded.
         \odef\z{\z\input calxint.sty\relax}%
      \fi
    \else
      \ifx\x\empty % LaTeX, first loading,
      % variable is initialized, but \ProvidesPackage not yet seen
          \ifx\w\relax % xintcore.sty not yet loaded.
            \odef\z{\z\RequirePackage{xintcore}}%
          \fi
          \ifx\c\relax % calxint.sty not yet loaded.
            \odef\z{\z\RequirePackage{calxint}}%
          \fi
      \else
        \def\z{\endgroup\endinput}% xint already loaded.
      \fi
    \fi
  \fi
\z%
\XINTsetupcatcodes% defined in xintkernel.sty (loaded by xintcore.sty)
```

22.3. Package identification

```
\XINT_providespackage
\ProvidesPackage{xint}%
  [2026/11/26 v2.0a2 Operations on big integers (JFB)]%
```

22.4. \xintLen, \xintiLen

\xintLen gets extended to fractions by xintfrac.sty: A/B is given length len(A)+len(B)-1 (somewhat arbitrary). It applies \xintNum to its argument. A minus sign is accepted and ignored.

For parallelism with \xintiNum/\xintNum, 1.2o defines \xintiLen.

\xintLen gets redefined by xintfrac.

```
\def\xintiLen {\romannumeral0\xintilen }%
\def\xintilen #1{\def\xintilen ##1%
{%
    \expandafter#1\the\numexpr
    \expandafter\XINT_len_fork\romannumeral0\xintinum{##1}%
      \xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:
      \xint_c_viii\xint_c_vii\xint_c_vi\xint_c_v
      \xint_c_iv\xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye\relax
}}\xintilen{ }%
\def\xintLen {\romannumeral0\xintlen }%
\let\xintlen\xintilen
\def\XINT_len_fork #1%
{%
    \expandafter\XINT_length_loop\xint_UDsignfork#1{}-#1\krof
}%
```

22.5. \xintiiLogTen

1.3e. Support for ilog10() function in \xintiiexpr. See \XINTiLogTen in xintfrac.sty which also currently uses -"7FFF8000 as value if input is zero.

```
\def\xintiiLogTen {\the\numexpr\xintiilogten }%
\def\xintiilogten #1%
{%
    \expandafter\XINT_iilogten\romannumeral`&&@#1%
      \xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:
      \xint_c_viii\xint_c_vii\xint_c_vi\xint_c_v
      \xint_c_iv\xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye
    \relax
}%
\def\XINT_iilogten #1{\if#10-"7FFF8000\fi -1+%
                      \expandafter\XINT_length_loop\xint_UDsignfork#1{}-#1\krof}%
```

22.6. \xintReverseDigits

1.2.

This puts digits in reverse order, not suppressing leading zeros after reverse. Despite lacking the "ii" in its name, it does not apply \xintNum to its argument (contrarily to \xintLen, this is not very coherent).

1.2l variant is robust against non terminated \the\numexpr input.

This macro is currently not used elsewhere in xint code.

```
\def\xintReverseDigits {\romannumeral0\xintreversedigits }%
\def\xintreversedigits #1%
{%
    \expandafter\XINT_revdigits\romannumeral`&&@#1%
     {\XINT_microrevsep_end\W}\XINT_microrevsep_end
      \XINT_microrevsep_end\XINT_microrevsep_end
      \XINT_microrevsep_end\XINT_microrevsep_end
      \XINT_microrevsep_end\XINT_microrevsep_end\XINT_microrevsep_end\Z
    1\Z!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W
}%
\def\XINT_revdigits #1%
{%
```

```
    \xint_UDsignfork
      #1{\expandafter-\romannumeral0\XINT_revdigits_a}%
       -{\XINT_revdigits_a #1}%
    \krof
}%
\def\XINT_revdigits_a
{%
    \expandafter\XINT_revdigits_b\expandafter{\expandafter}%
    \the\numexpr\XINT_microrevsep
}%
\def\XINT_microrevsep #1#2#3#4#5#6#7#8#9%
{%
    1#9#8#7#6#5#4#3#2#1\expandafter!\the\numexpr\XINT_microrevsep
}%
\def\XINT_microrevsep_end #1\W #2\expandafter #3\Z{\relax#2!}%
\def\XINT_revdigits_b #11#2!1#3!1#4!1#5!1#6!1#7!1#8!1#9!%
{%
    \xint_gob_til_R #9\XINT_revdigits_end\R
                    \XINT_revdigits_b {#9#8#7#6#5#4#3#2#1}%
}%
\def\XINT_revdigits_end#1{%
\def\XINT_revdigits_end\R\XINT_revdigits_b ##1##2\W
   {\expandafter#1\xint_gob_til_Z ##1}%
}\XINT_revdigits_end{ }%
\let\xintRev\xintReverseDigits
```

22.7. \xintiiE

Originally was used in \xintiiexpr. Transferred from xintfrac for 1.1. Code rewritten for 1.2i. \xintiiE{x}{e} extends x with e zeroes if e is positive and simply outputs x if e is zero or negative. Attention, le comportement pour e < 0 ne doit pas être modifié car \xintMod et autres macros en dépendent.

```
\def\xintiiE {\romannumeral0\xintiie }%
\def\xintiie #1#2%
   {\expandafter\XINT_iie_fork\the\numexpr #2\expandafter.\romannumeral`&&@#1;}%
\def\XINT_iie_fork #1%
{%
    \xint_UDsignfork
      #1\XINT_iie_neg
       -\XINT_iie_a
    \krof #1%
}%
```

le #2 a le bon pattern terminé par ; #1=0 est OK pour \XINT_rep.

```
\def\XINT_iie_a #1.%
 {\expandafter\XINT_dsx_append\romannumeral\XINT_rep #1\endcsname 0.}%
\def\XINT_iie_neg #1.#2;{ #2}%
```

22.8. \xintDecSplit

DECIMAL SPLIT
  The macro \xintDecSplit {x}{A} cuts A which is composed of digits (leading zeroes ok, but no sign) (*) into two (each possibly empty) pieces L and R. The concatenation LR always reproduces A.

The position of the cut is specified by the first argument x. If x is zero or positive the cut location is x slots to the left of the right end of the number. If x becomes equal to or larger than the length of the number then L becomes empty. If x is negative the location of the cut is |x| slots to the right of the left end of the number.

(*) versions earlier than 1.2i first replaced A with its absolute value. This is not the case anymore. This macro should NOT be used for A with a leading sign (+ or -).

Entirely rewritten for 1.2i (2016/12/11).

Attention: \xintDecSplit not robust against non terminated second argument.

```
\def\xintDecSplit {\romannumeral0\xintdecsplit }%
\def\xintdecsplit #1#2%
{%
    \expandafter\XINT_split_finish
    \romannumeral0\expandafter\XINT_split_xfork
    \the\numexpr #1\expandafter.\romannumeral`&&@#2%
    \xint_bye2345678\xint_bye..%
}%
\def\XINT_split_finish  #1.#2.{{#1}{#2}}%
\def\XINT_split_xfork #1%
{%
    \xint_UDzerominusfork
      #1-\XINT_split_zerosplit
      0#1\XINT_split_fromleft
       0-{\XINT_split_fromright #1}%
    \krof
}%
\def\XINT_split_zerosplit .#1\xint_bye#2\xint_bye..{ #1..}%
```

\XINT_split_fromleft and its subordinates (\XINT_split_fromleft_a, _b, _end_a, _clean, _end0 to _end7, _end_b) moved verbatim to calxcore.sty at 2.0a1: the float printer core in calxfp.sty uses them.

```
\def\XINT_split_fromright #1.#2\xint_bye
{%
    \expandafter\XINT_split_fromright_a
    \the\numexpr#1-\numexpr\XINT_length_loop
    #2\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:
      \xint_c_viii\xint_c_vii\xint_c_vi\xint_c_v
      \xint_c_iv\xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye
    .#2\xint_bye
}%
\def\XINT_split_fromright_a #1%
{%
    \xint_UDsignfork
      #1\XINT_split_fromleft
       -\XINT_split_fromright_Lempty
    \krof
}%
\def\XINT_split_fromright_Lempty #1.#2\xint_bye#3..{.#2.}%
```

22.9. \xintDecSplitL

```
\def\xintDecSplitL {\romannumeral0\xintdecsplitl }%
\def\xintdecsplitl #1#2%
{%
```

```
    \expandafter\XINT_splitl_finish
    \romannumeral0\expandafter\XINT_split_xfork
    \the\numexpr #1\expandafter.\romannumeral`&&@#2%
    \xint_bye2345678\xint_bye..%
}%
\def\XINT_splitl_finish #1.#2.{ #1}%
```

22.10. \xintDecSplitR

```
\def\xintDecSplitR {\romannumeral0\xintdecsplitr }%
\def\xintdecsplitr #1#2%
{%
    \expandafter\XINT_splitr_finish
    \romannumeral0\expandafter\XINT_split_xfork
    \the\numexpr #1\expandafter.\romannumeral`&&@#2%
    \xint_bye2345678\xint_bye..%
}%
\def\XINT_splitr_finish #1.#2.{ #2}%
```

22.11. \xintDSHr

```
DECIMAL SHIFTS \xintDSH {x}{A}
si x <= 0, fait A -> A.10^(|x|).  si x > 0, et A >=0, fait A -> quo(A,10^(x))
si x > 0, et A < 0, fait A -> -quo(-A,10^(x))
(donc pour x > 0 c'est comme DSR itéré x fois)
\xintDSHr donne le `reste' (si x<=0 donne zéro).
  Badly named macros.
  Rewritten for 1.2i, this was old code and \xintDSx has changed interface.
```

```
\def\xintDSHr {\romannumeral0\xintdshr }%
\def\xintdshr #1#2%
{%
    \expandafter\XINT_dshr_fork\the\numexpr#1\expandafter.\romannumeral`&&@#2;%
}%
\def\XINT_dshr_fork #1%
{%
    \xint_UDzerominusfork
      0#1\XINT_dshr_xzeroorneg
      #1-\XINT_dshr_xzeroorneg
       0-\XINT_dshr_xpositive
    \krof #1%
}%
\def\XINT_dshr_xzeroorneg #1;{ 0}%
\def\XINT_dshr_xpositive
{%
    \expandafter\xint_stop_atsecondoftwo\romannumeral0\XINT_dsx_xisPos
}%
```

22.12. \xintDSH

```
\def\xintDSH {\romannumeral0\xintdsh }%
\def\xintdsh #1#2%
{%
    \expandafter\XINT_dsh_fork\the\numexpr#1\expandafter.\romannumeral`&&@#2;%
```

```
}%
\def\XINT_dsh_fork #1%
{%
    \xint_UDzerominusfork
      #1-\XINT_dsh_xiszero
      0#1\XINT_dsx_xisNeg_checkA
       0-{\XINT_dsh_xisPos #1}%
    \krof
}%
\def\XINT_dsh_xiszero #1.#2;{ #2}%
\def\XINT_dsh_xisPos
{%
    \expandafter\xint_stop_atfirstoftwo\romannumeral0\XINT_dsx_xisPos
}%
```

22.13. \xintDSx

```
--> Attention le cas x=0 est traité dans la même catégorie que x > 0 <--
   si x < 0, fait A -> A.10^(|x|)
   si x >=  0, et A >=0, fait A -> {quo(A,10^(x))}{rem(A,10^(x))}
   si x >=  0, et A < 0, d'abord on calcule {quo(-A,10^(x))}{rem(-A,10^(x))}
      puis, si le premier n'est pas nul on lui donne le signe -
            si le premier est nul on donne le signe - au second.
On peut donc toujours reconstituer l'original A par 10^x Q \pm R où il faut prendre le signe plus
si Q est positif ou nul et le signe moins si Q est strictement négatif.
  Rewritten for 1.2i, this was old code.
```

```
\def\xintDSx {\romannumeral0\xintdsx }%
\def\xintdsx #1#2%
{%
    \expandafter\XINT_dsx_fork\the\numexpr#1\expandafter.\romannumeral`&&@#2;%
}%
\def\XINT_dsx_fork #1%
{%
    \xint_UDzerominusfork
      #1-\XINT_dsx_xisZero
      0#1\XINT_dsx_xisNeg_checkA
       0-{\XINT_dsx_xisPos #1}%
    \krof
}%
\def\XINT_dsx_xisZero #1.#2;{{#2}{0}}%
\def\XINT_dsx_xisNeg_checkA #1.#2%
{%
    \xint_gob_til_zero #2\XINT_dsx_xisNeg_Azero 0%
    \expandafter\XINT_dsx_append\romannumeral\XINT_rep #1\endcsname 0.#2%
}%
\def\XINT_dsx_xisNeg_Azero #1;{ 0}%
\def\XINT_dsx_addzeros #1%
   {\expandafter\XINT_dsx_append\romannumeral\XINT_rep#1\endcsname0.}%
\def\XINT_dsx_addzerosnofuss #1%
   {\expandafter\XINT_dsx_append\romannumeral\xintreplicate{#1}0.}%
\def\XINT_dsx_append #1.#2;{ #2#1}%
\def\XINT_dsx_xisPos #1.#2%
```

```
{%
    \xint_UDzerominusfork
      #2-\XINT_dsx_AisZero
      0#2\XINT_dsx_AisNeg
       0-\XINT_dsx_AisPos
    \krof #1.#2%
}%
\def\XINT_dsx_AisZero #1;{{0}{0}}%
\def\XINT_dsx_AisNeg #1.-#2;%
{%
    \expandafter\XINT_dsx_AisNeg_checkiffirstempty
    \romannumeral0\XINT_split_xfork #1.#2\xint_bye2345678\xint_bye..%
}%
\def\XINT_dsx_AisNeg_checkiffirstempty #1%
{%
    \xint_gob_til_dot #1\XINT_dsx_AisNeg_finish_zero.%
    \XINT_dsx_AisNeg_finish_notzero #1%
}%
\def\XINT_dsx_AisNeg_finish_zero.\XINT_dsx_AisNeg_finish_notzero.#1.%
{%
    \expandafter\XINT_dsx_end
    \expandafter {\romannumeral0\XINT_num {-#1}}{0}%
}%
\def\XINT_dsx_AisNeg_finish_notzero #1.#2.%
{%
    \expandafter\XINT_dsx_end
    \expandafter {\romannumeral0\XINT_num {#2}}{-#1}%
}%
\def\XINT_dsx_AisPos #1.#2;%
{%
    \expandafter\XINT_dsx_AisPos_finish
    \romannumeral0\XINT_split_xfork #1.#2\xint_bye2345678\xint_bye..%
}%
\def\XINT_dsx_AisPos_finish #1.#2.%
{%
    \expandafter\XINT_dsx_end
    \expandafter {\romannumeral0\XINT_num {#2}}%
                 {\romannumeral0\XINT_num {#1}}%
}%
\def\XINT_dsx_end #1#2{\expandafter{#2}{#1}}%
```

22.14. \xintiiEq

no \xintiieq.

```
\def\xintiiEq #1#2{\romannumeral0\xintiiifeq{#1}{#2}{1}{0}}%
```

22.15. \xintiiNotEq

Pour xintexpr. Pas de version en lowercase.

```
\def\xintiiNotEq #1#2{\romannumeral0\xintiiifeq {#1}{#2}{0}{1}}%
```

22.16. \xintiiGeq

```
PLUS GRAND OU ÉGAL attention compare les **valeurs absolues**
  1.2l made \xintiiGeq robust against non terminated items.
  1.2l rewrote \xintiiCmp, but forgot to handle \xintiiGeq too.  Done at 1.2m.
  This macro should have been called \xintGEq for example.
```

```
\def\xintiiGeq   {\romannumeral0\xintiigeq }%
\def\xintiigeq #1{\expandafter\XINT_iigeq\romannumeral`&&@#1\xint:}%
\def\XINT_iigeq #1#2\xint:#3%
{%
    \expandafter\XINT_geq_fork\expandafter #1\romannumeral`&&@#3\xint:#2\xint:
}%
\def\XINT_geq #1#2\xint:#3%
{%
    \expandafter\XINT_geq_fork\expandafter #1\romannumeral0\xintnum{#3}\xint:#2\xint:
}%
\def\XINT_geq_fork #1#2%
{%
    \xint_UDzerofork
      #1\XINT_geq_firstiszero
      #2\XINT_geq_secondiszero
       0{}%
    \krof
    \xint_UDsignsfork
          #1#2\XINT_geq_minusminus
           #1-\XINT_geq_minusplus
           #2-\XINT_geq_plusminus
            --\XINT_geq_plusplus
    \krof #1#2%
}%
\def\XINT_geq_firstiszero  #1\krof 0#2#3\xint:#4\xint:
                             {\xint_UDzerofork #2{ 1}0{ 0}\krof }%
\def\XINT_geq_secondiszero #1\krof #20#3\xint:#4\xint:{ 1}%
\def\XINT_geq_plusminus    #1-{\XINT_geq_plusplus #1{}}%
\def\XINT_geq_minusplus    -#1{\XINT_geq_plusplus {}#1}%
\def\XINT_geq_minusminus    --{\XINT_geq_plusplus {}{}}%
\def\XINT_geq_plusplus
   {\expandafter\XINT_geq_finish\romannumeral0\XINT_cmp_plusplus}%
\def\XINT_geq_finish #1{\if-#1\expandafter\XINT_geq_no
                        \else\expandafter\XINT_geq_yes\fi}%
\def\XINT_geq_no 1{ 0}%
\def\XINT_geq_yes { 1}%
```

22.17. \xintiiGt

```
\def\xintiiGt #1#2{\romannumeral0\xintiiifgt{#1}{#2}{1}{0}}%
```

22.18. \xintiiLt

```
\def\xintiiLt #1#2{\romannumeral0\xintiiiflt{#1}{#2}{1}{0}}%
```

22.19. \xintiiGtorEq

```
\def\xintiiGtorEq #1#2{\romannumeral0\xintiiiflt {#1}{#2}{0}{1}}%
```

22.20. \xintiiLtorEq

```
\def\xintiiLtorEq #1#2{\romannumeral0\xintiiifgt {#1}{#2}{0}{1}}%
```

22.21. \xintiiIsZero

1.09a. restyled in 1.09i. 1.1 adds \xintiiIsZero, etc... for optimization in \xintexpr

```
\def\xintiiIsZero {\romannumeral0\xintiiiszero }%
\def\xintiiiszero #1{\if0\xintiiSgn{#1}\xint_afterfi{ 1}\else\xint_afterfi{ 0}\fi}%
```

22.22. \xintiiIsNotZero

1.09a. restyled in 1.09i. 1.1 adds \xintiiIsZero, etc... for optimization in \xintexpr

```
\def\xintiiIsNotZero {\romannumeral0\xintiiisnotzero }%
\def\xintiiisnotzero
          #1{\if0\xintiiSgn{#1}\xint_afterfi{ 0}\else\xint_afterfi{ 1}\fi}%
```

22.23. \xintiiIsOne

Added in 1.03. 1.09a defines \xintIsOne. 1.1a adds \xintiiIsOne.
\XINT_isOne rewritten for 1.2g. Works with expanded strict integers, positive or negative.

```
\def\xintiiIsOne {\romannumeral0\xintiiisone }%
\def\xintiiisone #1{\expandafter\XINT_isone\romannumeral`&&@#1XY}%
\def\XINT_isone #1#2#3Y%
{%
    \unless\if#2X\xint_dothis{ 0}\fi
    \unless\if#11\xint_dothis{ 0}\fi
    \xint_orthat{ 1}%
}%
\def\XINT_isOne #1{\XINT_is_One#1XY}%
\def\XINT_is_One #1#2#3Y%
{%
    \unless\if#2X\xint_dothis0\fi
    \unless\if#11\xint_dothis0\fi
    \xint_orthat1%
}%
```

22.24. \xintiiOdd

\xintOdd is needed for the xintexpr-essions even() and odd() functions (and also by \xintNewExpr).

```
\def\xintiiOdd {\romannumeral0\xintiiodd }%
\def\xintiiodd #1%
{%
    \ifodd\xintLDg{#1} %<- intentional space
        \xint_afterfi{ 1}%
    \else
        \xint_afterfi{ 0}%
    \fi
}%
```

22.25. \xintiiEven

```
\def\xintiiEven {\romannumeral0\xintiieven }%
\def\xintiieven #1%
{%
    \ifodd\xintLDg{#1} %<- intentional space
        \xint_afterfi{ 0}%
    \else
        \xint_afterfi{ 1}%
    \fi
}%
```

22.26. \xintiiMON

MINUS ONE TO THE POWER N

```
\def\xintiiMON {\romannumeral0\xintiimon }%
\def\xintiimon #1%
{%
    \ifodd\xintLDg {#1} %<- intentional space
        \xint_afterfi{ -1}%
    \else
        \xint_afterfi{ 1}%
    \fi
}%
```

22.27. \xintiiMMON

MINUS ONE TO THE POWER N-1

```
\def\xintiiMMON {\romannumeral0\xintiimmon }%
\def\xintiimmon #1%
{%
    \ifodd\xintLDg {#1} %<- intentional space
        \xint_afterfi{ 1}%
    \else
        \xint_afterfi{ -1}%
    \fi
}%
```

22.28. \xintSgnFork

Expandable three-way fork added in 1.07. The argument #1 must expand to non-self-ending -1,0 or 1. 1.09i with _thenstop (now _stop_at...).

```
\def\xintSgnFork {\romannumeral0\xintsgnfork }%
\def\xintsgnfork #1%
{%
    \ifcase #1 \expandafter\xint_stop_atsecondofthree
            \or\expandafter\xint_stop_atthirdofthree
          \else\expandafter\xint_stop_atfirstofthree
    \fi
}%
```

22.29. \xintiiifSgn

Expandable three-way fork added in 1.09a. Branches expandably depending on whether <0, =0, >0. Choice of branch guaranteed in two steps.

1.09i has \xint_firstofthreeafterstop (now \xint_stop_atfirstofthree) etc for faster expansion.

1.1 adds \xintiiifSgn for optimization in xintexpr-essions. Should I move them to xintcore? (for bnumexpr)

```
\def\xintiiifSgn {\romannumeral0\xintiiifsgn }%
\def\xintiiifsgn #1%
{%
    \ifcase \xintiiSgn{#1}
                \expandafter\xint_stop_atsecondofthree
              \or\expandafter\xint_stop_atthirdofthree
            \else\expandafter\xint_stop_atfirstofthree
    \fi
}%
```

22.30. \xintiiifCmp

1.09e \xintifCmp {n}{m}{if n<m}{if n=m}{if n>m}. 1.1a adds ii variant

```
\def\xintiiifCmp {\romannumeral0\xintiiifcmp }%
\def\xintiiifcmp #1#2%
{%
    \ifcase\xintiiCmp {#1}{#2}
                \expandafter\xint_stop_atsecondofthree
              \or\expandafter\xint_stop_atthirdofthree
            \else\expandafter\xint_stop_atfirstofthree
    \fi
}%
```

22.31. \xintiiifEq

1.09a \xintifEq {n}{m}{YES if n=m}{NO if n<>m}. 1.1a adds ii variant

```
\def\xintiiifEq {\romannumeral0\xintiiifeq }%
\def\xintiiifeq #1#2%
{%
    \if0\xintiiCmp{#1}{#2}%
                \expandafter\xint_stop_atfirstoftwo
            \else\expandafter\xint_stop_atsecondoftwo
    \fi
}%
```

22.32. \xintiiifGt

1.09a \xintifGt {n}{m}{YES if n>m}{NO if n<=m}. 1.1a adds ii variant

```
\def\xintiiifGt {\romannumeral0\xintiiifgt }%
\def\xintiiifgt #1#2%
{%
    \if1\xintiiCmp{#1}{#2}%
                \expandafter\xint_stop_atfirstoftwo
            \else\expandafter\xint_stop_atsecondoftwo
```

```
    \fi
}%
```

22.33. \xintiiifLt

Moved to calxint for stupid reasons.

22.34. \xintiiifZero

Expandable two-way fork added in 1.09a. Branches expandably depending on whether the argument is zero (branch A) or not (branch B). 1.09i restyling. By the way it appears (not thoroughly tested, though) that \if tests are faster than \ifnum tests. 1.1 adds ii versions.

```
\def\xintiiifZero {\romannumeral0\xintiiifzero }%
\def\xintiiifzero #1%
{%
    \if0\xintiiSgn{#1}%
       \expandafter\xint_stop_atfirstoftwo
    \else
       \expandafter\xint_stop_atsecondoftwo
    \fi
}%
```

22.35. \xintiiifNotZero

```
\def\xintiiifNotZero {\romannumeral0\xintiiifnotzero }%
\def\xintiiifnotzero #1%
{%
    \if0\xintiiSgn{#1}%
       \expandafter\xint_stop_atsecondoftwo
    \else
       \expandafter\xint_stop_atfirstoftwo
    \fi
}%
```

22.36. \xintiiifOne

added in 1.09i. 1.1a adds \xintiiifOne.

```
\def\xintiiifOne {\romannumeral0\xintiiifone }%
\def\xintiiifone #1%
{%
    \if1\xintiiIsOne{#1}%
       \expandafter\xint_stop_atfirstoftwo
    \else
       \expandafter\xint_stop_atsecondoftwo
    \fi
}%
```

22.37. \xintiiifOdd

1.09e. Restyled in 1.09i. 1.1a adds \xintiiifOdd.

```
\def\xintiiifOdd {\romannumeral0\xintiiifodd }%
\def\xintiiifodd #1%
```

```
{%
    \if\xintiiOdd{#1}1%
       \expandafter\xint_stop_atfirstoftwo
    \else
       \expandafter\xint_stop_atsecondoftwo
    \fi
}%
```

22.38. \xintifTrueAelseB, \xintifFalseAelseB

1.09i, with name changes at 1.2i.

1.2o uses \xintiiifNotZero, see comments to \xintAND etc... This will work fine with arguments being nested xintfrac.sty macros, without the overhead of \xintNum or \xintRaw parsing.

```
\def\xintifTrueAelseB {\romannumeral0\xintiiifnotzero}%
\def\xintifFalseAelseB{\romannumeral0\xintiiifzero}%
```

22.39. \xintIsTrue, \xintIsFalse

1.09c. Suppressed at 1.2o. They seem not to have been documented, fortunately.

```
%\let\xintIsTrue \xintIsNotZero
%\let\xintIsFalse\xintIsZero
```

22.40. \xintNOT

1.09c with name change at 1.2o. Besides, the macro is now defined as ii-type.

```
\def\xintNOT{\romannumeral0\xintiiiszero}%
```

22.41. \xintAND, \xintOR, \xintXOR

Added with 1.09a. But they used \xintSgn, etc... rather than \xintiiSgn. This brings \xintNum overhead which is not really desired, and which is not needed for use by xintexpr.sty. At 1.2o I modify them to use only ii macros. This is enough for sign or zeroness even for xintfrac format, as manipulated inside the \xintexpr. Big hesitation whether there should be however \xintiiAND outputting 1 or 0 versus an \xintAND outputting 1[0] versus 0[0] for example.

```
\def\xintAND {\romannumeral0\xintand }%
\def\xintand #1#2{\if0\xintiiSgn{#1}\expandafter\xint_firstoftwo
                              \else\expandafter\xint_secondoftwo\fi
                  { 0}{\xintiiisnotzero{#2}}}%
\def\xintOR {\romannumeral0\xintor }%
\def\xintor #1#2{\if0\xintiiSgn{#1}\expandafter\xint_firstoftwo
                             \else\expandafter\xint_secondoftwo\fi
                 {\xintiiisnotzero{#2}}{ 1}}%
\def\xintXOR {\romannumeral0\xintxor }%
\def\xintxor #1#2{\if\xintiiIsZero{#1}\xintiiIsZero{#2}%
                     \xint_afterfi{ 0}\else\xint_afterfi{ 1}\fi }%
```

22.42. \xintANDof

New with 1.09a. \xintANDof works also with an empty list. Empty items however are not accepted.

1.2l made \xintANDof robust against non terminated items.

1.2o's \xintifTrueAelseB is now an ii macro, actually.

1.4. This macro as well as ORof and XORof were formally not used by xintexpr, which uses comma separated items, but at 1.4 xintexpr uses braced items. And the macros here got slightly refactored and \XINT_ANDof added for usage by xintexpr and the NewExpr hook. For some random reason I decided to use ^ as delimiter this has to do that other macros in xintfrac in same family (such as \xintGCDof, \xintSum) also use \xint: internally and although not strictly needed having two separate ones clarifies.

```
\def\xintANDof  {\romannumeral0\xintandof }%
\def\xintandof  #1{\expandafter\XINT_andof\romannumeral`&&@#1^}%
\def\XINT_ANDof {\romannumeral0\XINT_andof}%
\def\XINT_andof #1%
{%
    \xint_gob_til_^ #1\XINT_andof_yes ^%
    \xintiiifNotZero{#1}\XINT_andof\XINT_andof_no
}%
\def\XINT_andof_no  #1^{ 0}%
\def\XINT_andof_yes ^#1\XINT_andof_no{ 1}%
```

22.43. \xintORof

New with 1.09a. Works also with an empty list. Empty items however are not accepted.
1.2l made \xintORof robust against non terminated items.
Refactored at 1.4.

```
\def\xintORof  {\romannumeral0\xintorof }%
\def\xintorof  #1{\expandafter\XINT_orof\romannumeral`&&@#1^}%
\def\XINT_ORof {\romannumeral0\XINT_orof}%
\def\XINT_orof #1%
{%
    \xint_gob_til_^ #1\XINT_orof_no ^%
    \xintiiifNotZero{#1}\XINT_orof_yes\XINT_orof
}%
\def\XINT_orof_yes#1^{ 1}%
\def\XINT_orof_no ^#1\XINT_orof{ 0}%
```

22.44. \xintXORof

New with 1.09a. Works with an empty list, too. Empty items however are not accepted. \XINT_xorof_c more efficient in 1.09i.
1.2l made \xintXORof robust against non terminated items.
Refactored at 1.4 to use \numexpr (or an \ifnum). I have not tested if more efficient or not or if one can do better without \the. \XINT_XORof for xintexpr matters.

```
\def\xintXORof  {\romannumeral0\xintxorof }%
\def\xintxorof  #1{\expandafter\XINT_xorof\romannumeral`&&@#1^}%
\def\XINT_XORof {\romannumeral0\XINT_xorof}%
\def\XINT_xorof {\if1\the\numexpr\XINT_xorof_a}%
\def\XINT_xorof_a #1%
{%
    \xint_gob_til_^ #1\XINT_xorof_e ^%
    \xintiiifNotZero{#1}{-}{}\XINT_xorof_a
}%
\def\XINT_xorof_e ^#1\XINT_xorof_a
   {1\relax\xint_afterfi{ 0}\else\xint_afterfi{ 1}\fi}%
```

22.45. \xintiiMax

At 1.2m, a long-standing bug was fixed: \xintiiMax had the overhead of applying \xintNum to its arguments due to use of a sub-macro of \xintGeq code to which this overhead was added at some point.

And on this occasion I reduced even more number of times input is grabbed.

```
\def\xintiiMax {\romannumeral0\xintiimax }%
\def\xintiimax #1%
{%
    \expandafter\xint_iimax \romannumeral`&&@#1\xint:
}%
\def\xint_iimax #1\xint:#2%
{%
    \expandafter\XINT_max_fork\romannumeral`&&@#2\xint:#1\xint:
}%
```

#3#4 vient du *premier*, #1#2 vient du *second*. I have renamed the sub-macros at 1.2m because the terminology was quite counter-intuitive; there was no bug, but still.

```
\def\XINT_max_fork #1#2\xint:#3#4\xint:
{%
    \xint_UDsignsfork
          #1#3\XINT_max_minusminus  % A < 0, B < 0
           #1-\XINT_max_plusminus   % B < 0, A >= 0
           #3-\XINT_max_minusplus   % A < 0, B >= 0
            --{\xint_UDzerosfork
                      #1#3\XINT_max_zerozero % A = B = 0
                       #10\XINT_max_pluszero % B = 0, A > 0
                       #30\XINT_max_zeroplus % A = 0, B > 0
                        00\XINT_max_plusplus % A, B > 0
                      \krof }%
    \krof
    #3#1#2\xint:#4\xint:
      \expandafter\xint_stop_atfirstoftwo
    \else
      \expandafter\xint_stop_atsecondoftwo
    \fi
    {#3#4}{#1#2}%
}%
```

Refactored at 1.2m for avoiding grabbing arguments. Position of inputs shared with iiCmp and iiGeq code.

```
\def\XINT_max_zerozero  #1\fi{\xint_stop_atfirstoftwo }%
\def\XINT_max_zeroplus  #1\fi{\xint_stop_atsecondoftwo }%
\def\XINT_max_pluszero  #1\fi{\xint_stop_atfirstoftwo }%
\def\XINT_max_minusplus #1\fi{\xint_stop_atsecondoftwo }%
\def\XINT_max_plusminus #1\fi{\xint_stop_atfirstoftwo }%
\def\XINT_max_plusplus
{%
    \if1\romannumeral0\XINT_geq_plusplus
}%
```

Premier des testés |A|=-A, second est |B|=-B. On veut le max(A,B), c'est donc A si |A|<|B| (ou |A|=|B|, mais peu importe alors). Donc on peut faire cela avec \unless. Simple.

```
\def\XINT_max_minusminus --%
{%
```

```
    \unless\if1\romannumeral0\XINT_geq_plusplus{}{}%
}%
```

22.46. \xintiiMin

New with 1.09a. I add \xintiiMin in 1.1 and \xintMin is an xintfrac macro.

At 1.2m, a long-standing bug was fixed: \xintiiMin had the overhead of applying \xintNum to its arguments due to use of a sub-macro of \xintGeq code to which this overhead was added at some point.

And on this occasion I reduced even more number of times input is grabbed.

```
\def\xintiiMin {\romannumeral0\xintiimin }%
\def\xintiimin #1%
{%
    \expandafter\xint_iimin \romannumeral`&&@#1\xint:
}%
\def\xint_iimin #1\xint:#2%
{%
    \expandafter\XINT_min_fork\romannumeral`&&@#2\xint:#1\xint:
}%
\def\XINT_min_fork #1#2\xint:#3#4\xint:
{%
    \xint_UDsignsfork
          #1#3\XINT_min_minusminus  % A < 0, B < 0
           #1-\XINT_min_plusminus   % B < 0, A >= 0
           #3-\XINT_min_minusplus   % A < 0, B >= 0
            --{\xint_UDzerosfork
                      #1#3\XINT_min_zerozero % A = B = 0
                       #10\XINT_min_pluszero % B = 0, A > 0
                       #30\XINT_min_zeroplus % A = 0, B > 0
                        00\XINT_min_plusplus % A, B > 0
                      \krof }%
    \krof
    #3#1#2\xint:#4\xint:
      \expandafter\xint_stop_atsecondoftwo
    \else
      \expandafter\xint_stop_atfirstoftwo
    \fi
    {#3#4}{#1#2}%
}%
\def\XINT_min_zerozero  #1\fi{\xint_stop_atfirstoftwo }%
\def\XINT_min_zeroplus  #1\fi{\xint_stop_atfirstoftwo }%
\def\XINT_min_pluszero  #1\fi{\xint_stop_atsecondoftwo }%
\def\XINT_min_minusplus #1\fi{\xint_stop_atfirstoftwo }%
\def\XINT_min_plusminus #1\fi{\xint_stop_atsecondoftwo }%
\def\XINT_min_plusplus
{%
    \if1\romannumeral0\XINT_geq_plusplus
}%
\def\XINT_min_minusminus --%
{%
    \unless\if1\romannumeral0\XINT_geq_plusplus{}{}%
}%
```

22.47. \xintiiMaxof

New with 1.09a. 1.2 has NO MORE \xintMaxof, requires \xintfracname. 1.2a adds \xintiiMaxof, as \xintiiMaxof:csv is not public.
NOT compatible with empty list.
1.2l made \xintiiMaxof robust against non terminated items.
1.4 refactors code to allow empty argument. For usage by \xintiiexpr. Slight deterioration, will come back.

```
\def\xintiiMaxof {\romannumeral0\xintiimaxof }%
\def\xintiimaxof #1{\expandafter\XINT_iimaxof\romannumeral`&&@#1^}%
\def\XINT_iiMaxof{\romannumeral0\XINT_iimaxof}%
\def\XINT_iimaxof#1%
{%
    \xint_gob_til_^ #1\XINT_iimaxof_empty ^%
    \expandafter\XINT_iimaxof_loop\romannumeral`&&@#1\xint:
}%
\def\XINT_iimaxof_empty ^#1\xint:{ 0}%
\def\XINT_iimaxof_loop #1\xint:#2%
{%
    \xint_gob_til_^ #2\XINT_iimaxof_e ^%
    \expandafter\XINT_iimaxof_loop\romannumeral0\xintiimax{#1}{#2}\xint:
}%
\def\XINT_iimaxof_e ^#1\xintiimax #2#3\xint:{ #2}%
```

22.48. \xintiiMinof

1.09a. 1.2a adds \xintiiMinof which was lacking.
1.4 refactoring for \xintiiexpr matters.

```
\def\xintiiMinof {\romannumeral0\xintiiminof }%
\def\xintiiminof #1{\expandafter\XINT_iiminof\romannumeral`&&@#1^}%
\def\XINT_iiMinof{\romannumeral0\XINT_iiminof}%
\def\XINT_iiminof#1%
{%
    \xint_gob_til_^ #1\XINT_iiminof_empty ^%
    \expandafter\XINT_iiminof_loop\romannumeral`&&@#1\xint:
}%
\def\XINT_iiminof_empty ^#1\xint:{ 0}%
\def\XINT_iiminof_loop #1\xint:#2%
{%
    \xint_gob_til_^ #2\XINT_iiminof_e ^%
    \expandafter\XINT_iiminof_loop\romannumeral0\xintiimin{#1}{#2}\xint:
}%
\def\XINT_iiminof_e ^#1\xintiimin #2#3\xint:{ #2}%
```

22.49. \xintiiSum

\xintiiSum {{a}{b}...{z}} Refactored at 1.4 for matters initially related to xintexpr delimiter choice.

```
\def\xintiiSum {\romannumeral0\xintiisum }%
\def\xintiisum #1{\expandafter\XINT_iisum\romannumeral`&&@#1^}%
\def\XINT_iiSum{\romannumeral0\XINT_iisum}%
\def\XINT_iisum #1{\expandafter\XINT_iisum_loop\expandafter{\romannumeral0\CALX_nfromraw{0}}{#1}}%
\def\XINT_iisum_loop #1#2{%
```

```
    \xint_gob_til_^ #2\XINT_iisum_end ^%
    \XINT_iisum_next{#1}{#2}%
}%
\def\XINT_iisum_next #1#2{%
    \expandafter\XINT_iisum_next_a\romannumeral0\CALX_nfromraw{#2}\xint:{#1}%
}%
\def\XINT_iisum_next_a #1\xint:#2{%
    \expandafter\XINT_iisum_loop\expandafter
    {\romannumeral0\CALX_naddready{#2}{#1}}%
}%
\def\XINT_iisum_end ^\XINT_iisum_next #1#2{\CALX_ntoraw{#1}}%
```

22.50. \xintiiPrd

\xintiiPrd {{a}...{z}}

Macros renamed and refactored (slightly more macros here to supposedly bring micro-gain) at 1.4 to match changes in xintfrac of delimiter, in sync with some usage in xintexpr.

Contrarily to the xintfrac version \xintPrd, this one aborts as soon as it hits a zero value.

```
\def\xintiiPrd {\romannumeral0\xintiiprd }%
\def\xintiiprd #1{\expandafter\XINT_iiprd\romannumeral`&&@#1^}%
\def\XINT_iiPrd{\romannumeral0\XINT_iiprd}%
```

The above romannumeral caused f-expansion of the list argument. We f-expand below the first item and each successive items because The product is accumulated in native format; each factor is converted once, and the accumulator is converted back to raw at the end.

```
\def\XINT_iiprd #1{\expandafter\XINT_iiprd_loop\expandafter{\romannumeral0\CALX_nfromraw{1}}{#1}}%
\def\XINT_iiprd_loop #1#2{%
    \xint_gob_til_^ #2\XINT_iiprd_end ^%
    \XINT_iiprd_next{#1}{#2}%
}%
\def\XINT_iiprd_next #1#2{%
    \expandafter\XINT_iiprd_next_a\romannumeral0\CALX_nfromraw{#2}\xint:{#1}%
}%
\def\XINT_iiprd_next_a #1\xint:#2{%
    \expandafter\XINT_iiprd_loop\expandafter
    {\romannumeral0\CALX_nmulready{#2}{#1}}%
}%
\def\XINT_iiprd_end ^\XINT_iiprd_next #1#2{\CALX_ntoraw{#1}}%
```

22.51. \xintiiSquareRoot

First done with 1.08.

1.1 added \xintiiSquareRoot.

1.1a added \xintiiSqrtR.

1.2f (2016/03/01-02-03) has rewritten the implementation, the underlying mathematics remaining about the same. The routine is much faster for inputs having up to 16 digits (because it does it all with \numexpr directly now), and also much faster for very long inputs (because it now fetches only the needed new digits after the first 16 (or 17) ones, via the geometric sequence 16, then 32, then 64, etc...; earlier version did the computations with all remaining digits after a suitable starting point with correct 4 or 5 leading digits). Note however that the fetching of tokens is via intrinsically O(N^2) macros, hence inevitably inputs with thousands of digits start being treated less well.

Actually there is some room for improvements, one could prepare better input X for the upcoming treatment of fetching its digits by 16, then 32, then 64, etc...

Incidently, as \xintiiSqrt uses subtraction and subtraction was broken from 1.2 to 1.2c, then for another reason from 1.2c to 1.2f, it could get wrong in certain (relatively rare) cases. There was also a bug that made it unneedlessly slow for odd number of digits on input.

1.2f also modifies \xintFloatSqrt in xintfrac.sty which now has more code in common with here and benefits from the same speed improvements.

1.2k belatedly corrects the output to {1}{1} and not 11 when input is zero. As braces are used in all other cases they should have been used here too.

Also, 1.2k adds an \xintiSqrtR macro, for coherence as \xintiSqrt is defined (and mentioned in user manual.)

calx (s58): at most 16 digits, calxint's \CALX_sqrt_start (Burnol's start, small and medium paths) as before; more than 16 digits, calxint's native \CALX_nsqrtrem: {S}{R} with S = isqrt(X), R = X - S^2, and the output {S+1}{2S+1-R} (\XINT_sqrt_nb, \XINT_sqrt_nc). xint's big path (\XINT_sqrt_big_*) is deleted. The digit count is that of \xintlength, so a count register (one token) still takes the \numexpr small path.

```
\def\xintiiSquareRoot {\romannumeral0\xintiisquareroot }%
\def\xintiisquareroot #1{\expandafter\XINT_sqrt_checkin\romannumeral`&&@#1\xint:}%
\def\XINT_sqrt_checkin #1%
{%
    \xint_UDzerominusfork
     #1-\XINT_sqrt_iszero
     0#1\XINT_sqrt_isneg
      0-\XINT_sqrt
    \krof #1%
}%
\def\XINT_sqrt_iszero #1\xint:{{1}{1}}%
\def\XINT_sqrt_isneg  #1\xint:
   {\XINT_legacysignalcondition{InvalidOperation}%
                          {Square root of negative: #1.}{}{{0}{0}}}%
\def\XINT_sqrt #1\xint:
{%
    \expandafter\XINT_sqrt_a\romannumeral0\xintlength {#1}.#1.%
}%
\def\XINT_sqrt_a #1.%
{%
    \ifnum #1>\xint_c_xvi \expandafter\XINT_sqrt_n\fi
    \CALX_sqrt_start #1.%
}%
\def\XINT_sqrt_n\CALX_sqrt_start #1.#2.%
{%
    \expandafter\XINT_sqrt_nb\romannumeral0\CALX_nsqrtrem{\CALX_nfromraw{#2}}%
}%
\def\XINT_sqrt_nb #1#2%
{%
    \expandafter\XINT_sqrt_nc
    \romannumeral0\CALX_nadd{#1}{\XINT_sqrt_one}\xint:{#1}{#2}%
}%
\def\XINT_sqrt_nc #1\xint:#2#3%
{%
    \expanded{{\romannumeral0\CALX_ntoraw{#1}}%
              {\romannumeral0\CALX_ntoraw{\CALX_nsub{\CALX_nadd{#1}{#2}}{#3}}}}%
```

```
}%
\expandafter\def\expandafter\XINT_sqrt_one\expandafter{\string N+1.100000001!}%
```

22.52. \xintiiSqrt, \xintiiSqrtR

```
\def\xintiiSqrt  {\romannumeral0\xintiisqrt  }%
\def\xintiisqrt  {\expandafter\XINT_sqrt_post\romannumeral0\xintiisquareroot  }%
\def\XINT_sqrt_post #1#2{\XINT_dec #1\XINT_dec_bye234567890\xint_bye}%
\def\xintiiSqrtR {\romannumeral0\xintiisqrtr }%
\def\xintiisqrtr {\expandafter\XINT_sqrtr_post\romannumeral0\xintiisquareroot }%
```

N = (#1)^2 - #2 avec #1 le plus petit possible et #2>0 (hence #2<2*#1). (#1-.5)^2=#1^2-#1+.25=N+#2-#1+.25. Si 0<#2<#1, <= N-0.75<N, donc rounded->#1 si #2>=#1, (#1-.5)^2>=N+.25>N, donc rounded->#1-1.

```
\def\XINT_sqrtr_post #1#2%
   {\xintiiifLt {#2}{#1}{ #1}{\XINT_dec #1\XINT_dec_bye234567890\xint_bye}}%
```

22.53. \xintiiBinomial

2015/11/28-29 for 1.2f.
2016/11/19 for 1.2h: I truly can't understand why I hard-coded last year an error-message for arguments outside of the range for binomial formula. Naturally there should be no error but a rather a 0 return value for binomial(x,y), if y<0 or x<y !
I really lack some kind of infinity or NaN value.

```
\def\xintiiBinomial {\romannumeral0\xintiibinomial }%
\def\xintiibinomial #1#2%
{%
    \expandafter\XINT_binom_pre\the\numexpr #1\expandafter.\the\numexpr #2.%
}%
\def\XINT_binom_pre #1.#2.%
{%
    \expandafter\XINT_binom_fork \the\numexpr#1-#2.#2.#1.%
}%
```

k.x-k.x. I hesitated to restrict maximal allowed value of x to 10000. Finally I don't. But due to using small multiplication and small division, x must have at most eight digits. If x>=2^31 an arithmetic overflow error will have happened already.

```
\def\XINT_binom_fork #1#2.#3#4.#5#6.%
{%
    \if-#5\xint_dothis{\XINT_legacysignalcondition{InvalidOperation}%
                       {Binomial with negative first argument: #5#6.}{}{ 0}}\fi
    \if-#1\xint_dothis{ 0}\fi
    \if-#3\xint_dothis{ 0}\fi
    \if0#1\xint_dothis{ 1}\fi
    \if0#3\xint_dothis{ 1}\fi
    \ifnum #5#6>\xint_c_e_viii_mone\xint_dothis
       {\XINT_legacysignalcondition{InvalidOperation}%
            {Binomial with too large argument: #5#6 >= 10^8.}{}{ 0}}\fi
    \ifnum #1#2>#3#4  \xint_dothis{\CALX_ntoraw{\CALX_binom_a #1#2.#3#4.}}\fi
                      \xint_orthat{\CALX_ntoraw{\CALX_binom_a #3#4.#1#2.}}%
}%
```

x-k.k. avec 0<k<x, k<=x-k. Les divisions produiront en extra après le quotient un terminateur 1!\Z!0!. On va procéder par petite multiplication suivie par petite division. Donc ici on met le 1!\Z!0! pour amorcer.

The engine (\CALX_binom_a of calxint) has native N output, converted by \CALX_ntoraw.

22.54. \xintiiPFactorial

2015/11/29 for 1.2f. Partial factorial pfac(a,b)=(a+1)...b, only for non-negative integers with a<=b<10^8.

1.2h (2016/11/20) removes the non-negativity condition. It was a bit unfortunate that the code raised \xintError:OutOfRangePFac if 0<=a<=b<10^8 was violated. The rule now applied is to interpret pfac(a,b) as the product for a<j<=b (not as a ratio of Gamma function), hence if a>=b, return 1 because of an empty product. If a<b: if a<0, return 0 for b>=0 and (-1)^(b-a) times |b|...(|a|-1) for b<0. But only for the range 0<= a <= b < 10^8 is the macro result to be considered as stable.

```
\def\xintiiPFactorial {\romannumeral0\xintiipfactorial }%
\def\xintiipfactorial #1#2%
{%
    \expandafter\XINT_pfac_fork\the\numexpr#1\expandafter.\the\numexpr #2.%
}%
\def\xintPFactorial{\romannumeral0\xintpfactorial}%
\let\xintpfactorial\xintiipfactorial
```

The product is computed by calxcore's \CALX_npfac.

```
\def\XINT_pfac_fork #1#2.#3#4.%
{%
    \unless\ifnum #1#2<#3#4 \xint_dothis\XINT_pfac_one\fi
    \if-#3\xint_dothis\XINT_pfac_neg\fi
    \if-#1\xint_dothis\XINT_pfac_zero\fi
    \ifnum #3#4>\xint_c_e_viii_mone\xint_dothis\XINT_pfac_outofrange\fi
    \xint_orthat \XINT_pfac_a #1#2.#3#4.%
}%
\def\XINT_pfac_outofrange #1.#2.%
   {\XINT_legacysignalcondition{InvalidOperation}%
         {pFactorial with too large argument: #2 >= 10^8.}{}{ 0}}%
\def\XINT_pfac_one        #1.#2.{ 1}%
\def\XINT_pfac_zero       #1.#2.{ 0}%
\def\XINT_pfac_neg -#1.-#2.%
{%
    \ifnum #1>\xint_c_e_viii\xint_dothis\XINT_pfac_outofrange\fi
    \xint_orthat
   {\ifodd\numexpr#2-#1\relax\xint_afterfi{\expandafter-\romannumeral`&&@}\fi
    \expandafter\XINT_pfac_a }%
    \the\numexpr #2-\xint_c_i\expandafter.\the\numexpr#1-\xint_c_i.%
}%
\def\XINT_pfac_a #1.#2.{\CALX_ntoraw{\CALX_npfac{#1}{#2}}}%
```

22.55. \xintiiGCD

Computed by calxint's \CALX_ngcd (Euclid's algorithm on native integers).

```
\def\xintiiGCD {\romannumeral0\xintiigcd }%
\def\xintiigcd #1#2%
   {\CALX_ntoraw{\CALX_ngcd{\CALX_nfromraw{#1}}{\CALX_nfromraw{#2}}}}%
```

22.56. \xintiiGCDof

New with 1.09a (was located in xintgcd.sty).

1.2l adds protection against items being non-terminated \the\numexpr.
1.4 renames the macro into \xintiiGCDof and moves it here. Terminator modified to ^ for direct call by \xintiiexpr function.
1.4d fixes breakage inherited since 1.3d rom \xintiiGCD, in case any argument vanished. Currently does not support empty list of arguments.

```
\def\xintiiGCDof    {\romannumeral0\xintiigcdof }%
\def\xintiigcdof    #1{\expandafter\XINT_iigcdof_a\romannumeral`&&@#1^}%
\def\XINT_iigcdof_a #1{\expandafter\XINT_iigcdof_b\romannumeral`&&@#1!}%
\def\XINT_iigcdof_b #1!#2{\expandafter\XINT_iigcdof_c\romannumeral`&&@#2!{#1}!}%
\def\XINT_iigcdof_c #1{\xint_gob_til_^ #1\XINT_iigcdof_e ^\XINT_iigcdof_d #1}%
\def\XINT_iigcdof_d #1!{\expandafter\XINT_iigcdof_b\romannumeral0\xintiigcd {#1}}%
\def\XINT_iigcdof_e #1!#2!{ #2}%
```

22.57. \xintiiLCM

Computed by calxint's \CALX_nlcm.

```
\def\xintiiLCM {\romannumeral0\xintiilcm}%
\def\xintiilcm #1#2%
   {\CALX_ntoraw{\CALX_nlcm{\CALX_nfromraw{#1}}{\CALX_nfromraw{#2}}}}%
```

22.58. \xintiiLCMof

See comments of \xintiiGCDof.

```
\def\xintiiLCMof      {\romannumeral0\xintiilcmof }%
\def\xintiilcmof    #1{\expandafter\XINT_iilcmof_a\romannumeral`&&@#1^}%
\def\XINT_iilcmof_a #1{\expandafter\XINT_iilcmof_b\romannumeral`&&@#1!}%
\def\XINT_iilcmof_b #1!#2{\expandafter\XINT_iilcmof_c\romannumeral`&&@#2!{#1}!}%
\def\XINT_iilcmof_c #1{\xint_gob_til_^ #1\XINT_iilcmof_e ^\XINT_iilcmof_d #1}%
\def\XINT_iilcmof_d #1!{\expandafter\XINT_iilcmof_b\romannumeral0\xintiilcm {#1}}%
\def\XINT_iilcmof_e #1!#2!{ #2}%
\XINTrestorecatcodesendinput%
```

23. Package xintbinhex implementation

Contents

.1 Introduction 353
.2 Catcodes, ε-TEX and reload detection . . 353
.3 Package identification 354
.4 Storage macros 354
.5 Helper macros 356
.5.1 \XINT_zeroes_foriv 356
.5.2 \XINT_zeroes_foriii 356
.6 \xintDecToHex 357
.7 \xintDecToOct 359
.8 \xintDecToBin 362
.9 \xintHexToDec 363
.10 \xintOctToDec 366
.11 \xintBinToDec 367
.12 \xintHexToOct 368
.13 \xintHexToBin 368
.14 \xintCHexToBin 369
.15 \xintOctToHex 370
.16 \xintBinToHex 370
.17 \xintOctToBin 371
.18 \xintCOctToBin 371
.19 \xintBinToOct 372
.20 \xintLowerHex, \xintUpperHex 372

23.1. Introduction

The commenting is currently (2026/11/26) still very sparse.

1.2m (2017/07/31) rewrote entirely the original macros coming with 1.08 (2013/06/07).

At 1.2n the dependencies on xintcore were removed, so now the package loads only xintkernel.

Also at 1.2n (2017/08/06), \csname governed expansion was used at some places rather than \numexpr. This increased the maximal input sizes for \xintDecToHex, \xintDecToBin, and \xintBinToHex.

1.4n (2025/09/05) adds:

- Conversion to and from octal base.
- Usage of \expanded in place of the \csname-based expansion from 1.2n. This has increased the maximal input sizes. The increase is spectacular for the conversions between binary, octal and hexadecimal, the limit on size being solely dependent on TEX main memory size. This change however caused seemingly (but barely tested) a slight (1%) decrease in speed for \xintDecToHex and \xintHexToDec.

Note that the whole architecture was last re-thought in 2017 at a time \expanded did not exist. The most subtle coding, which is the one regarding conversion to and from decimal radix is of such a nature that I am not motivated at this stage to modify it other than minimally (see initial code comments for \xintDecToHex). So the \expanded was used therein but only for some subroutines. For the conversions between binary, octal, and hexadecimal, the whole expansion basically is governed by a single \expanded. But for legacy reasons and stylistic coherence across the codebase, the code keeps using \romannumeral0 trigger at the start of the macros with a systematic naming scheme. I am not much motivated to engage into systematic changes across about 20000 code lines in the xint bundle!

xintexpr loads xintbinhex automatically as of 1.4n. Formerly it supported only " prefix for hexadecimal, now it supports also ' for octal, as well as 0x, 0o and 0b.

1.4p adds \xintLowerHex and \xintUpperHex.

23.2. Catcodes, ε-TEX and reload detection

The code for reload detection was initially copied from Heiko Oberdiek's packages, then modified.

The method for catcodes was also initially directly inspired by these packages.

```
\begingroup\catcode61\catcode48\catcode32=10\relax%
  \catcode13=5    % ^^M
  \endlinechar=13 %
  \catcode123=1   % {
  \catcode125=2   % }
  \catcode64=11   % @
```

```
  \catcode44=12   % ,
  \catcode46=12   % .
  \catcode58=12   % :
  \catcode94=7    % ^
  \def\empty{}\def\space{ }\newlinechar10
  \def\z{\endgroup}%
  \expandafter\let\expandafter\x\csname ver@xintbinhex.sty\endcsname
  \expandafter\let\expandafter\w\csname ver@xintkernel.sty\endcsname
  \expandafter\ifx\csname numexpr\endcsname\relax
    \expandafter\ifx\csname PackageWarningNoLine\endcsname\relax
      \immediate\write128{^^JPackage xintbinhex Warning:^^J%
                          \space\space\space\space
                          \numexpr not available, aborting input.^^J}%
    \else
      \PackageWarningNoLine{xintbinhex}{\numexpr not available, aborting input}%
    \fi
    \def\z{\endgroup\endinput}%
  \else
    \ifx\x\relax   % plain-TeX, first loading of xintbinhex.sty
      \ifx\w\relax % but xintkernel.sty not yet loaded.
         \def\z{\endgroup\input xintkernel.sty\relax}%
      \fi
    \else
      \ifx\x\empty % LaTeX, first loading,
      % variable is initialized, but \ProvidesPackage not yet seen
          \ifx\w\relax % xintkernel.sty not yet loaded.
            \def\z{\endgroup\RequirePackage{xintkernel}}%
          \fi
      \else
        \def\z{\endgroup\endinput}% xintbinhex already loaded.
      \fi
    \fi
  \fi
\z%
\XINTsetupcatcodes% defined in xintkernel.sty
```

23.3. Package identification

```
\XINT_providespackage
\ProvidesPackage{xintbinhex}%
  [2026/11/26 v2.0a2 Radix conversions (2, 8, 10, and 16) (JFB)]%
```

23.4. Storage macros

Modified at 1.2n (2017/08/06). Switch to \csname-governed expansion at various places.

Modified at 1.4n (2025/09/05). Move two \newcount's to xintkernel.

Modified at 1.4n (2025/09/05). This release uses \expanded in place of \csname governed expansion. This means that the mysterious \endcsname's have vanished from the storage macros. And support for octal was added, hence a few more storage macros were added.

Added at 1.4p (2026/03/01) [on 2025/09/15]. New storage macros for expandable uppercasing and lowercasing utilities. I hesitate having the same but for all of a..z and A..Z but then should I allow more ascii chars beyond decimal digits?

```
\def\XINT_tmpa #1{\ifx\relax#1\else
  \expandafter\edef\csname XINT_csdth_#1\endcsname
  {\ifcase #1 0\or 1\or 2\or 3\or 4\or 5\or 6\or 7\or
              8\or 9\or A\or B\or C\or D\or E\or F\fi}%
  \expandafter\XINT_tmpa\fi }%
\XINT_tmpa {0}{1}{2}{3}{4}{5}{6}{7}{8}{9}{10}{11}{12}{13}{14}{15}\relax
\def\XINT_tmpa #1{\ifx\relax#1\else
  \expandafter\edef\csname XINT_csdtb_#1\endcsname
  {\ifcase #1
   0000\or 0001\or 0010\or 0011\or 0100\or 0101\or 0110\or 0111\or
   1000\or 1001\or 1010\or 1011\or 1100\or 1101\or 1110\or 1111\fi}%
  \expandafter\XINT_tmpa\fi }%
\XINT_tmpa {0}{1}{2}{3}{4}{5}{6}{7}{8}{9}{10}{11}{12}{13}{14}{15}\relax
\let\XINT_tmpa\relax
\expandafter\def\csname XINT_csbth_0000\endcsname {0}%
\expandafter\def\csname XINT_csbth_0001\endcsname {1}%
\expandafter\def\csname XINT_csbth_0010\endcsname {2}%
\expandafter\def\csname XINT_csbth_0011\endcsname {3}%
\expandafter\def\csname XINT_csbth_0100\endcsname {4}%
\expandafter\def\csname XINT_csbth_0101\endcsname {5}%
\expandafter\def\csname XINT_csbth_0110\endcsname {6}%
\expandafter\def\csname XINT_csbth_0111\endcsname {7}%
\expandafter\def\csname XINT_csbth_1000\endcsname {8}%
\expandafter\def\csname XINT_csbth_1001\endcsname {9}%
\expandafter\def\csname XINT_csbth_1010\endcsname {A}%
\expandafter\def\csname XINT_csbth_1011\endcsname {B}%
\expandafter\def\csname XINT_csbth_1100\endcsname {C}%
\expandafter\def\csname XINT_csbth_1101\endcsname {D}%
\expandafter\def\csname XINT_csbth_1110\endcsname {E}%
\expandafter\def\csname XINT_csbth_1111\endcsname {F}%
\expandafter\def\csname XINT_csbto_000\endcsname {0}%
\expandafter\def\csname XINT_csbto_001\endcsname {1}%
\expandafter\def\csname XINT_csbto_010\endcsname {2}%
\expandafter\def\csname XINT_csbto_011\endcsname {3}%
\expandafter\def\csname XINT_csbto_100\endcsname {4}%
\expandafter\def\csname XINT_csbto_101\endcsname {5}%
\expandafter\def\csname XINT_csbto_110\endcsname {6}%
\expandafter\def\csname XINT_csbto_111\endcsname {7}%
\expandafter\def\csname XINT_cshtb_0\endcsname {0000}%
\expandafter\def\csname XINT_cshtb_1\endcsname {0001}%
\expandafter\def\csname XINT_cshtb_2\endcsname {0010}%
\expandafter\def\csname XINT_cshtb_3\endcsname {0011}%
\expandafter\def\csname XINT_cshtb_4\endcsname {0100}%
\expandafter\def\csname XINT_cshtb_5\endcsname {0101}%
\expandafter\def\csname XINT_cshtb_6\endcsname {0110}%
\expandafter\def\csname XINT_cshtb_7\endcsname {0111}%
\expandafter\def\csname XINT_cshtb_8\endcsname {1000}%
\expandafter\def\csname XINT_cshtb_9\endcsname {1001}%
\def\XINT_cshtb_A {1010}%
\def\XINT_cshtb_B {1011}%
\def\XINT_cshtb_C {1100}%
\def\XINT_cshtb_D {1101}%
```

```
\def\XINT_cshtb_E {1110}%
\def\XINT_cshtb_F {1111}%
\expandafter\def\csname XINT_csotb_0\endcsname {000}%
\expandafter\def\csname XINT_csotb_1\endcsname {001}%
\expandafter\def\csname XINT_csotb_2\endcsname {010}%
\expandafter\def\csname XINT_csotb_3\endcsname {011}%
\expandafter\def\csname XINT_csotb_4\endcsname {100}%
\expandafter\def\csname XINT_csotb_5\endcsname {101}%
\expandafter\def\csname XINT_csotb_6\endcsname {110}%
\expandafter\def\csname XINT_csotb_7\endcsname {111}%
\def\XINT_tmpa#1#2{%
  \ifx#1\relax\else
    \expandafter\def\csname XINT_lower #1\endcsname{#1}%
    \expandafter\def\csname XINT_upper #1\endcsname{#2}%
    \expandafter\def\csname XINT_lower #2\endcsname{#1}%
    \expandafter\def\csname XINT_upper #2\endcsname{#2}%
  \expandafter\XINT_tmpa\fi
}%
\XINT_tmpa 0011223344556677889aAbBcCdDeEfF\relax\relax
```

23.5. Helper macros

We need at 1.4n to ensure lengths either multiple of 5, 4 or 3. For multiples of three we simply imitate what we had in 2017 for multiples of four. For multiples of five we will do it rather ``in place'' appealing to \xintLength and \xintReplicate for convenience. I suspected grabbing five by five (as we can't do ten by ten) would prove less efficient, but I am not motivated enough at this stage to devote the time needed for implementing and comparing.

23.5.1. \XINT_zeroes_foriv

```
   \romannumeral0\XINT_zeroes_foriv #1\R{0\R}{00\R}{000\R}%
                                       \R{0\R}{00\R}{000\R}\R\W
```

expands to <empty> or 0 or 00 or 000 as needed to prepend to #1 to extend it to length 4N.

```
\def\XINT_zeroes_foriv #1#2#3#4#5#6#7#8%
{%
    \xint_gob_til_R #8\XINT_zeroes_foriv_end\R\XINT_zeroes_foriv
}%
\def\XINT_zeroes_foriv_end\R\XINT_zeroes_foriv #1#2\W
   {\XINT_zeroes_foriv_done #1}%
\def\XINT_zeroes_foriv_done #1\R{ #1}%
```

23.5.2. \XINT_zeroes_foriii

Added at 1.4n (2025/09/05).

```
   \romannumeral0\XINT_zeroes_foriii #1\R{0\R}{00\R}%
                                        \R{0\R}{00\R}%
                                        \R{0\R}{00\R}\R\W
```

expands to <empty> or 0 or 00 as needed to prepend to #1 to extend it to length 3N.

```
\def\XINT_zeroes_foriii #1#2#3#4#5#6#7#8#9%
{%
    \xint_gob_til_R #9\XINT_zeroes_foriii_end\R\XINT_zeroes_foriii
}%
\def\XINT_zeroes_foriii_end\R\XINT_zeroes_foriii #1#2\W
```

```
    {\XINT_zeroes_foriii_done #1}%
\def\XINT_zeroes_foriii_done #1\R{ #1}%
```

23.6. \xintDecToHex

Now that illicit tools such as unravel exist, we can not hide anymore too much the crux of the matter. Let's simply say that the decimal input is tacitly converted into a sequence of successive elementary steps ``multiply by 10000 and add a base 10000 digit''. This is made to act on a sequence of radix 16^4 digits (note that $10000 < 16^4$). Those radix 16^4 digits are of course manipulated via \numexpr as 10^5-based digits (note that $16^4 < 10^5$ and that not only $16^4 \cdot 1000 < 10^9 < 2^{31}$ but also $10^9 + 16^4 \cdot 1000 < 2^{31}$ which helps in ensuring certain operations expand to a fixed number of digit tokens). When the sequence of elementary steps is complete, the base-16^4 digits only need to be converted to hexadecimal notation and purged of separators. Of course doing the whole thing expandably requires some skill. This was done when I was not yet an old man, and obviously was still very clever. Enjoy.

Modified at 1.2m (2017/07/31). Rewritten from scratch using the xintcore 1.2 style. Now guards against non terminated inputs.

Modified at 1.2n (2017/08/06). Coding improvements, \csname-governed expansion, increased maximal size.

Modified at 1.2o (2017/08/29). Again coding improvements (efficiency gain about 6%).

Modified at 1.4n (2025/09/05). Replacement of \csname-based expansion by usage of \expanded. This has increased the maximal input size from 12035 to 16042 digits (depends on nesting level; evaluated with TL2025 default TEX memory parameters, see user manual). I noticed a slight performance decrease of the order of 1% but did not test extensively.

Perhaps \expanded could be used to a deeper refactoring but... I preferred to make minimal changes, not having kept in mind all subtle details of the 2017 code.

```
\def\xintDecToHex {\romannumeral0\xintdectohex }%
\def\xintdectohex #1%
{%
    \expandafter\XINT_dth_checkin\romannumeral`&&@#1\xint:
}%
\def\XINT_dth_checkin #1%
{%
    \xint_UDsignfork
       #1\XINT_dth_neg
        -{\XINT_dth_main #1}%
     \krof
}%
\def\XINT_dth_neg {\expandafter-\romannumeral0\XINT_dth_main}%
\def\XINT_dth_main #1\xint:
{%
    \expandafter\XINT_dth_finish
    \romannumeral`&&@\expandafter\XINT_dthb_start
    \romannumeral0\XINT_zeroes_foriv
       #1\R{0\R}{00\R}{000\R}\R{0\R}{00\R}{000\R}\R\W
    #1\xint_bye\XINT_dth_tohex
}%
\def\XINT_dthb_start #1#2#3#4#5%
{%
    \xint_bye#5\XINT_dthb_small\xint_bye\XINT_dthb_start_a #1#2#3#4#5%
}%
\def\XINT_dthb_small\xint_bye\XINT_dthb_start_a #1\xint_bye#2{#2#1!}%
```

```
\def\XINT_dthb_start_a #1#2#3#4#5#6#7#8#9%
{%
    \expandafter\XINT_dthb_again
    \the\numexpr\expandafter\XINT_dthb_update
    \the\numexpr#1#2#3#4%
    \xint_bye#9\XINT_dthb_lastpass\xint_bye
    #5#6#7#8!\XINT_dthb_exclam\relax\XINT_dthb_nextfour #9%
}%
```

The 1.2n inserted exclamations marks, which when bumping back from \XINT_dthb_again gave rise to a \numexpr-loop which gathered the ! delimited arguments and inserted \expandafter\XINT_dthb_update\the\numexpr dynamically. The 1.2o trick is to insert it here immediately. Then at \XINT_dthb_again the \numexpr will trigger an already prepared chain.

The crux of the thing is handling of #3 at \XINT_dthb_update_a.

```
\def\XINT_dthb_exclam {!\XINT_dthb_exclam\relax
                       \expandafter\XINT_dthb_update\the\numexpr}%
\def\XINT_dthb_update #1!%
{%
    \expandafter\XINT_dthb_update_a
    \the\numexpr (#1+\xint_c_ii^xv)/\xint_c_ii^xvi-\xint_c_i\xint:
    #1\xint:%
}%
```

The four zeros here will get appended to prior digits. They enact the multiplication by 10000.

```
\def\XINT_dthb_update_a #1\xint:#2\xint:#3%
{%
    0000+#1\expandafter#3\the\numexpr#2-#1*\xint_c_ii^xvi
}%
\def\XINT_dthb_nextfour #1#2#3#4#5%
{%
    \xint_bye#5\XINT_dthb_lastpass\xint_bye
    #1#2#3#4!\XINT_dthb_exclam\relax\XINT_dthb_nextfour#5%
}%
\def\XINT_dthb_lastpass\xint_bye #1!#2\xint_bye#3{#1!#3!}%
\def\XINT_dthb_again #1!#2#3%
{%
    \ifx#3\relax
          \expandafter\xint_firstoftwo
      \else
          \expandafter\xint_secondoftwo
    \fi
    {\expandafter\XINT_dthb_again
     \the\numexpr
     \ifnum #1>\xint_c_
        \xint_afterfi{\expandafter\XINT_dthb_update\the\numexpr#1}%
     \fi}%
    {\ifnum #1>\xint_c_ \xint_dothis{#2#1!}\fi\xint_orthat{!#2!}}%
}%
```

1.4n replaces here \csname's method by simpler \expanded control. This is the part of the algorithm which rewrites base 16^4 (kept in decimal) digits into four hexadecimal digits.

```
\def\XINT_dth_tohex
{%
    \expandafter\XINT_dth_tohex_a\expanded\XINT_tofourhex
```

```
}%
\def\XINT_dth_tohex_a{!\XINT_dth_tohex!}%
\def\XINT_tofourhex #1!%
{%
    {\iffalse}\fi\expandafter\XINT_tofourhex_a
    \the\numexpr (#1+\xint_c_ii^vii)/\xint_c_ii^viii-\xint_c_i\xint:
    #1\xint:
}%
\def\XINT_tofourhex_a #1\xint:#2\xint:
{%
    \expandafter\XINT_tofourhex_c
    \the\numexpr (#1+\xint_c_viii)/\xint_c_xvi-\xint_c_i\xint:
    #1\xint:
    #2-\xint_c_ii^viii*#1!%
}%
\def\XINT_tofourhex_c #1\xint:#2\xint:
{%
    \csname XINT_csdth_#1\endcsname
    \csname XINT_csdth_\the\numexpr #2-\xint_c_xvi*#1\endcsname
    \expandafter\XINT_tofourhex_d\the\numexpr
}%
\def\XINT_tofourhex_d #1!%
{%
    \expandafter\XINT_tofourhex_e
    \the\numexpr (#1+\xint_c_viii)/\xint_c_xvi-\xint_c_i\xint:
    #1\xint:
}%
\def\XINT_tofourhex_e #1\xint:#2\xint:
{%
    \csname XINT_csdth_#1\endcsname
    \csname XINT_csdth_\the\numexpr #2-\xint_c_xvi*#1\endcsname
    \iffalse{\fi}%
}%
```

We only clean up up to 3 hexadecimal zeros, as output is produced in chunks of 4 hex digits, and (this comment added at 1.4n) as far as I understand the leading chunk can be 0000 only if input was vanishing.

The coding for this simple trimming may not be the most efficient, but this code is executed only once. I remember from 2017 that I had gotten tired to always try to optimize and did not even try to test efficiency.

```
\def\XINT_dth_finish !\XINT_dth_tohex!#1#2#3%
{%
    \unless\if#10\xint_dothis{ #1#2#3}\fi
    \unless\if#20\xint_dothis{ #2#3}\fi
    \unless\if#30\xint_dothis{ #3}\fi
    \xint_orthat{ }%
}%
```

23.7. \xintDecToOct

Added at 1.4n (2025/09/05).

This late extension to the package is imitated from (the \expanded updated) \xintDecToHex.

It does not share macros with \xintDecToHeX to same extent as \xintDecToBin does (those macros with _dthb_ in their names), because the input gets converted to radix 8^5, not radix 16^4.

Note that we have $10000 < 8^5 < 100000$ and refer to \xintDecToHex for some information on the algorithm.

```
\def\xintDecToOct {\romannumeral0\xintdectooct }%
\def\xintdectooct #1%
{%
    \expandafter\XINT_dto_checkin\romannumeral`&&@#1\xint:
}%
\def\XINT_dto_checkin #1%
{%
    \xint_UDsignfork
       #1\XINT_dto_neg
        -{\XINT_dto_main #1}%
     \krof
}%
\def\XINT_dto_neg {\expandafter-\romannumeral0\XINT_dto_main}%
\def\XINT_dto_main #1\xint:
{%
    \expandafter\XINT_dto_finish
    \romannumeral`&&@\expandafter\XINT_dto_start
    \romannumeral0\XINT_zeroes_foriv
       #1\R{0\R}{00\R}{000\R}\R{0\R}{00\R}{000\R}\R\W
    #1\xint_bye\XINT_dto_tooct
}%
\def\XINT_dto_start #1#2#3#4#5%
{%
    \xint_bye#5\XINT_dto_small\xint_bye\XINT_dto_start_a #1#2#3#4#5%
}%
\def\XINT_dto_small\xint_bye\XINT_dto_start_a #1\xint_bye#2{#2#1!}%
\def\XINT_dto_start_a #1#2#3#4#5#6#7#8#9%
{%
    \expandafter\XINT_dto_again\the\numexpr\expandafter\XINT_dto_update
    \the\numexpr#1#2#3#4%
    \xint_bye#9\XINT_dto_lastpass\xint_bye
    #5#6#7#8!\XINT_dto_exclam\relax\XINT_dto_nextfour #9%
}%
\def\XINT_dto_exclam {!\XINT_dto_exclam\relax
                      \expandafter\XINT_dto_update\the\numexpr}%
\def\XINT_dto_update #1!%
{%
    \expandafter\XINT_dto_update_a
    \the\numexpr (#1+\xint_c_ii^xiv)/\xint_c_ii^xv-\xint_c_i\xint:
    #1\xint:%
}%
\def\XINT_dto_update_a #1\xint:#2\xint:#3%
{%
    0000+#1\expandafter#3\the\numexpr#2-#1*\xint_c_ii^xv
}%
\def\XINT_dto_nextfour #1#2#3#4#5%
{%
    \xint_bye#5\XINT_dto_lastpass\xint_bye
```

```
    #1#2#3#4!\XINT_dto_exclam\relax\XINT_dto_nextfour#5%
}%
\def\XINT_dto_lastpass\xint_bye #1!#2\xint_bye#3{#1!#3!}%
\def\XINT_dto_again #1!#2#3%
{%
    \ifx#3\relax
          \expandafter\xint_firstoftwo
      \else
          \expandafter\xint_secondoftwo
    \fi
    {\expandafter\XINT_dto_again
     \the\numexpr
     \ifnum #1>\xint_c_
        \xint_afterfi{\expandafter\XINT_dto_update\the\numexpr#1}%
     \fi}%
    {\ifnum #1>\xint_c_ \xint_dothis{#2#1!}\fi\xint_orthat{!#2!}}%
}%
\def\XINT_dto_tooct
{%
    \expandafter\XINT_dto_tooct_a\expanded\XINT_tofiveoct
}%
\def\XINT_dto_tooct_a{!\XINT_dto_tooct!}%
\def\XINT_tofiveoct #1!%
{%
    {\iffalse}\fi\expandafter\XINT_tofiveoct_a
    \the\numexpr (#1+\xint_c_ii^viii)/\xint_c_ii^ix-\xint_c_i\xint:
    #1\xint:
}%
\def\XINT_tofiveoct_a #1\xint:#2\xint:
{%
    \expandafter\XINT_tofiveoct_c
    \the\numexpr (#1+\xint_c_iv)/\xint_c_viii-\xint_c_i\xint:#1\xint:
    #2-\xint_c_ii^ix*#1!%
}%
\def\XINT_tofiveoct_c #1\xint:#2\xint:
{%
    #1\the\numexpr #2-\xint_c_viii*#1\relax
    \expandafter\XINT_tofiveoct_d\the\numexpr
}%
\def\XINT_tofiveoct_d #1!%
{%
    \expandafter\XINT_tofiveoct_e
    \the\numexpr (#1+\xint_c_ii^v)/\xint_c_ii^vi-\xint_c_i\xint:
    #1\xint:
}%
\def\XINT_tofiveoct_e #1\xint:#2\xint:
{%
    #1\expandafter\XINT_tofiveoct_f\the\numexpr #2-\xint_c_ii^vi*#1!%
}%
\def\XINT_tofiveoct_f #1!%
{%
    \expandafter\XINT_tofiveoct_g
```

```
    \the\numexpr (#1+\xint_c_iv)/\xint_c_viii-\xint_c_i\xint:
    #1\xint:
}%
\def\XINT_tofiveoct_g #1\xint:#2\xint:
{%
    #1\the\numexpr #2-\xint_c_viii*#1\iffalse{\fi}%
}%
```

We only clean-up up to 4 zero octal digits, as output was produced in chunks of 5 octal digits and as far as this author understands his own code, the leading block can not be vanishing, except when input was itself only with zeros.

```
\def\XINT_dto_finish !\XINT_dto_tooct!#1#2#3#4%
{%
    \unless\if#10\xint_dothis{ #1#2#3#4}\fi
    \unless\if#20\xint_dothis{ #2#3#4}\fi
    \unless\if#30\xint_dothis{ #3#4}\fi
    \unless\if#40\xint_dothis{ #4}\fi
    \xint_orthat{ }%
}%
```

23.8. \xintDecToBin

An input without leading zeroes gives an output without leading zeroes.
Macros with _dtohb in their names are shared with \xintDecToHex.

Modified at 1.2m (2017/07/31). Complete rewrite in the 1.2 style. Also, 1.2m version is robust against non terminated inputs.

Modified at 1.2n (2017/08/06). Increased maximal size from using \csname-based expansion.

Modified at 1.4n (2025/09/05). Same use of \expanded as in \xintDecToHex.

```
\def\xintDecToBin {\romannumeral0\xintdectobin }%
\def\xintdectobin #1%
{%
    \expandafter\XINT_dtb_checkin\romannumeral`&&@#1\xint:
}%
\def\XINT_dtb_checkin #1%
{%
    \xint_UDsignfork
       #1\XINT_dtb_neg
        -{\XINT_dtb_main #1}%
     \krof
}%
\def\XINT_dtb_neg {\expandafter-\romannumeral0\XINT_dtb_main}%
\def\XINT_dtb_main #1\xint:
{%
    \expandafter\XINT_dtb_finish
    \romannumeral`&&@\expandafter\XINT_dthb_start
    \romannumeral0\XINT_zeroes_foriv
       #1\R{0\R}{00\R}{000\R}\R{0\R}{00\R}{000\R}\R\W
    #1\xint_bye\XINT_dtb_tobin
}%
\def\XINT_dtb_tobin
{%
    \expandafter\XINT_dtb_tobin_a\expanded\XINT_tosixteenbits
```

```
}%
\def\XINT_dtb_tobin_a{!\XINT_dtb_tobin!}%
\def\XINT_tosixteenbits #1!%
{%
    {\iffalse}\fi\expandafter\XINT_tosixteenbits_a
    \the\numexpr (#1+\xint_c_ii^vii)/\xint_c_ii^viii-\xint_c_i\xint:
    #1\xint:
}%
\def\XINT_tosixteenbits_a #1\xint:#2\xint:
{%
    \expandafter\XINT_tosixteenbits_c
    \the\numexpr (#1+\xint_c_viii)/\xint_c_xvi-\xint_c_i\xint:
    #1\xint:
    #2-\xint_c_ii^viii*#1!%
}%
\def\XINT_tosixteenbits_c #1\xint:#2\xint:
{%
    \csname XINT_csdtb_#1\endcsname
    \csname XINT_csdtb_\the\numexpr #2-\xint_c_xvi*#1\endcsname
    \expandafter\XINT_tosixteenbits_d\the\numexpr
}%
\def\XINT_tosixteenbits_d #1!%
{%
    \expandafter\XINT_tosixteenbits_e
    \the\numexpr (#1+\xint_c_viii)/\xint_c_xvi-\xint_c_i\xint:
    #1\xint:
}%
\def\XINT_tosixteenbits_e #1\xint:#2\xint:
{%
    \csname XINT_csdtb_#1\endcsname
    \csname XINT_csdtb_\the\numexpr #2-\xint_c_xvi*#1\endcsname
    \iffalse{\fi}%
}%
\def\XINT_dtb_finish !\XINT_dtb_tobin!#1#2#3#4#5#6#7#8%
{%
    \expandafter\XINT_dtb_finish_a\the\numexpr #1#2#3#4#5#6#7#8\relax
}%
\def\XINT_dtb_finish_a #1{%
\def\XINT_dtb_finish_a ##1##2##3##4##5##6##7##8##9%
{%
    \expandafter#1\the\numexpr ##1##2##3##4##5##6##7##8##9\relax
}}\XINT_dtb_finish_a { }%
```

23.9. \xintHexToDec

Completely (and belatedly) rewritten at 1.2m in the 1.2 style.

1.2m version robust against non terminated inputs, but there is no primitive from TeX which may generate hexadecimal digits and provoke expansion ahead, afaik, except of course if decimal digits are treated as hexadecimal. This robustness is not on purpose but from need to expand argument and then grab it again. So we do it safely.

Increased maximal size at 1.2n.

1.2m version robust against non terminated inputs.

An input without leading zeroes gives an output without leading zeroes.

```
\def\xintHexToDec {\romannumeral0\xinthextodec }%
\def\xinthextodec #1%
{%
    \expandafter\XINT_htd_checkin\romannumeral`&&@#1\xint:
}%
\def\XINT_htd_checkin #1%
{%
    \xint_UDsignfork
       #1\XINT_htd_neg
        -{\XINT_htd_main #1}%
     \krof
}%
\def\XINT_htd_neg {\expandafter-\romannumeral0\XINT_htd_main}%
\def\XINT_htd_main #1\xint:
{%
    \expandafter\XINT_htd_startb
    \the\numexpr\expandafter\XINT_htd_starta
    \romannumeral0\XINT_zeroes_foriv
       #1\R{0\R}{00\R}{000\R}\R{0\R}{00\R}{000\R}\R\W
    #1\xint_bye!2!3!4!5!6!7!8!9!\xint_bye\relax
}%
\def\XINT_htd_starta #1#2#3#4{"#1#2#3#4+\xint_c_e_v!}%
\def\XINT_htd_startb 1#1%
{%
    \if#10\expandafter\XINT_htd_startba\else
          \expandafter\XINT_htd_startbb
    \fi 1#1%
}%
\def\XINT_htd_startba 10#1!{\XINT_htd_again #1%
    \xint_bye!2!3!4!5!6!7!8!9!\xint_bye\XINT_htd_nextfour}%
\def\XINT_htd_startbb 1#1#2!{\XINT_htd_again #1!#2%
    \xint_bye!2!3!4!5!6!7!8!9!\xint_bye\XINT_htd_nextfour}%
```

It is a bit annoying to grab all to the end here. I had a version, modeled on the 1.2n variant of \xintDecToHex which solved that problem, but it did not prove much (or at all) faster in my brief testing and it had the defect of a reduced maximal allowed size of the input.

```
\def\XINT_htd_again #1\XINT_htd_nextfour #2%
{%
    \xint_bye #2\XINT_htd_finish\xint_bye
    \expandafter\XINT_htd_A\the\numexpr
    \XINT_htd_a #1\XINT_htd_nextfour #2%
}%
\def\XINT_htd_a #1!#2!#3!#4!#5!#6!#7!#8!#9!%
{%
    #1\expandafter\XINT_htd_update
    \the\numexpr #2\expandafter\XINT_htd_update
    \the\numexpr #3\expandafter\XINT_htd_update
    \the\numexpr #4\expandafter\XINT_htd_update
    \the\numexpr #5\expandafter\XINT_htd_update
    \the\numexpr #6\expandafter\XINT_htd_update
    \the\numexpr #7\expandafter\XINT_htd_update
    \the\numexpr #8\expandafter\XINT_htd_update
```

```
    \the\numexpr #9\expandafter\XINT_htd_update
    \the\numexpr \XINT_htd_a
}%
\def\XINT_htd_nextfour #1#2#3#4%
{%
    *\xint_c_ii^xvi+"#1#2#3#4+\xint_c_e_ix\relax\xint_bye!%
    2!3!4!5!6!7!8!9!\xint_bye\XINT_htd_nextfour
}%
\def\XINT_htd_update 1#1#2#3#4#5%
{%
    *\xint_c_ii^xvi+10000#1#2#3#4#5!%
}%
```

Modified at 1.4n (2025/09/05). \XINT_htd_A small refactoring to introduce shared macros with the octal conversion routine. Slight loss of efficiency.

```
\def\XINT_htd_A {\XINT_hotd_A\XINT_htd_again}%
\def\XINT_hotd_A #11#2%
{%
    \if#20\expandafter\XINT_hotd_Aa\else
          \expandafter\XINT_hotd_Ab
    \fi #11#2%
}%
\def\XINT_hotd_Aa #110#2#3#4#5{#1#2#3#4#5!}%
\def\XINT_hotd_Ab #11#2#3#4#5#6{#1#2!#3#4#5#6!}%
\def\XINT_htd_finish\xint_bye
    \expandafter\XINT_htd_A\the\numexpr \XINT_htd_a #1\XINT_htd_nextfour
{%
    \expandafter\XINT_htd_finish_cuz\the\numexpr0\XINT_htd_unsep_loop #1%
}%
\def\XINT_htd_unsep_loop #1!#2!#3!#4!#5!#6!#7!#8!#9!%
{%
    \expandafter\XINT_unsep_clean
    \the\numexpr 1#1#2\expandafter\XINT_unsep_clean
    \the\numexpr 1#3#4\expandafter\XINT_unsep_clean
    \the\numexpr 1#5#6\expandafter\XINT_unsep_clean
    \the\numexpr 1#7#8\expandafter\XINT_unsep_clean
    \the\numexpr 1#9\XINT_htd_unsep_loop_a
}%
\def\XINT_htd_unsep_loop_a #1!#2!#3!#4!#5!#6!#7!#8!#9!%
{%
    #1\expandafter\XINT_unsep_clean
    \the\numexpr 1#2#3\expandafter\XINT_unsep_clean
    \the\numexpr 1#4#5\expandafter\XINT_unsep_clean
    \the\numexpr 1#6#7\expandafter\XINT_unsep_clean
    \the\numexpr 1#8#9\XINT_htd_unsep_loop
}%
\def\XINT_unsep_clean 1{\relax}% also in xintcore
\def\XINT_htd_finish_cuz #1{%
\def\XINT_htd_finish_cuz ##1##2##3##4##5%
   {\expandafter#1\the\numexpr ##1##2##3##4##5\relax}%
}\XINT_htd_finish_cuz{ }%
```

23.10. \xintOctToDec

Added at 1.4n (2025/09/05). Yes, the explanations are somewhat lacking. Basically this imitates \xintxintHexToDec the main difference being to handle 5 octal digits at a time in place of 4 hexadecimal ones.

```
\def\xintOctToDec {\romannumeral0\xintocttodec }%
\def\xintocttodec #1%
{%
    \expandafter\XINT_otd_checkin\romannumeral`&&@#1\xint:
}%
\def\XINT_otd_checkin #1%
{%
    \xint_UDsignfork
       #1\XINT_otd_neg
        -{\XINT_otd_main #1}%
     \krof
}%
\def\XINT_otd_neg {\expandafter-\romannumeral0\XINT_otd_main}%
```

First we inject leading octal zeroes to make the length a multiple of 5. We do not try to code a direct way as with \XINT_zeroes_foriv, in part because as we can't grab 10 by 10, we would have to proceed 5 by 5, which for very long input may prove slower than using \xintLength combined with \xintReplicate, as provided by xintkernel. However, the author is not motivated enough to do the alternative coding and compare its efficiency with the one here.

```
\def\XINT_otd_main #1\xint:
{%
    \expandafter\XINT_otd_startb
    \the\numexpr\expandafter\XINT_otd_starta_i
                           \romannumeral0\xintlength{#1}\xint:
    #1\xint_bye!2!3!4!5!6!7!8!9!\xint_bye\relax
}%
\def\XINT_otd_starta_i #1\xint:
{%
    \expandafter\XINT_otd_starta
    \romannumeral\xintreplicate{\xint_c_v*((#1+\xint_c_ii)/\xint_c_v)-#1}{0}%
}%
```

Now we grab five octal digits and convert to decimal in a \numexpr.

```
\def\XINT_otd_starta #1#2#3#4#5{'#1#2#3#4#5+\xint_c_e_v!}%
\def\XINT_otd_startb 1#1%
{%
    \if#10\expandafter\XINT_otd_startba\else
          \expandafter\XINT_otd_startbb
    \fi 1#1%
}%
\def\XINT_otd_startba 10#1!{\XINT_otd_again #1%
    \xint_bye!2!3!4!5!6!7!8!9!\xint_bye\XINT_otd_nextfive}%
\def\XINT_otd_startbb 1#1#2!{\XINT_otd_again #1!#2%
    \xint_bye!2!3!4!5!6!7!8!9!\xint_bye\XINT_otd_nextfive}%
\def\XINT_otd_again #1\XINT_otd_nextfive #2%
{%
    \xint_bye #2\XINT_otd_finish\xint_bye
    \expandafter\XINT_otd_A\the\numexpr
    \XINT_otd_a #1\XINT_otd_nextfive #2%
```

```
}%
\def\XINT_otd_a #1!#2!#3!#4!#5!#6!#7!#8!#9!%
{%
    #1\expandafter\XINT_otd_update
    \the\numexpr #2\expandafter\XINT_otd_update
    \the\numexpr #3\expandafter\XINT_otd_update
    \the\numexpr #4\expandafter\XINT_otd_update
    \the\numexpr #5\expandafter\XINT_otd_update
    \the\numexpr #6\expandafter\XINT_otd_update
    \the\numexpr #7\expandafter\XINT_otd_update
    \the\numexpr #8\expandafter\XINT_otd_update
    \the\numexpr #9\expandafter\XINT_otd_update
    \the\numexpr \XINT_otd_a
}%
\def\XINT_otd_nextfive #1#2#3#4#5%
{%
    *\xint_c_ii^xv+'#1#2#3#4#5+\xint_c_e_ix\relax\xint_bye!%
    2!3!4!5!6!7!8!9!\xint_bye\XINT_otd_nextfive
}%
\def\XINT_otd_update 1#1#2#3#4#5%
{%
    *\xint_c_ii^xv+10000#1#2#3#4#5!%
}%
```

We can hook here into the \xintHexToDec final sub-routines.

```
\def\XINT_otd_A {\XINT_hotd_A\XINT_otd_again}%
\def\XINT_otd_finish\xint_bye
    \expandafter\XINT_otd_A\the\numexpr \XINT_otd_a #1\XINT_otd_nextfive
{%
    \expandafter\XINT_htd_finish_cuz\the\numexpr0\XINT_htd_unsep_loop #1%
}%
```

23.11. \xintBinToDec

Redone entirely for 1.2m. Starts by converting to hexadecimal first.
Increased maximal size at 1.2n.
An input without leading zeroes gives an output without leading zeroes.
Robust against non-terminated input.

```
\def\xintBinToDec {\romannumeral0\xintbintodec }%
\def\xintbintodec #1%
{%
    \expandafter\XINT_btd_checkin\romannumeral`&&@#1\xint:
}%
\def\XINT_btd_checkin #1%
{%
    \xint_UDsignfork
       #1\XINT_btd_N
        -{\XINT_btd_main #1}%
     \krof
}%
\def\XINT_btd_N {\expandafter-\romannumeral0\XINT_btd_main }%
\def\XINT_btd_main #1\xint:
{%
```

```
    \expandafter\XINT_btd_htd
    \expanded{\expandafter\XINT_bth_loop
    \romannumeral0\XINT_zeroes_foriv
       #1\R{0\R}{00\R}{000\R}\R{0\R}{00\R}{000\R}\R\W
    #1nonenone\xint_bye}\xint:
}%
\def\XINT_btd_htd #1\xint:
{%
    \expandafter\XINT_htd_startb
    \the\numexpr\expandafter\XINT_htd_starta
    \romannumeral0\XINT_zeroes_foriv
       #1\R{0\R}{00\R}{000\R}\R{0\R}{00\R}{000\R}\R\W
    #1\xint_bye!2!3!4!5!6!7!8!9!\xint_bye\relax
}%
```

23.12. \xintHexToOct

Added at 1.4n (2025/09/05). This is done the lazy way, from hexadecimal to binary to octal. I am simply not motivated enough at this stage to implement a direct conversion (3 hexa digits mapping to 4 octal ones), handle leading zeros, and compare efficiency and size limits with the simple minded one here.

```
\def\xintHexToOct {\romannumeral0\xinthextooct }%
\def\xinthextooct #1%
{%
    \expandafter\XINT_bto_checkin
    \romannumeral0\expandafter\XINT_htb_checkin\romannumeral`&&@#1.........\xint_bye
    \iffalse{\fi}%
    \xint:
}%
```

23.13. \xintHexToBin

Completely rewritten for 1.2m.

Only up to three zeros are removed on front of output: if the input had a leading zero, there will be a leading zero (and then possibly 4n of them if inputs had more leading zeroes) on output.

Rewritten again at 1.2n for \csname governed expansion.

Modified at 1.4n (2025/09/05). Use of \expanded (quasi globally, in contrast to the local uses made in the decimal conversions).. Dramatic increase of upper limit on the size of input.

As a by-product the initial expansion of the argument #1 is now guarded at the end by (many) full stops, but anyhow I don't know TEX construct potentially creating hexadecimal digits and potentially causing expansion of what comes next if not terminated.

```
\def\xintHexToBin {\romannumeral0\xinthextobin }%
\def\xinthextobin #1%
{%
    \expandafter\XINT_htb_checkin\romannumeral`&&@#1.........\xint_bye
    \iffalse{\fi}%
}%
\def\XINT_htb_checkin #1%
{%
    \xint_UDsignfork
       #1\XINT_htb_N
        -{\XINT_htb_main #1}%
```

```
      \krof
}%
\def\XINT_htb_N {\expandafter-\romannumeral0\XINT_htb_main }%
\def\XINT_htb_main
{%
    \expandafter\XINT_htb_cuz
    \expanded{\iffalse}\fi\XINT_htb_loop
}%
\def\XINT_htb_loop #1#2#3#4#5#6#7#8#9%
{%
     \csname XINT_cshtb_#1\endcsname
     \csname XINT_cshtb_#2\endcsname
     \csname XINT_cshtb_#3\endcsname
     \csname XINT_cshtb_#4\endcsname
     \csname XINT_cshtb_#5\endcsname
     \csname XINT_cshtb_#6\endcsname
     \csname XINT_cshtb_#7\endcsname
     \csname XINT_cshtb_#8\endcsname
     \csname XINT_cshtb_#9\endcsname
     \XINT_htb_loop
}%
\expandafter\let\csname XINT_cshtb_.\endcsname\xint_bye
\def\XINT_htb_cuz #1{%
\def\XINT_htb_cuz ##1##2##3##4%
   {\expandafter#1\the\numexpr##1##2##3##4\relax}%
}\XINT_htb_cuz { }%
```

23.14. \xintCHexToBin

The 1.08 macro had same functionality as \xintHexToBin, and slightly different code, the 1.2m version has the same code as \xintHexToBin except that it does not remove leading zeros from output: if the input had N hexadecimal digits, the output will have exactly 4N binary digits.

Rewritten again at 1.2n for \csname governed expansion.

Modified at 1.4n (2025/09/05). Kept in sync with new \xintHexToBin.

```
\def\xintCHexToBin {\romannumeral0\xintchextobin }%
\def\xintchextobin #1%
{%
    \expandafter\XINT_chtb_checkin\romannumeral`&&@#1.........\xint_bye
    \iffalse{\fi}%
}%
\def\XINT_chtb_checkin #1%
{%
    \xint_UDsignfork
       #1\XINT_chtb_N
        -{\XINT_chtb_main #1}%
     \krof
}%
\def\XINT_chtb_N {\expandafter-\romannumeral0\XINT_chtb_main }%
\def\XINT_chtb_main {\expanded{ \iffalse}\fi\XINT_htb_loop}%
```

23.15. \xintOctToHex

Added at 1.4n (2025/09/05). This is done the lazy way, from octal to binary to hexadecimal.

```
\def\xintOctToHex {\romannumeral0\xintocttohex }%
\def\xintocttohex #1%
{%
    \expandafter\XINT_bth_checkin
    \romannumeral0\expandafter\XINT_otb_checkin\romannumeral`&&@#1.........\xint_bye
    \iffalse{\fi}%
    \xint:
}%
```

23.16. \xintBinToHex

Modified at 1.2m (2017/07/31). Complete rewrite. Much more efficient but smaller maximals sizes.

Modified at 1.2n (2017/08/06). Again redone and now using \csname governed expansion: increased maximal size.

Size of output is ceil(size(input)/4). If the input has leading zeroes, they may exist in the output too. An input without leading zeroes gives an output without leading zeroes.

Modified at 1.4n (2025/09/05). Expansion is governed by \expanded. Tremendous increase of maximal size. Note that for legacy reasons of the package history, the primary trigger is via \romannumeral0, we could use \expanded upfront, with some advantages. But well.

```
\def\xintBinToHex {\romannumeral0\xintbintohex }%
\def\xintbintohex #1%
{%
    \expandafter\XINT_bth_checkin\romannumeral`&&@#1\xint:
}%
\def\XINT_bth_checkin #1%
{%
    \xint_UDsignfork
       #1\XINT_bth_N
        -{\XINT_bth_main #1}%
     \krof
}%
\def\XINT_bth_N {\expandafter-\romannumeral0\XINT_bth_main }%
\def\XINT_bth_main #1\xint:
{%
    \expanded{ \expandafter\XINT_bth_loop
    \romannumeral0\XINT_zeroes_foriv
       #1\R{0\R}{00\R}{000\R}\R{0\R}{00\R}{000\R}\R\W
    #1%
    nonenone\xint_bye}%
}%
\def\XINT_bth_loop #1#2#3#4#5#6#7#8%
{%
    \csname XINT_csbth_#1#2#3#4\endcsname
    \csname XINT_csbth_#5#6#7#8\endcsname
    \XINT_bth_loop
}%
\let\XINT_csbth_none\xint_bye
```

23.17. \xintOctToBin

Added at 1.4n (2025/09/05).

```
\def\xintOctToBin {\romannumeral0\xintocttobin }%
\def\xintocttobin #1%
{%
    \expandafter\XINT_otb_checkin\romannumeral`&&@#1.........\xint_bye
    \iffalse{\fi}%
}%
\def\XINT_otb_checkin #1%
{%
    \xint_UDsignfork
       #1\XINT_otb_N
        -{\XINT_otb_main #1}%
     \krof
}%
\def\XINT_otb_N {\expandafter-\romannumeral0\XINT_otb_main }%
\def\XINT_otb_main
{%
    \expandafter\XINT_otb_cuz
    \expanded{\iffalse}\fi \XINT_otb_loop
}%
\def\XINT_otb_loop #1#2#3#4#5#6#7#8#9%
{%
     \csname XINT_csotb_#1\endcsname
     \csname XINT_csotb_#2\endcsname
     \csname XINT_csotb_#3\endcsname
     \csname XINT_csotb_#4\endcsname
     \csname XINT_csotb_#5\endcsname
     \csname XINT_csotb_#6\endcsname
     \csname XINT_csotb_#7\endcsname
     \csname XINT_csotb_#8\endcsname
     \csname XINT_csotb_#9\endcsname
     \XINT_otb_loop
}%
\expandafter\let\csname XINT_csotb_.\endcsname\xint_bye
\def\XINT_otb_cuz #1{%
\def\XINT_otb_cuz ##1##2##3%
   {\expandafter#1\the\numexpr##1##2##3\relax}%
}\XINT_otb_cuz { }%
```

23.18. \xintCOctToBin

Added at 1.4n (2025/09/05).

```
\def\xintCOctToBin {\romannumeral0\xintcocttobin }%
\def\xintcocttobin #1%
{%
    \expandafter\XINT_cotb_checkin\romannumeral`&&@#1.........\xint_bye
    \iffalse{\fi}%
}%
\def\XINT_cotb_checkin #1%
{%
```

```
    \xint_UDsignfork
       #1\XINT_cotb_N
        -{\XINT_cotb_main #1}%
     \krof
}%
\def\XINT_cotb_N {\expandafter-\romannumeral0\XINT_cotb_main }%
\def\XINT_cotb_main {\expanded{ \iffalse}\fi\XINT_otb_loop}%
```

23.19. \xintBinToOct

Added at 1.4n (2025/09/05).

```
\def\xintBinToOct {\romannumeral0\xintbintooct }%
\def\xintbintooct #1%
{%
    \expandafter\XINT_bto_checkin\romannumeral`&&@#1\xint:
}%
\def\XINT_bto_checkin #1%
{%
    \xint_UDsignfork
       #1\XINT_bto_N
        -{\XINT_bto_main #1}%
     \krof
}%
\def\XINT_bto_N {\expandafter-\romannumeral0\XINT_bto_main }%
\def\XINT_bto_main #1\xint:
{%
    \expanded{ \expandafter\XINT_bto_loop
    \romannumeral0\XINT_zeroes_foriii
       #1\R{0\R}{00\R}\R{0\R}{00\R}\R{0\R}{00\R}\R\W
    #1%
    endendend\xint_bye}%
}%
\def\XINT_bto_loop #1#2#3#4#5#6#7#8#9%
{%
    \csname XINT_csbto_#1#2#3\endcsname
    \csname XINT_csbto_#4#5#6\endcsname
    \csname XINT_csbto_#7#8#9\endcsname
    \XINT_bto_loop
}%
\let\XINT_csbto_end\xint_bye
```

23.20. \xintLowerHex, \xintUpperHex

Added at 1.4p (2026/03/01) [on 2025/09/15].

I really hesitate breaking with the \romannumeral0 expansion trigger which is used throughout the codebase with very few exceptions. This adds to documentation chores. Also there is no absolute necessity to keep defining the lowercase \xintlowerhex but it can help when one needs to propagate expansion.

This is modeled on a bnumexpr 1.7 addition.

There is a deliberate space after #1 to accomodate for empty input.

```
\def\xintLowerHex{\expanded\xintlowerhex}%
\def\xintlowerhex#1{%
```

```
    {\expandafter\XINT_lowerhex\romannumeral`&&@#1 %
                \xint_bye23456789\xint_bye\endcsname}%
}%
\def\XINT_lowerhex #1#2#3#4#5#6#7#8#9{%
    \csname XINT_lower #1\endcsname
    \csname XINT_lower #2\endcsname
    \csname XINT_lower #3\endcsname
    \csname XINT_lower #4\endcsname
    \csname XINT_lower #5\endcsname
    \csname XINT_lower #6\endcsname
    \csname XINT_lower #7\endcsname
    \csname XINT_lower #8\endcsname
    \csname XINT_lower #9\endcsname
    \XINT_lowerhex
}%
\expandafter\let\csname XINT_lower \endcsname\empty
\def\xintUpperHex{\expanded\xintupperhex}%
\def\xintupperhex#1{%
    {\expandafter\XINT_upperhex\romannumeral`&&@#1 %
                \xint_bye23456789\xint_bye\endcsname}%
}%
\def\XINT_upperhex #1#2#3#4#5#6#7#8#9{%
    \csname XINT_upper #1\endcsname
    \csname XINT_upper #2\endcsname
    \csname XINT_upper #3\endcsname
    \csname XINT_upper #4\endcsname
    \csname XINT_upper #5\endcsname
    \csname XINT_upper #6\endcsname
    \csname XINT_upper #7\endcsname
    \csname XINT_upper #8\endcsname
    \csname XINT_upper #9\endcsname
    \XINT_upperhex
}%
\expandafter\let\csname XINT_upper \endcsname\empty
\XINTrestorecatcodesendinput%
```

24. Package xintgcd implementation

Contents

.1 Introduction 374
.2 Catcodes, ε-TEX and reload detection . . 374
.3 Package identification 375
.4 \xintBezout 375
.5 \xintEuclideAlgorithm 379
.6 \xintBezoutAlgorithm 380
.7 \xintTypesetEuclideAlgorithm 382
.8 \xintTypesetBezoutAlgorithm 383

24.1. Introduction

The commenting is currently (2026/11/26) very sparse.

Release 1.09h has modified a bit the \xintTypesetEuclideAlgorithm and \xintTypesetBezoutAlgorithm layout with respect to line indentation in particular. And they use the xinttools \xintloop rather than the Plain TEX or LATEX's \loop.

Breaking change at 1.2p: \xintBezout{A}{B} formerly had output {A}{B}{U}{V}{D} with AU-BV=D, now it is {U}{V}{D} with AU+BV=D.

From 1.1 to 1.3f the package loaded only xintcore. At 1.4 it now automatically loads both of xint and xinttools (the latter being in fact a requirement of \xintTypesetEuclideAlgorithm and \xintTypesetBezoutAlgorithm since 1.09h).

CHANGED AT 1.4!

At 1.4 \xintGCD, \xintLCM, \xintGCDof, and \xintLCMof are *removed* from the package:they are provided only by xintfrac and they handle general fractions, not only integers.

The original integer-only macros have been renamed into respectively \xintiiGCD, \xintiiLCM, \xintiiGCDof, and \xintiiLCMof and got relocated into xint package.

24.2. Catcodes, ε-TEX and reload detection

The code for reload detection was initially copied from Heiko Oberdiek's packages, then modified.

The method for catcodes was also initially directly inspired by these packages.

```
\begingroup\catcode61\catcode48\catcode32=10\relax%
  \catcode13=5    % ^^M
  \endlinechar=13 %
  \catcode123=1   % {
  \catcode125=2   % }
  \catcode64=11   % @
  \catcode44=12   % ,
  \catcode46=12   % .
  \catcode58=12   % :
  \catcode94=7    % ^
  \def\empty{}\def\space{ }\newlinechar10
  \def\z{\endgroup}%
  \expandafter\let\expandafter\x\csname ver@xintgcd.sty\endcsname
  \expandafter\let\expandafter\w\csname ver@xint.sty\endcsname
  \expandafter\let\expandafter\t\csname ver@xinttools.sty\endcsname
  \expandafter\ifx\csname numexpr\endcsname\relax
    \expandafter\ifx\csname PackageWarningNoLine\endcsname\relax
      \immediate\write128{^^JPackage xintgcd Warning:^^J%
                          \space\space\space\space
                          \numexpr not available, aborting input.^^J}%
    \else
```

```
      \PackageWarningNoLine{xintgcd}{\numexpr not available, aborting input}%
    \fi
    \def\z{\endgroup\endinput}%
  \else
    \ifx\x\relax   % plain-TeX, first loading of xintgcd.sty
      \ifx\w\relax % but xint.sty not yet loaded.
         \expandafter\def\expandafter\z\expandafter{\z\input xint.sty\relax}%
      \fi
      \ifx\t\relax % but xinttools.sty not yet loaded.
         \expandafter\def\expandafter\z\expandafter{\z\input xinttools.sty\relax}%
      \fi
    \else
      \ifx\x\empty % LaTeX, first loading,
      % variable is initialized, but \ProvidesPackage not yet seen
          \ifx\w\relax % xint.sty not yet loaded.
            \expandafter\def\expandafter\z\expandafter{\z\RequirePackage{xint}}%
          \fi
          \ifx\t\relax % xinttools.sty not yet loaded.
            \expandafter\def\expandafter\z\expandafter{\z\RequirePackage{xinttools}}%
          \fi
      \else
        \def\z{\endgroup\endinput}% xintgcd already loaded.
      \fi
    \fi
  \fi
\z%
\XINTsetupcatcodes% defined in xintkernel.sty
```

24.3. Package identification

```
\XINT_providespackage
\ProvidesPackage{xintgcd}%
  [2026/11/26 v2.0a2 Euclide algorithm with xint package (JFB)]%
```

24.4. \xintBezout

\xintBezout{#1}{#2} produces {U}{V}{D} with UA+VB=D, D = PGCD(A,B) (non-positive), where #1 and #2 f-expand to big integers A and B.

I had not checked this macro for about three years when I realized in January 2017 that \xintBezout{A}{B} was buggy for the cases A = 0 or B = 0. I fixed that blemish in 1.2l but overlooked the other blemish that \xintBezout{A}{B} with A multiple of B produced a coefficient U as -0 in place of 0.

Hence I rewrote again for 1.2p. On this occasion I modified the output of the macro to be {U}{V}{D} with AU+BV=D, formerly it was {A}{B}{U}{V}{D} with AU - BV = D. This is quite breaking change!

Note in particular change of sign of V.

I don't know why I had designed this macro to contain {A}{B} in its output. Perhaps I initially intended to output {A//D}{B//D} (but forgot), as this is actually possible from outcome of the last iteration, with no need of actually dividing. Current code however arranges to skip this last update, as U and V are already furnished by the iteration prior to realizing that the last non-zero remainder was found.

Also 1.2l raised InvalidOperation if both A and B vanished, but I removed this behaviour at 1.2p.

```
\def\xintBezout {\romannumeral0\xintbezout }%
\def\xintbezout #1%
{%
    \expandafter\XINT_bezout\expandafter {\romannumeral0\xintnum{#1}}%
}%
\def\XINT_bezout #1#2%
{%
    \expandafter\XINT_bezout_fork \romannumeral0\xintnum{#2}\Z #1\Z
}%
```

#3#4 = A, #1#2=B. Micro improvement for 1.21.

```
\def\XINT_bezout_fork #1#2\Z #3#4\Z
{%
    \xint_UDzerosfork
     #1#3\XINT_bezout_botharezero
      #10\XINT_bezout_secondiszero
      #30\XINT_bezout_firstiszero
       00\xint_UDsignsfork
    \krof
          #1#3\XINT_bezout_minusminus % A < 0, B < 0
           #1-\XINT_bezout_minusplus  % A > 0, B < 0
           #3-\XINT_bezout_plusminus  % A < 0, B > 0
            --\XINT_bezout_plusplus   % A > 0, B > 0
    \krof
    {#2}{#4}#1#3% #1#2=B, #3#4=A
}%
\def\XINT_bezout_botharezero #1\krof#2#300{{0}{0}{0}}%
\def\XINT_bezout_firstiszero #1\krof#2#3#4#5%
{%
    \xint_UDsignfork
      #4{{0}{-1}{#2}}%
       -{{0}{1}{#4#2}}%
    \krof
}%
\def\XINT_bezout_secondiszero #1\krof#2#3#4#5%
{%
    \xint_UDsignfork
       #5{{-1}{0}{#3}}%
        -{{1}{0}{#5#3}}%
    \krof
}%
```

#4#2= A < 0, #3#1 = B < 0

```
\def\XINT_bezout_minusminus #1#2#3#4%
{%
    \expandafter\XINT_bezout_mm_post
    \romannumeral0\expandafter\XINT_bezout_preloop_a
    \romannumeral0\XINT_ndivrawpair {#1}{#2}{#1}%
}%
\def\XINT_bezout_mm_post #1#2%
{%
    \expandafter\XINT_bezout_mm_postb\expandafter
    {\romannumeral0\xintiiopp{#2}}{\romannumeral0\xintiiopp{#1}}%
```

```
}%
\def\XINT_bezout_mm_postb #1#2{\expandafter{#2}{#1}}%
```

minusplus #4#2= A > 0, B < 0

```
\def\XINT_bezout_minusplus #1#2#3#4%
{%
    \expandafter\XINT_bezout_mp_post
    \romannumeral0\expandafter\XINT_bezout_preloop_a
    \romannumeral0\XINT_ndivrawpair {#1}{#4#2}{#1}%
}%
\def\XINT_bezout_mp_post #1#2%
{%
    \expandafter\xint_exchangetwo_keepbraces\expandafter
    {\romannumeral0\xintiiopp {#2}}{#1}%
}%
```

plusminus A < 0, B > 0

```
\def\XINT_bezout_plusminus #1#2#3#4%
{%
    \expandafter\XINT_bezout_pm_post
    \romannumeral0\expandafter\XINT_bezout_preloop_a
    \romannumeral0\XINT_ndivrawpair {#3#1}{#2}{#3#1}%
}%
\def\XINT_bezout_pm_post #1{\expandafter{\romannumeral0\xintiiopp{#1}}}%
```

plusplus, B = #3#1 > 0, A = #4#2 > 0

```
\def\XINT_bezout_plusplus #1#2#3#4%
{%
    \expandafter\XINT_bezout_preloop_a
    \romannumeral0\XINT_ndivrawpair {#3#1}{#4#2}{#3#1}%
}%
```

```
   n = 0: BA1001 (B, A, e=1, vv, uu, v, u)
   r(1)=B, r(0)=A, après n étapes {r(n+1)}{r(n)}{vv}{uu}{v}{u}
   q(n) quotient de r(n-1) par r(n)
   si reste nul, exit et renvoie U = -e*uu, V = e*vv, A*U+B*V=D
   sinon mise à jour
     vv, v = q * vv + v, vv
     uu, u = q * uu + u, uu
     e = -e
   puis calcul quotient reste et itération
```

We arrange for \xintiiMul sub-routine to be called only with positive arguments, thus skipping some un-needed sign parsing there. For that though we have to screen out the special cases A divides B, or B divides A. And we first want to exchange A and B if A < B. These special cases are the only one possibly leading to U or V zero (for A and B positive which is the case here.) Thus the general case always leads to non-zero U and V's and assigning a final sign is done simply adding a - to one of them, with no fear of producing -0.

```
\def\XINT_bezout_preloop_a #1#2#3%
{%
    \if0#1\xint_dothis\XINT_bezout_preloop_exchange\fi
    \if0#2\xint_dothis\XINT_bezout_preloop_exit\fi
    \xint_orthat{\expandafter\XINT_bezout_loop_B}%
    \romannumeral0\XINT_ndivrawpair {#2}{#3}{#2}{#1}110%
}%
```

```
\def\XINT_bezout_preloop_exit
    \romannumeral0\XINT_ndivrawpair #1#2#3#4#5#6#7%
{%
    {0}{1}{#2}%
}%
\def\XINT_bezout_preloop_exchange
{%
    \expandafter\xint_exchangetwo_keepbraces
    \romannumeral0\expandafter\XINT_bezout_preloop_A
}%
\def\XINT_bezout_preloop_A #1#2#3#4%
{%
    \if0#2\xint_dothis\XINT_bezout_preloop_exit\fi
    \xint_orthat{\expandafter\XINT_bezout_loop_B}%
    \romannumeral0\XINT_ndivrawpair {#2}{#3}{#2}{#1}%
}%
\def\XINT_bezout_loop_B #1#2%
{%
    \if0#2\expandafter\XINT_bezout_exitA
     \else\expandafter\XINT_bezout_loop_C
    \fi {#1}{#2}%
}%
```

We use the fact that the \romannumeral-`0 (or equivalent) done by \xintiiadd will absorb the initial space token left by \XINT_mul_plusplus in its output.

We arranged for operands here to be always positive which is needed for \XINT_mul_plusplus entry point (last time I checked...). Admittedly this kind of optimization is not good for maintenance of code, but I can't resist temptation of limiting the shuffling around of tokens...

```
\def\XINT_bezout_loop_C #1#2#3#4#5#6#7%
{%
    \expandafter\XINT_bezout_loop_D\expandafter
        {\romannumeral0\xintiiadd{\xintiimul{#1}{#4}}{#6}}%
        {\romannumeral0\xintiiadd{\xintiimul{#1}{#5}}{#7}}%
    {#2}{#3}{#4}{#5}%
}%
\def\XINT_bezout_loop_D #1#2%
{%
    \expandafter\XINT_bezout_loop_E\expandafter{#2}{#1}%
}%
\def\XINT_bezout_loop_E #1#2#3#4%
{%
    \expandafter\XINT_bezout_loop_b
    \romannumeral0\XINT_ndivrawpair {#3}{#4}{#3}{#2}{#1}%
}%
\def\XINT_bezout_loop_b #1#2%
{%
    \if0#2\expandafter\XINT_bezout_exita
     \else\expandafter\XINT_bezout_loop_c
    \fi {#1}{#2}%
}%
\def\XINT_bezout_loop_c #1#2#3#4#5#6#7%
{%
    \expandafter\XINT_bezout_loop_d\expandafter
```

```
        {\romannumeral0\xintiiadd{\xintiimul{#1}{#4}}{#6}}%
        {\romannumeral0\xintiiadd{\xintiimul{#1}{#5}}{#7}}%
    {#2}{#3}{#4}{#5}%
}%
\def\XINT_bezout_loop_d #1#2%
{%
    \expandafter\XINT_bezout_loop_e\expandafter{#2}{#1}%
}%
\def\XINT_bezout_loop_e #1#2#3#4%
{%
    \expandafter\XINT_bezout_loop_B
    \romannumeral0\XINT_ndivrawpair {#3}{#4}{#3}{#2}{#1}%
}%
```

sortir U, V, D mais on a travaillé avec vv, uu, v, u dans cet ordre.
The code is structured so that #4 and #5 are guaranteed non-zero if we exit here, hence we can not create a -0 in output.

```
\def\XINT_bezout_exita #1#2#3#4#5#6#7{{-#5}{#4}{#3}}%
\def\XINT_bezout_exitA #1#2#3#4#5#6#7{{#5}{-#4}{#3}}%
```

24.5. \xintEuclideAlgorithm

Pour Euclide: {N}{A}{D=r(n)}{B}{q1}{r1}{q2}{r2}{q3}{r3}....{qN}{rN=0}
u<2n> = u<2n+3>u<2n+2> + u<2n+4> à la n ième étape.
Formerly, used \xintiabs, but got deprecated at 1.2o.

```
\def\xintEuclideAlgorithm {\romannumeral0\xinteuclidealgorithm }%
\def\xinteuclidealgorithm #1%
{%
    \expandafter\XINT_euc\expandafter{\romannumeral0\xintiiabs{\xintNum{#1}}}%
}%
\def\XINT_euc #1#2%
{%
    \expandafter\XINT_euc_fork\romannumeral0\xintiiabs{\xintNum{#2}}\Z #1\Z
}%
```

Ici #3#4=A, #1#2=B

```
\def\XINT_euc_fork #1#2\Z #3#4\Z
{%
    \xint_UDzerofork
      #1\XINT_euc_BisZero
      #3\XINT_euc_AisZero
       0\XINT_euc_a
    \krof
    {0}{#1#2}{#3#4}{{#3#4}{#1#2}}{}\Z
}%
```

Le {} pour protéger {{A}{B}} si on s'arrête après une étape (B divise A). On va renvoyer:
{N}{A}{D=r(n)}{B}{q1}{r1}{q2}{r2}{q3}{r3}....{qN}{rN=0}

```
\def\XINT_euc_AisZero #1#2#3#4#5#6{{1}{0}{#2}{#2}{0}{0}}%
\def\XINT_euc_BisZero #1#2#3#4#5#6{{1}{0}{#3}{#3}{0}{0}}%
```

{n}{rn}{an}{{qn}{rn}}...{{A}{B}}{}\Z
a(n) = r(n-1). Pour n=0 on a juste {0}{B}{A}{{A}{B}}{}\Z
\XINT_ndivrawpair {u}{v} divise v par u

```
\def\XINT_euc_a #1#2#3%
{%
    \expandafter\XINT_euc_b\the\numexpr #1+\xint_c_i\expandafter.%
    \romannumeral0\XINT_ndivrawpair {#2}{#3}{#2}%
}%
```

```
  {n+1}{q(n+1)}{r(n+1)}{rn}{{qn}{rn}}...
\def\XINT_euc_b #1.#2#3#4%
{%
    \XINT_euc_c #3\Z {#1}{#3}{#4}{{#2}{#3}}%
}%
```

```
  r(n+1)\Z {n+1}{r(n+1)}{r(n)}{{q(n+1)}{r(n+1)}}{{qn}{rn}}...
Test si r(n+1) est nul.
\def\XINT_euc_c #1#2\Z
{%
    \xint_gob_til_zero #1\XINT_euc_end0\XINT_euc_a
}%
```

```
  {n+1}{r(n+1)}{r(n)}{{q(n+1)}{r(n+1)}}...{}\Z Ici r(n+1) = 0.  On arrête on se prépare à in-
verser {n+1}{0}{r(n)}{{q(n+1)}{r(n+1)}}.....{{q1}{r1}}{{A}{B}}{}\Z
On veut renvoyer:  {N=n+1}{A}{D=r(n)}{B}{q1}{r1}{q2}{r2}{q3}{r3}....{qN}{rN=0}
\def\XINT_euc_end0\XINT_euc_a #1#2#3#4\Z%
{%
    \expandafter\XINT_euc_end_a
    \romannumeral0%
    \XINT_rord_main {}#4{{#1}{#3}}%
    \xint:
      \xint_bye\xint_bye\xint_bye\xint_bye
      \xint_bye\xint_bye\xint_bye\xint_bye
    \xint:
}%
\def\XINT_euc_end_a #1#2#3{{#1}{#3}{#2}}%
```

24.6. \xintBezoutAlgorithm

```
Pour Bezout:  objectif, renvoyer
{N}{A}{0}{1}{D=r(n)}{B}{1}{0}{q1}{r1}{alpha1=q1}{beta1=1}
{q2}{r2}{alpha2}{beta2}....{qN}{rN=0}{alphaN=A/D}{betaN=B/D}
alpha0=1, beta0=0, alpha(-1)=0, beta(-1)=1
\def\xintBezoutAlgorithm {\romannumeral0\xintbezoutalgorithm }%
\def\xintbezoutalgorithm #1%
{%
    \expandafter \XINT_bezalg
    \expandafter{\romannumeral0\xintiiabs{\xintNum{#1}}}%
}%
\def\XINT_bezalg #1#2%
{%
    \expandafter\XINT_bezalg_fork\romannumeral0\xintiiabs{\xintNum{#2}}\Z #1\Z
}%
```

```
  Ici #3#4=A, #1#2=B
\def\XINT_bezalg_fork #1#2\Z #3#4\Z
```

```
{%
    \xint_UDzerofork
      #1\XINT_bezalg_BisZero
      #3\XINT_bezalg_AisZero
       0\XINT_bezalg_a
    \krof
    0{#1#2}{#3#4}1001{{#3#4}{#1#2}}{}\Z
}%
\def\XINT_bezalg_AisZero #1#2#3\Z{{1}{0}{0}{1}{#2}{#2}{1}{0}{0}{0}{0}{1}}%
\def\XINT_bezalg_BisZero #1#2#3#4\Z{{1}{0}{0}{1}{#3}{#3}{1}{0}{0}{0}{0}{1}}%
```

pour préparer l'étape n+1 il faut {n}{r(n)}{r(n-1)}{alpha(n)}{beta(n)}{alpha(n-1)}{beta(n-1)} {{q(n)}{r(n)}{alpha(n)}{beta(n)}}... division de #3 par #2

```
\def\XINT_bezalg_a #1#2#3%
{%
    \expandafter\XINT_bezalg_b\the\numexpr #1+\xint_c_i\expandafter.%
    \romannumeral0\XINT_ndivrawpair {#2}{#3}{#2}%
}%
```

{n+1}{q(n+1)}{r(n+1)}{r(n)}{alpha(n)}{beta(n)}{alpha(n-1)}{beta(n-1)}...

```
\def\XINT_bezalg_b #1.#2#3#4#5#6#7#8%
{%
    \expandafter\XINT_bezalg_c\expandafter
     {\romannumeral0\xintiiadd {\xintiiMul {#6}{#2}}{#8}}%
     {\romannumeral0\xintiiadd {\xintiiMul {#5}{#2}}{#7}}%
     {#1}{#2}{#3}{#4}{#5}{#6}%
}%
```

{beta(n+1)}{alpha(n+1)}{n+1}{q(n+1)}{r(n+1)}{r(n)}{alpha(n)}{beta(n)}

```
\def\XINT_bezalg_c #1#2#3#4#5#6%
{%
    \expandafter\XINT_bezalg_d\expandafter {#2}{#3}{#4}{#5}{#6}{#1}%
}%
```

{alpha(n+1)}{n+1}{q(n+1)}{r(n+1)}{r(n)}{beta(n+1)}

```
\def\XINT_bezalg_d #1#2#3#4#5#6#7#8%
{%
    \XINT_bezalg_e #4\Z {#2}{#4}{#5}{#1}{#6}{#7}{#8}{{#3}{#4}{#1}{#6}}%
}%
```

r(n+1)\Z {n+1}{r(n+1)}{r(n)}{alpha(n+1)}{beta(n+1)}
{alpha(n)}{beta(n)}{q,r,alpha,beta(n+1)}
Test si r(n+1) est nul.

```
\def\XINT_bezalg_e #1#2\Z
{%
    \xint_gob_til_zero #1\XINT_bezalg_end0\XINT_bezalg_a
}%
```

Ici r(n+1) = 0. On arrête on se prépare à inverser.
{n+1}{r(n+1)}{r(n)}{alpha(n+1)}{beta(n+1)}{alpha(n)}{beta(n)}
{q,r,alpha,beta(n+1)}...{{A}{B}}{}\Z
On veut renvoyer
{N}{A}{0}{1}{D=r(n)}{B}{1}{0}{q1}{r1}{alpha1=q1}{beta1=1}
{q2}{r2}{alpha2}{beta2}....{qN}{rN=0}{alphaN=A/D}{betaN=B/D}

```
\def\XINT_bezalg_end0\XINT_bezalg_a #1#2#3#4#5#6#7#8\Z
{%
    \expandafter\XINT_bezalg_end_a
    \romannumeral0%
    \XINT_rord_main {}#8{{#1}{#3}}%
    \xint:
      \xint_bye\xint_bye\xint_bye\xint_bye
      \xint_bye\xint_bye\xint_bye\xint_bye
    \xint:
}%
```

```
  {N}{D}{A}{B}{q1}{r1}{alpha1=q1}{beta1=1}{q2}{r2}{alpha2}{beta2}
....{qN}{rN=0}{alphaN=A/D}{betaN=B/D}
On veut renvoyer
{N}{A}{0}{1}{D=r(n)}{B}{1}{0}{q1}{r1}{alpha1=q1}{beta1=1}
{q2}{r2}{alpha2}{beta2}....{qN}{rN=0}{alphaN=A/D}{betaN=B/D}
```

```
\def\XINT_bezalg_end_a #1#2#3#4{{#1}{#3}{0}{1}{#2}{#4}{1}{0}}%
```

24.7. \xintTypesetEuclideAlgorithm

```
TYPESETTING
  Organisation:
  {N}{A}{D}{B}{q1}{r1}{q2}{r2}{q3}{r3}....{qN}{rN=0}
\U1 = N = nombre d'étapes, \U3 = PGCD, \U2 = A, \U4=B q1 = \U5, q2 = \U7 --> qn = \U<2n+3>, rn =
\U<2n+4> bn = rn.  B = r0.  A=r(-1)
  r(n-2) = q(n)r(n-1)+r(n) (n e étape)
  \U{2n} = \U{2n+3} \times \U{2n+2} + \U{2n+4}, n e étape.  (avec n entre 1 et N)
  1.09h uses \xintloop, and \par rather than \endgraf; and \par rather than \hfill\break
```

```
\def\xintTypesetEuclideAlgorithm {%
    \unless\ifdefined\xintAssignArray
       \errmessage
        {xintgcd: package xinttools is required for \string\xintTypesetEuclideAlgorithm}%
       \expandafter\xint_gobble_iii
    \fi
    \XINT_TypesetEuclideAlgorithm
}%
\def\XINT_TypesetEuclideAlgorithm #1#2%
{% l'algo remplace #1 et #2 par |#1| et |#2|
  \par
  \begingroup
    \xintAssignArray\xintEuclideAlgorithm {#1}{#2}\to\U
    \edef\A{\U2}\edef\B{\U4}\edef\N{\U1}%
    \setbox 0 \vbox{\halign {$##$\cr \A\cr \B \cr}}%
    \count 255 1
    \xintloop
      \indent\hbox to \wd 0 {\hfil$\U{\numexpr 2*\count255\relax}$}%
      ${} =  \U{\numexpr 2*\count255 + 3\relax}
      \times \U{\numexpr 2*\count255 + 2\relax}
          +  \U{\numexpr 2*\count255 + 4\relax}$%
    \ifnum \count255 < \N
      \par
      \advance \count255 1
    \repeat
```

```
  \endgroup
}%
```

24.8. \xintTypesetBezoutAlgorithm

```
Pour Bezout on a: {N}{A}{0}{1}{D=r(n)}{B}{1}{0}{q1}{r1}{alpha1=q1}{beta1=1}
{q2}{r2}{alpha2}{beta2}....{qN}{rN=0}{alphaN=A/D}{betaN=B/D}
  Donc 4N+8 termes: U1 = N, U2= A, U5=D, U6=B, q1 = U9, qn = U{4n+5}, n au moins 1
rn = U{4n+6}, n au moins -1
alpha(n) = U{4n+7}, n au moins -1
beta(n) = U{4n+8}, n au moins -1
  1.09h uses \xintloop, and \par rather than \endgraf; and no more \parindent0pt
\def\xintTypesetBezoutAlgorithm {%
    \unless\ifdefined\xintAssignArray
       \errmessage
        {xintgcd: package xinttools is required for \string\xintTypesetBezoutAlgorithm}%
       \expandafter\xint_gobble_iii
    \fi
    \XINT_TypesetBezoutAlgorithm
}%
\def\XINT_TypesetBezoutAlgorithm #1#2%
{%
  \par
  \begingroup
    \xintAssignArray\xintBezoutAlgorithm {#1}{#2}\to\BEZ
    \edef\A{\BEZ2}\edef\B{\BEZ6}\edef\N{\BEZ1}% A = |#1|, B = |#2|
    \setbox 0 \vbox{\halign {$##$\cr \A\cr \B \cr}}%
    \count255 1
    \xintloop
      \indent\hbox to \wd 0 {\hfil$\BEZ{4*\count255 - 2}$}%
      ${} =  \BEZ{4*\count255 + 5}
      \times \BEZ{4*\count255 + 2}
          +  \BEZ{4*\count255 + 6}$\hfill\break
      \hbox to \wd 0 {\hfil$\BEZ{4*\count255 +7}$}%
      ${} = \BEZ{4*\count255 + 5}
      \times \BEZ{4*\count255 + 3}
          +  \BEZ{4*\count255 - 1}$\hfill\break
      \hbox to \wd 0 {\hfil$\BEZ{4*\count255 +8}$}%
      ${} =  \BEZ{4*\count255 + 5}
      \times \BEZ{4*\count255 + 4}
          +  \BEZ{4*\count255 }$
      \par
    \ifnum \count255 < \N
    \advance \count255 1
  \repeat
    \edef\U{\BEZ{4*\N + 4}}%
    \edef\V{\BEZ{4*\N + 3}}%
    \edef\D{\BEZ5}%
    \ifodd\N
       $\U\times\A  - \V\times \B = -\D$%
    \else
       $\U\times\A - \V\times\B = \D$%
```

```
    \fi
    \par
  \endgroup
}%
\XINTrestorecatcodesendinput%
```

25. Package xintfrac implementation

Contents

.1 Introduction . . . 386
.2 Catcodes, ε-TEX and reload detection . . . 386
.3 Package identification . . . 387
.4 \XINT_cntSgnFork . . . 387
.5 \xintLen . . . 387
.6 \XINT_outfrac . . . 388
.7 \XINT_infrac . . . 388
.8 \XINT_frac_gen . . . 390
.9 \XINT_factortens . . . 392
.10 \xintEq, \xintNotEq, \xintGt, \xintLt, \xintGtorEq, \xintLtorEq, \xintIsZero, \xintIsNotZero, \xintOdd, \xintEven, \xintifSgn, \xintifCmp, \xintifEq, \xintifGt, \xintifLt, \xintifZero, \xintifNotZero, \xintifOne, \xintifOdd . . . 393
.11 \xintRaw . . . 396
.12 \xintRawBraced . . . 396
.13 \xintiLogTen . . . 396
.14 \xintPRaw . . . 397
.15 \xintSPRaw . . . 397
.16 \xintFracToSci . . . 397
.17 \xintFracToDecimal . . . 398
.18 \xintRawWithZeros . . . 398
.19 \xintDecToString . . . 398
.20 \xintDecToStringREZ . . . 399
.21 \xintFloor, \xintiFloor . . . 399
.22 \xintCeil, \xintiCeil . . . 399
.23 \xintNumerator . . . 400
.24 \xintDenominator . . . 400
.25 \xintTeXFrac . . . 400
.26 \xintTeXsignedFrac . . . 402
.27 \xintTeXFromSci . . . 402
.28 \xintTeXOver . . . 403
.29 \xintTeXsignedOver . . . 403
.30 \xintREZ . . . 404
.31 \xintE . . . 405
.32 \xintIrr, \xintPIrr . . . 405
.33 \xintifInt . . . 407
.34 \xintIsInt . . . 407
.35 \xintJrr . . . 407
.36 \xintTFrac . . . 409
.37 \xintTrunc, \xintiTrunc . . . 409
.38 \xintTTrunc . . . 410
.39 \xintNum, \xintnum . . . 410
.40 \xintRound, \xintiRound . . . 411
.41 \xintXTrunc . . . 411
.42 \xintAdd . . . 411
.43 \xintSub . . . 412
.44 \xintSum . . . 413
.45 \xintMul . . . 413
.46 \xintSqr . . . 414
.47 \xintPow . . . 414
.48 \xintFac . . . 415
.49 \xintBinomial . . . 415
.50 \xintPFactorial . . . 415
.51 \xintPrd . . . 415
.52 \xintDiv . . . 416
.53 \xintDivFloor . . . 416
.54 \xintDivTrunc . . . 416
.55 \xintDivRound . . . 417
.56 \xintModTrunc . . . 417
.57 \xintDivMod . . . 418
.58 \xintMod . . . 419
.59 \xintIsOne . . . 420
.60 \xintGeq . . . 420
.61 \xintMax . . . 421
.62 \xintMaxof . . . 422
.63 \xintMin . . . 422
.64 \xintMinof . . . 423
.65 \xintCmp . . . 423
.66 \xintAbs . . . 425
.67 \xintOpp . . . 425
.68 \xintInv . . . 425
.69 \xintSgn . . . 425
.70 \xintSignBit . . . 426
.71 \xintGCD . . . 426
.72 \xintGCDof . . . 427
.73 \xintLCM . . . 428
.74 \xintLCMof . . . 429
.75 Floating point macros . . . 429
.76 \xintFloat, \xintFloatZero . . . 431
.77 \xintFloatBraced . . . 433
.78 \XINTinFloat, \XINTinFloatS . . . 434
.79 \XINTFloatiLogTen . . . 440
.80 \xintPFloat . . . 440
.81 \xintFloatToDecimal . . . 442
.82 \xintFloatAdd, \XINTinFloatAdd . . . 442
.83 \xintFloatSub, \XINTinFloatSub . . . 443
.84 \xintFloatMul, \XINTinFloatMul . . . 444
.85 \xintFloatSqr, \XINTinFloatSqr . . . 445
.86 \xintFloatDiv, \XINTinFloatDiv . . . 445
.87 \xintFloatPow, \XINTinFloatPow . . . 446
.88 \xintFloatPower, \XINTinFloatPower . . . 450
.89 \xintFloatFac, \XINTFloatFac . . . 453
.90 \xintFloatPFactorial, \XINTinFloatPFactorial . . . 454
.91 \xintFloatBinomial, \XINTinFloatBinomial . . . 455

.92 \xintFloatSqrt, \XINTinFloatSqrt . . 456
.93 \xintFloatE 457
.94 \xintifFloatInt 458
.95 \xintFloatIsInt 458
.96 \xintFloatIntType 459

25.1. Introduction

The commenting is currently (2026/11/26) very sparse.

25.2. Catcodes, ε-TEX and reload detection

The code for reload detection was initially copied from Heiko Oberdiek's packages, then modified. The method for catcodes was also initially directly inspired by these packages.

```
\begingroup\catcode61\catcode48\catcode32=10\relax%
  \catcode13=5    % ^^M
  \endlinechar=13 %
  \catcode123=1   % {
  \catcode125=2   % }
  \catcode64=11   % @
  \catcode44=12   % ,
  \catcode46=12   % .
  \catcode58=12   % :
  \catcode94=7    % ^
  \catcode35=6    % #
  \def\odef#1{\expandafter\def\expandafter#1\expandafter}%
  \def\empty{}\def\space{ }\newlinechar10
  \def\z{\endgroup}%
  \def\odef#1{\expandafter\def\expandafter#1\expandafter}%
  \expandafter\let\expandafter\x\csname ver@xintfrac.sty\endcsname
  \expandafter\let\expandafter\c\csname ver@calxfp.sty\endcsname
  \expandafter\let\expandafter\w\csname ver@xint.sty\endcsname
  \expandafter\ifx\csname numexpr\endcsname\relax
    \expandafter\ifx\csname PackageWarningNoLine\endcsname\relax
      \immediate\write128{^^JPackage xint Warning:^^J%
                          \space\space\space\space
                          \numexpr not available, aborting input.^^J}%
    \else
      \PackageWarningNoLine{xint}{\numexpr not available, aborting input}%
    \fi
    \def\z{\endgroup\endinput}%
  \else
    \ifx\x\relax   % plain-TeX, first loading of xintfrac.sty
      \ifx\w\relax % but xint.sty not yet loaded.
         \odef\z{\z\input xint.sty\relax}%
      \fi
      \ifx\c\relax % but calxfp.sty not yet loaded.
         \odef\z{\z\input calxfp.sty\relax}%
      \fi
    \else
      \ifx\x\empty % LaTeX, first loading,
      % variable is initialized, but \ProvidesPackage not yet seen
          \ifx\w\relax % xint.sty not yet loaded.
            \odef\z{\z\RequirePackage{xint}}%
          \fi
```

```
          \ifx\c\relax % calxfp.sty not yet loaded.
            \odef\z{\z\RequirePackage{calxfp}}%
          \fi
      \else
        \def\z{\endgroup\endinput}% xintfrac already loaded.
      \fi
    \fi
  \fi
\z%
\XINTsetupcatcodes% defined in xintkernel.sty
```

25.3. Package identification

```
\XINT_providespackage
\ProvidesPackage{xintfrac}%
  [2026/11/26 v2.0a2 Operations on fractions (JFB)]%
```

25.4. \XINT_cntSgnFork

1.09i. Used internally, #1 must expand to \m@ne, \z@, or \@ne or equivalent. \XINT_cntSgnFork does not insert a romannumeral stopper.

```
\def\XINT_cntSgnFork #1%
{%
    \ifcase #1\expandafter\xint_secondofthree
            \or\expandafter\xint_thirdofthree
          \else\expandafter\xint_firstofthree
    \fi
}%
```

25.5. \xintLen

The used formula is disputable, the idea is that A/1 and A should have same length. Venerable code rewritten for 1.2i, following updates to \xintLength in xintkernel.sty. And sadly, I forgot on this occasion that this macro is not supposed to count the sign... Fixed in 1.2k.

```
\def\xintLen {\romannumeral0\xintlen }%
\def\xintlen #1%
{%
    \expandafter\XINT_flen\romannumeral0\XINT_infrac {#1}%
}%
\def\XINT_flen#1{\def\XINT_flen ##1##2##3%
{%
    \expandafter#1%
    \the\numexpr \XINT_abs##1+%
    \XINT_len_fork ##2##3\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:
      \xint_c_viii\xint_c_vii\xint_c_vi\xint_c_v
      \xint_c_iv\xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye-\xint_c_i
    \relax
}}\XINT_flen{ }%
```

25.6. \XINT_outfrac

Modified at 1.06b (2013/05/14). 1.06b version now outputs 0/1[0] and not 0[0] in case of zero. More generally all macros have been checked in xintfrac, xintseries, xintcfrac, to make sure the output format for fractions was always A/B[n]. (except \xintIrr, \xintJrr, \xintRawWithZeros). Months later (2014/10/22): perhaps I should document what this macro does before I forget? from {e}{N}{D} it outputs N/D[e], checking in passing if D=0 or if N=0. It also makes sure D is not < 0. I am not sure but I don't think there is any place in the code which could call \XINT_outfrac with a D < 0, but I should check.

```
\def\XINT_outfrac #1#2#3%
{%
    \ifcase\XINT_cntSgn #3\xint:
        \expandafter \XINT_outfrac_divisionbyzero
    \or
        \expandafter \XINT_outfrac_P
    \else
        \expandafter \XINT_outfrac_N
    \fi
    {#2}{#3}[#1]%
}%
\def\XINT_outfrac_divisionbyzero #1#2[#3]%
{%
    \XINT_legacysignalcondition{DivisionByZero}{Division by zero: #1/#2.}{}{ 0/1[0]}%
}%
\def\XINT_outfrac_P#1{%
\def\XINT_outfrac_P ##1##2%
   {\if0\XINT_Sgn ##1\xint:\expandafter\XINT_outfrac_Zero\fi#1##1/##2}%
}\XINT_outfrac_P{ }%
\def\XINT_outfrac_Zero #1[#2]{ 0/1[0]}%
\def\XINT_outfrac_N #1#2%
{%
    \expandafter\XINT_outfrac_N_a\expandafter
    {\romannumeral0\XINT_opp #2}{\romannumeral0\XINT_opp #1}%
}%
\def\XINT_outfrac_N_a #1#2%
{%
    \expandafter\XINT_outfrac_P\expandafter {#2}{#1}%
}%
```

25.7. \XINT_infrac

Added at 1.03 (2013/04/14). Parses fraction, scientific notation, etc... and produces {n}{A}{B} corresponding to A/B times 10^n. No reduction to smallest terms.

Modified at 1.07 (2013/05/25). Extended in 1.07 to accept scientific notation on input. With lowercase e only. The \xintexpr parser does accept uppercase E also. Ah, by the way, perhaps I should at least say what this macro does? (belated addition 2014/10/22...), before I forget! It prepares the fraction in the internal format {exponent}{Numerator}{Denominator} where Denominator is at least 1.

Modified at 1.2 (2015/10/10). This venerable macro from the very early days has gotten a lifting for release 1.2. There were two kinds of issues:
1) use of \W, \Z, \T delimiters was very poor choice as this could clash with user input,

2) the new \XINT_frac_gen handles macros (possibly empty) in the input as general as \A.\Be\C/\D.\Ee\F. The earlier version would not have expanded the \B or \E: digits after decimal mark were constrained to arise from expansion of the first token. Thus the 1.03 original code would have expanded only \A, \D, \C, and \F for this input.

This reminded me think I should revisit the remaining earlier portions of code, as I was still learning TeX coding when I wrote them.

Also I thought about parsing even faster the A/B[N] input, not expanding B, but this turned out to clash with some established uses in the documentation such as 1/\xintiiSqr{...}[0]. For the implementation, careful here about potential brace removals with parameter patterns such as like #1/#2#3[#4]for example.

While I was at it 1.2 added \numexpr parsing of the N, which earlier was restricted to be only explicit digits. I allowed [] with empty N, but the way I did it in 1.2 with \the\numexpr 0#1 was buggy, as it did not allow #1 to be a \count for example or itself a \numexpr (although such inputs were not previously allowed, I later turned out to use them in the code itself, e.g. the float factorial of version 1.2f). The better way would be \the\numexpr#1+\xint_c_ but 1.2f finally does only \the\numexpr #1 and #1 is not allowed to be empty.

The 1.2 \XINT_frac_gen had two locations with such a problematic \numexpr 0#1 which I replaced for 1.2f with \numexpr#1+\xint_c_.

Regarding calling the macro with an argument A[<expression>], a / inthe expression must be suitably hidden for example in \firstofone type constructs.

Note: when the numerator is found to be zero \XINT_infrac *always* returns {0}{0}{1}. This behaviour must not change because 1.2g \xintFloat and XINTinFloat (for example) rely upon it: if the denominator on output is not 1, then \xintFloat assumes that the numerator is not zero.

As described in the manual, if the input contains a (final) [N] part, it is assumed that it is in the shape A[N] or A/B[N] with A (and B) not containing neither decimal mark nor scientific part, moreover B must be positive and A have at most one minus sign (and no plus sign). Else there will be errors, for example -0/2[0] would not be recognized as being zero at this stage and this could cause issues afterwards. When there is no ending [N] part, both numerator and denominator will be parsed for the more general format allowing decimal digits and scientific part and possibly multiple leading signs.

Modified at 1.2l (2017/07/26). 1.2l fixes frailty of \XINT_infrac (hence basically of all xintfrac macros) respective to non terminated \numexpr input: \xintRaw{\the\numexpr1} for example. The issue was that \numexpr sees the / and expands what's next. But even \numexpr 1// for example creates an error, and to my mind this is a defect of \numexpr. It should be able to trace back and see that / was used as delimiter not as operator. Anyway, I thus fixed this problem belatedly here regarding \XINT_infrac.

Modified at 1.4l (2022/05/29). Deprecate venerable \XINT_inFrac, whose new name is \xintRawBraced. Should have been removed then.

Modified at 1.4n (2025/09/05). Remove \XINT_inFrac.

```
\def\XINT_infrac #1% the core xintfrac in-parser; triggered by \romannumeral0
{%
    \expandafter\XINT_infrac_fork\romannumeral`&&@#1\xint:/\XINT_W[\XINT_W\XINT_T
}%
\def\XINT_infrac_fork #1[#2%
{%
    \xint_UDXINTWfork
      #2\XINT_frac_gen          % input has no brackets [N]
      \XINT_W\XINT_infrac_res_a % there is some [N], must be strict A[N] or A/B[N] input
    \krof
    #1[#2%
}%
\def\XINT_infrac_res_a #1%
```

```
{%
    \xint_gob_til_zero #1\XINT_infrac_res_zero 0\XINT_infrac_res_b #1%
}%
```

Note that input exponent is here ignored and forced to be zero.

```
\def\XINT_infrac_res_zero 0\XINT_infrac_res_b #1\XINT_T {{0}{0}{1}}%
\def\XINT_infrac_res_b #1/#2%
{%
    \xint_UDXINTWfork
     #2\XINT_infrac_res_ca      % it was A[N] input
      \XINT_W\XINT_infrac_res_cb % it was A/B[N] input
    \krof
    #1/#2%
}%
```

An empty [] is not allowed. (this was authorized in 1.2, removed in 1.2f).

```
\def\XINT_infrac_res_ca #1[#2]\xint:/\XINT_W[\XINT_W\XINT_T
   {\expandafter{\the\numexpr #2}{#1}{1}}%
\def\XINT_infrac_res_cb #1/#2[%
   {\expandafter\XINT_infrac_res_cc\romannumeral`&&@#2~#1[}%
\def\XINT_infrac_res_cc #1~#2[#3]\xint:/\XINT_W[\XINT_W\XINT_T
   {\expandafter{\the\numexpr #3}{#2}{#1}}%
```

25.8. \XINT_frac_gen

Modified at 1.07 (2013/05/25). Extended at to recognize and accept scientific notation both at the numerator and (possible) denominator. Only a lowercase e will do here, but uppercase E is possible within an \xintexpr..\relax

Modified at 1.2 (2015/10/10). Completely rewritten. The parsing handles inputs such as \A.\B↪ e\C/\D.\Ee\F where each of \A, \B, \D, and \E may need f-expansion and \C and \F will end up in \numexpr.

Modified at 1.2f (2016/03/12). 1.2f corrects an issue to allow \C and \F to be \count variable (or expressions with \numexpr): 1.2 did a bad \numexpr0#1 which allowed only explicit digits for expanded #1.

```
\def\XINT_frac_gen #1/#2%
{%
    \xint_UDXINTWfork
      #2\XINT_frac_gen_A      % there was no /
      \XINT_W\XINT_frac_gen_B % there was a /
    \krof
    #1/#2%
}%
```

Note that #1 is only expanded so far up to decimal mark or "e".

calx: native values reaching xintfrac (in \xinteval: a nested \xintfloatexpr or \xintiiexpr, a float or ii variable, a float or ii function). Input without [and without /, hence also F, M, I values. \CALX_fga tests the first token (empty input: the e of ee., then \CALX_infr_o restores the original path, as for any other letter). F: \CALX_ftoraw gives A[N]; M: blocks by \CALX_ntoraw, exponent 8e (calxint's \CALX_infr_m); output {N}{A}{1} as \XINT_infrac. I: InvalidOperation, 0.

```
\def\XINT_frac_gen_A #1\xint:/\XINT_W [\XINT_W {\CALX_fga #1ee.\XINT_W }%
\def\CALX_fga #1{\ifnum`#1>57 \CALX_infr #1\fi\XINT_frac_gen_C 0~1!#1}%
\def\CALX_infr #1\fi\XINT_frac_gen_C 0~1!#2ee.\XINT_W\XINT_T
```

```
    {\fi\if I#1\xint_dothis\CALX_infr_i\fi
     \if M#1\xint_dothis\CALX_infr_m\fi
     \if F#1\xint_dothis\CALX_infr_f\fi
     \xint_orthat\CALX_infr_o{#2}}%
\def\CALX_infr_o #1{\XINT_frac_gen_C 0~1!#1ee.\XINT_W\XINT_T}%
\def\CALX_infr_f #1{\expandafter\CALX_infr_a\romannumeral0\CALX_ftoraw{#1}}%
\def\CALX_infr_a #1[#2]{{#2}{#1}{1}}%
\def\CALX_infr_i #1%
    {\XINT_signalcondition{InvalidOperation}{#1 has no xintfrac value}{}%
     {{0}{0}{1}}}%
\def\XINT_frac_gen_B #1/#2\xint:/\XINT_W[%\XINT_W
{%
    \expandafter\XINT_frac_gen_Ba
    \romannumeral`&&@#2ee.\XINT_W\XINT_Z #1ee.%\XINT_W
}%
\def\XINT_frac_gen_Ba #1.#2%
{%
    \xint_UDXINTWfork
      #2\XINT_frac_gen_Bb
      \XINT_W\XINT_frac_gen_Bc
    \krof
    #1.#2%
}%
\def\XINT_frac_gen_Bb #1e#2e#3\XINT_Z
                 {\expandafter\XINT_frac_gen_C\the\numexpr #2+\xint_c_~#1!}%
\def\XINT_frac_gen_Bc #1.#2e%
{%
    \expandafter\XINT_frac_gen_Bd\romannumeral`&&@#2.#1e%
}%
\def\XINT_frac_gen_Bd #1.#2e#3e#4\XINT_Z
{%
    \expandafter\XINT_frac_gen_C\the\numexpr #3-%
    \numexpr\XINT_length_loop
    #1\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:
      \xint_c_viii\xint_c_vii\xint_c_vi\xint_c_v
      \xint_c_iv\xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye
    ~#2#1!%
}%
\def\XINT_frac_gen_C #1!#2.#3%
{%
    \xint_UDXINTWfork
      #3\XINT_frac_gen_Ca
      \XINT_W\XINT_frac_gen_Cb
    \krof
    #1!#2.#3%
}%
\def\XINT_frac_gen_Ca #1~#2!#3e#4e#5\XINT_T
{%
    \expandafter\XINT_frac_gen_F\the\numexpr #4-#1\expandafter
    ~\romannumeral0\expandafter\XINT_num_cleanup\the\numexpr\XINT_num_loop
     #2\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\Z~#3~%
}%
```

```
\def\XINT_frac_gen_Cb #1.#2e%
{%
    \expandafter\XINT_frac_gen_Cc\romannumeral`&&@#2.#1e%
}%
\def\XINT_frac_gen_Cc #1.#2~#3!#4e#5e#6\XINT_T
{%
    \expandafter\XINT_frac_gen_F\the\numexpr #5-#2-%
    \numexpr\XINT_length_loop
    #1\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:
      \xint_c_viii\xint_c_vii\xint_c_vi\xint_c_v
      \xint_c_iv\xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye
    \relax\expandafter~%
    \romannumeral0\expandafter\XINT_num_cleanup\the\numexpr\XINT_num_loop
    #3\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\Z
    ~#4#1~%
}%
\def\XINT_frac_gen_F #1~#2%
{%
    \xint_UDzerominusfork
      #2-\XINT_frac_gen_Gdivbyzero
      0#2{\XINT_frac_gen_G  -{}}%
       0-{\XINT_frac_gen_G {}#2}%
    \krof  #1~%
}%
\def\XINT_frac_gen_Gdivbyzero #1~~#2~%
{%
   \expandafter\XINT_frac_gen_Gdivbyzero_a
   \romannumeral0\expandafter\XINT_num_cleanup\the\numexpr\XINT_num_loop
   #2\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\Z~#1~%
}%
\def\XINT_frac_gen_Gdivbyzero_a #1~#2~%
{%
    \XINT_legacysignalcondition{DivisionByZero}{Division by zero: #1/0.}{}{{#2}{#1}{0}}%
}%
\def\XINT_frac_gen_G #1#2#3~#4~#5~%
{%
    \expandafter\XINT_frac_gen_Ga
    \romannumeral0\expandafter\XINT_num_cleanup\the\numexpr\XINT_num_loop
    #1#5\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\Z~#3~{#2#4}%
}%
\def\XINT_frac_gen_Ga #1#2~#3~%
{%
    \xint_gob_til_zero #1\XINT_frac_gen_zero 0%
    {#3}{#1#2}%
}%
\def\XINT_frac_gen_zero 0#1#2#3{{0}{0}{1}}%
```

25.9. \XINT_factortens

This is the core macro for \xintREZ. To be used as \romannumeral0\XINT_factortens{...}. Output is A.N. (formerly {A}{N}) where A is the integer stripped from trailing zeroes and N is the number of removed zeroes. Only for positive strict integers!

Modified at 1.3a (2018/03/07). Completely rewritten at 1.3a to replace a double \xintReverseOrder by a direct \numexpr governed expansion to the end and back, à la 1.2. I should comment more... and perhaps improve again in future.
Testing shows significant gain at 100 digits or more.

Modified at 1.4p (2026/03/01). \XINT_factortens_cc had an unnecessary \if#90 test, the false branch was never taken.

```
\def\XINT_factortens #1{\expandafter\XINT_factortens_z
                        \romannumeral0\XINT_factortens_a#1%
                        \XINT_factortens_b123456789.}%
\def\XINT_factortens_z.\XINT_factortens_y{ }%
\def\XINT_factortens_a #1#2#3#4#5#6#7#8#9%
   {\expandafter\XINT_factortens_x
    \the\numexpr 1#1#2#3#4#5#6#7#8#9\XINT_factortens_a}%
\def\XINT_factortens_b#1\XINT_factortens_a#2#3.%
   {.\XINT_factortens_cc 000000000-#2.}%
\def\XINT_factortens_x1#1.#2{#2#1}%
\def\XINT_factortens_y{.\XINT_factortens_y}%
\def\XINT_factortens_cc #1#2#3#4#5#6#7#8#9%
   {\expandafter\XINT_factortens_d\the\numexpr #8#7#6#5#4#3#2#1\relax
    \xint_c_i 2345678.}%
\def\XINT_factortens_yy #1#2.{.\XINT_factortens_y#1.0.}%
\def\XINT_factortens_c #1#2#3#4#5#6#7#8#9%
   {\if#90\xint_dothis
     {\expandafter\XINT_factortens_d\the\numexpr #8#7#6#5#4#3#2#1\relax
      \xint_c_i 2345678.}\fi
    \xint_orthat{.\XINT_factortens_y #1#2#3#4#5#6#7#8#9.}}%
\def\XINT_factortens_d #1#2#3#4#5#6#7#8#9%
   {\if#10\expandafter\XINT_factortens_e\fi
    \XINT_factortens_f #9#9#8#7#6#5#4#3#2#1.}%
\def\XINT_factortens_f #1#2\xint_c_i#3.#4.#5.%
   {\expandafter\XINT_factortens_g\the\numexpr#1+#5.#3.}%
\def\XINT_factortens_g #1.#2.{.\XINT_factortens_y#2.#1.}%
\def\XINT_factortens_e #1..#2.%
   {\expandafter.\expandafter\XINT_factortens_c
    \the\numexpr\xint_c_ix+#2.}%
```

25.10. \xintEq, \xintNotEq, \xintGt, \xintLt, \xintGtorEq, \xintLtorEq, \xintIsZero, \xintIsNotZero, \xintOdd, \xintEven, \xintifSgn, \xintifCmp, \xintifEq, \xintifGt, \xintifLt, \xintifZero, \xintifNotZero, \xintifOne, \xintifOdd

Moved here at 1.3. Formerly these macros were already defined in xint.sty or even xintcore.sty. They are slim wrappers of macros defined elsewhere in xintfrac.

```
\def\xintEq   {\romannumeral0\xinteq }%
\def\xinteq   #1#2{\xintifeq{#1}{#2}{1}{0}}%
\def\xintNotEq#1#2{\romannumeral0\xintifeq {#1}{#2}{0}{1}}%
\def\xintGt {\romannumeral0\xintgt }%
\def\xintgt #1#2{\xintifgt{#1}{#2}{1}{0}}%
\def\xintLt   {\romannumeral0\xintlt }%
\def\xintlt #1#2{\xintiflt{#1}{#2}{1}{0}}%
\def\xintGtorEq #1#2{\romannumeral0\xintiflt {#1}{#2}{0}{1}}%
\def\xintLtorEq #1#2{\romannumeral0\xintifgt {#1}{#2}{0}{1}}%
```

```
\def\xintIsZero    {\romannumeral0\xintiszero }%
\def\xintiszero #1{\if0\xintSgn{#1}\xint_afterfi{ 1}\else\xint_afterfi{ 0}\fi}%
\def\xintIsNotZero{\romannumeral0\xintisnotzero }%
\def\xintisnotzero
           #1{\if0\xintSgn{#1}\xint_afterfi{ 0}\else\xint_afterfi{ 1}\fi}%
\def\xintOdd       {\romannumeral0\xintodd }%
\def\xintodd #1%
{%
    \ifodd\xintLDg{\xintNum{#1}} %<- intentional space
        \xint_afterfi{ 1}%
    \else
        \xint_afterfi{ 0}%
    \fi
}%
\def\xintEven      {\romannumeral0\xinteven }%
\def\xinteven #1%
{%
    \ifodd\xintLDg{\xintNum{#1}} %<- intentional space
        \xint_afterfi{ 0}%
    \else
        \xint_afterfi{ 1}%
    \fi
}%
\def\xintifSgn{\romannumeral0\xintifsgn }%
\def\xintifsgn #1%
{%
    \ifcase \xintSgn{#1}
                \expandafter\xint_stop_atsecondofthree
              \or\expandafter\xint_stop_atthirdofthree
            \else\expandafter\xint_stop_atfirstofthree
    \fi
}%
\def\xintifCmp{\romannumeral0\xintifcmp }%
\def\xintifcmp #1#2%
{%
    \ifcase\xintCmp {#1}{#2}
                \expandafter\xint_stop_atsecondofthree
              \or\expandafter\xint_stop_atthirdofthree
            \else\expandafter\xint_stop_atfirstofthree
    \fi
}%
\def\xintifEq {\romannumeral0\xintifeq }%
\def\xintifeq #1#2%
{%
    \if0\xintCmp{#1}{#2}%
                \expandafter\xint_stop_atfirstoftwo
            \else\expandafter\xint_stop_atsecondoftwo
    \fi
}%
\def\xintifGt {\romannumeral0\xintifgt }%
\def\xintifgt #1#2%
{%
```

```
    \if1\xintCmp{#1}{#2}%
                \expandafter\xint_stop_atfirstoftwo
          \else\expandafter\xint_stop_atsecondoftwo
    \fi
}%
\def\xintifLt {\romannumeral0\xintiflt }%
\def\xintiflt #1#2%
{%
    \ifnum\xintCmp{#1}{#2}<\xint_c_
          \expandafter\xint_stop_atfirstoftwo
    \else \expandafter\xint_stop_atsecondoftwo
    \fi
}%
\def\xintifZero   {\romannumeral0\xintifzero }%
\def\xintifzero #1%
{%
    \if0\xintSgn{#1}%
       \expandafter\xint_stop_atfirstoftwo
    \else
       \expandafter\xint_stop_atsecondoftwo
    \fi
}%
\def\xintifNotZero{\romannumeral0\xintifnotzero }%
\def\xintifnotzero #1%
{%
    \if0\xintSgn{#1}%
       \expandafter\xint_stop_atsecondoftwo
    \else
       \expandafter\xint_stop_atfirstoftwo
    \fi
}%
\def\xintifOne {\romannumeral0\xintifone }%
\def\xintifone #1%
{%
    \if1\xintIsOne{#1}%
       \expandafter\xint_stop_atfirstoftwo
    \else
       \expandafter\xint_stop_atsecondoftwo
    \fi
}%
\def\xintifOdd {\romannumeral0\xintifodd }%
\def\xintifodd #1%
{%
    \if\xintOdd{#1}1%
       \expandafter\xint_stop_atfirstoftwo
    \else
       \expandafter\xint_stop_atsecondoftwo
    \fi
}%
```

25.11. \xintRaw

Added at 1.07 (2013/05/25). 1.07: this macro simply prints in a user readable form the fraction after its initial scanning. Useful when put inside braces in an \xintexpr, when the input is not yet in the A/B[n] form.

```
\def\xintRaw {\romannumeral0\xintraw }%
\def\xintraw
{%
    \expandafter\XINT_raw\romannumeral0\XINT_infrac
}%
\def\XINT_raw #1#2#3{ #2/#3[#1]}%
```

25.12. \xintRawBraced

Added at 1.4l (2022/05/29). User level interface to core \romannumeral0\XINT_infrac. Replaces \XINT_inFrac which was defined but nowhere used by the xint packages.

```
\def\xintRawBraced {\romannumeral0\xintrawbraced }%
\let\xintrawbraced \XINT_infrac
```

25.13. \xintiLogTen

Added at 1.3e (2019/04/05). The exponent a, such that 10^a<= abs(x) < 10^(a+1). No rounding done on x, handled as an exact fraction.

```
\def\xintiLogTen {\the\numexpr\xintilogten}%
\def\xintilogten
{%
    \expandafter\XINT_ilogten\romannumeral0\xintraw
}%
\def\XINT_ilogten #1%
{%
    \xint_UDzerominusfork
      0#1\XINT_ilogten_p
      #1-\XINT_ilogten_z
      0-{\XINT_ilogten_p#1}%
    \krof
}%
\def\XINT_ilogten_z #1[#2]{-"7FFF8000\relax}%
\def\XINT_ilogten_p #1/#2[#3]%
{%
    #3+\expandafter\XINT_ilogten_a
       \the\numexpr\xintLength{#1}\expandafter.\the\numexpr\xintLength{#2}.#1.#2.%
}%
\def\XINT_ilogten_a #1.#2.%
{%
    #1-#2\ifnum#1>#2
      \expandafter\XINT_ilogten_aa
    \else
      \expandafter\XINT_ilogten_ab
    \fi #1.#2.%
}%
\def\XINT_ilogten_aa #1.#2.#3.#4.%
{%
```

```
    \xintiiifLt{#3}{\XINT_dsx_addzerosnofuss{#1-#2}#4;}{-1}{}\relax
}%
\def\XINT_ilogten_ab #1.#2.#3.#4.%
{%
    \xintiiifLt{\XINT_dsx_addzerosnofuss{#2-#1}#3;}{#4}{-1}{}\relax
}%
```

25.14. \xintPRaw

Added at 1.09b (2013/10/03).

```
\def\xintPRaw {\romannumeral0\xintpraw }%
\def\xintpraw
{%
    \expandafter\XINT_praw\romannumeral0\XINT_infrac
}%
\def\XINT_praw #1%
{%
    \ifnum #1=\xint_c_ \expandafter\XINT_praw_a\fi \XINT_praw_A {#1}%
}%
\def\XINT_praw_A #1#2#3%
{%
    \if\XINT_isOne{#3}1\expandafter\xint_firstoftwo
                  \else\expandafter\xint_secondoftwo
    \fi { #2[#1]}{ #2/#3[#1]}%
}%
\def\XINT_praw_a\XINT_praw_A #1#2#3%
{%
    \if\XINT_isOne{#3}1\expandafter\xint_firstoftwo
                  \else\expandafter\xint_secondoftwo
    \fi { #2}{ #2/#3}%
}%
```

25.15. \xintSPRaw

This private macro was for internal usage by \xinttheexpr. It got moved here at 1.4 and is not used anymore by the package.

It checks if input has a [N] part, if yes uses \xintPRaw, else simply lets the input pass through as is.

```
\def\xintSPRaw    {\romannumeral0\xintspraw }%
\def\xintspraw  #1{\expandafter\XINT_spraw\romannumeral`&&@#1[\W]}%
\def\XINT_spraw #1[#2#3]{\xint_gob_til_W #2\XINT_spraw_a\W\XINT_spraw_p #1[#2#3]}%
\def\XINT_spraw_a\W\XINT_spraw_p #1[\W]{ #1}%
\def\XINT_spraw_p #1[\W]{\xintpraw {#1}}%
```

25.16. \xintFracToSci

Added at 1.4l (2022/05/29). The macro with this name which was added here at 1.4 then had various changes and finally was moved to xintexpr at 1.4k is now called there \xint_FracToSci_x and is private. The present macro is public and behaves like the other xintfrac macros: f-expandable and accepts general input. Its output is exactly the same as \xint_FracToSci_x for same inputs, with the exception of the empty input which \xintFracToSci will output as 0 but

\xint_FracToSci_x as empty. But the latter is not used by \xinteval for an empty leaf as it employs then \xintexprEmptyItem.

```
\def\xintFracToSci{\romannumeral0\xintfractosci}%
\def\xintfractosci#1{\expandafter\XINT_fractosci\romannumeral0\xintraw{#1}}%
\def\XINT_fractosci#1#2/#3[#4]{\expanded{ %
    \ifnum#4=\xint_c_ #1#2\else
      \romannumeral0\expandafter\XINT_pfloat_a_fork\romannumeral0\xintrez{#1#2[#4]}%
    \fi
    \if\XINT_isOne{#3}1\else\if#10\else/#3\fi\fi}%
}%
```

25.17. \xintFracToDecimal

Added at 1.4l (2022/05/29). The macro with this name which was added at 1.4k to xintexpr has been removed. The public variant here behaves like the other xintfrac macros: f-expandable and accepts general input.

```
\def\xintFracToDecimal{\romannumeral0\xintfractodecimal}%
\def\xintfractodecimal#1{\expandafter\XINT_fractodecimal\romannumeral0\xintraw{#1}}%
\def\XINT_fractodecimal #1#2/#3[#4]{\expanded{ %
    \ifnum#4=\xint_c_ #1#2\else
      \romannumeral0\expandafter\XINT_dectostr\romannumeral0\xintrez{#1#2[#4]}%
    \fi
    \if\XINT_isOne{#3}1\else\if#10\else/#3\fi\fi}%
}%
```

25.18. \xintRawWithZeros

This was called \xintRaw in versions earlier than 1.07

```
\def\xintRawWithZeros {\romannumeral0\xintrawwithzeros }%
\def\xintrawwithzeros
{%
    \expandafter\XINT_rawz_fork\romannumeral0\XINT_infrac
}%
\def\XINT_rawz_fork #1%
{%
    \ifnum#1<\xint_c_
      \expandafter\XINT_rawz_Ba
    \else
      \expandafter\XINT_rawz_A
    \fi
    #1.%
}%
\def\XINT_rawz_A  #1.#2#3{\XINT_dsx_addzeros{#1}#2;/#3}%
\def\XINT_rawz_Ba -#1.#2#3{\expandafter\XINT_rawz_Bb
    \expandafter{\romannumeral0\XINT_dsx_addzeros{#1}#3;}{#2}}%
\def\XINT_rawz_Bb #1#2{ #2/#1}%
```

25.19. \xintDecToString

Added at 1.3 (2018/03/01). This is a backport from polexpr 0.4. It is definitely not in final form, consider it to be an unstable macro.

```
\def\xintDecToString{\romannumeral0\xintdectostring}%
\def\xintdectostring#1{\expandafter\XINT_dectostr\romannumeral0\xintraw{#1}}%
\def\XINT_dectostr #1/#2[#3]{\xintiiifZero {#1}%
       \XINT_dectostr_z
       {\if1\XINT_isOne{#2}\expandafter\XINT_dectostr_a
                      \else\expandafter\XINT_dectostr_b
        \fi}%
  #1/#2[#3]%
}%
\def\XINT_dectostr_z#1[#2]{ 0}%
\def\XINT_dectostr_a#1/#2[#3]{%
    \ifnum#3<\xint_c_\xint_dothis{\xinttrunc{-#3}{#1[#3]}}\fi
    \xint_orthat{\xintiie{#1}{#3}}%
}%
\def\XINT_dectostr_b#1/#2[#3]{% just to handle this somehow
    \ifnum#3<\xint_c_\xint_dothis{\xinttrunc{-#3}{#1[#3]}/#2}\fi
    \xint_orthat{\xintiie{#1}{#3}/#2}%
}%
```

25.20. \xintDecToStringREZ

Added at 1.4e (2021/05/05). And I took this opportunity to improve documentation in manual.

```
\def\xintDecToStringREZ{\romannumeral0\xintdectostringrez}%
\def\xintdectostringrez#1{\expandafter\XINT_dectostr\romannumeral0\xintrez{#1}}%
```

25.21. \xintFloor, \xintiFloor

Added at 1.09a (2013/09/24). By calxfp's \CALX_fintp on the parts {N}{A}{B} of \XINT_infrac: s sign, T digits of |trunc(X)|, z = 1 if X is not an integer. floor(X) = T (s = +) or -(T + z); ceil(X) = T + z (s = +) or -T. +1 by xintcore's \xintinc. \xintFloor, \xintCeil append /1[0].

```
\def\xintFloor {\romannumeral0\xintfloor }%
\def\xintfloor #1{\expandafter\XINT_fl_q\romannumeral0\xintifloor{#1}\xint:}%
\def\XINT_fl_q #1\xint:{ #1/1[0]}%
\def\xintiFloor {\romannumeral0\xintifloor }%
\def\xintifloor #1%
   {\expandafter\XINT_ifloor\romannumeral0\XINT_infrac{#1}\XINT_ifloor_f}%
\def\XINT_ifloor #1#2#3#4{\expandafter#4\romannumeral0\CALX_fintp{#2}{#3}{#1}}%
\def\XINT_ifloor_f #1#2.#3#4%
   {\if-#1\xint_dothis{\XINT_ifloor_m#4{#2}}\fi\xint_orthat{ #2}}%
\def\XINT_ifloor_m #1#2%
{%
    \if1#1\xint_dothis{\expandafter-\romannumeral0\xintinc{#2}}\fi
    \xint_orthat{ -#2}%
}%
```

25.22. \xintCeil, \xintiCeil

Added at 1.09a (2013/09/24).

```
\def\xintCeil {\romannumeral0\xintceil }%
\def\xintceil #1{\expandafter\XINT_fl_q\romannumeral0\xinticeil{#1}\xint:}%
\def\xintiCeil {\romannumeral0\xinticeil }%
\def\xinticeil #1%
```

```
525    {\expandafter\XINT_ifloor\romannumeral0\XINT_infrac{#1}\XINT_iceil_f}%
526 \def\XINT_iceil_f #1#2.#3#4%
527 {%
528     \if-#1\xint_dothis{\XINT_iceil_m #2\xint:}\fi
529     \if1#4\xint_dothis{\xintinc{#2}}\fi
530     \xint_orthat{ #2}%
531 }%
532 \def\XINT_iceil_m #1#2\xint:{\if0#1\xint_dothis{ 0}\fi\xint_orthat{ -#1#2}}%
```

25.23. \xintNumerator

```
533 \def\xintNumerator {\romannumeral0\xintnumerator }%
534 \def\xintnumerator
535 {%
536     \expandafter\XINT_numer\romannumeral0\XINT_infrac
537 }%
538 \def\XINT_numer #1%
539 {%
540     \ifcase\XINT_cntSgn #1\xint:
541       \expandafter\XINT_numer_B
542     \or
543       \expandafter\XINT_numer_A
544     \else
545       \expandafter\XINT_numer_B
546     \fi
547     {#1}%
548 }%
549 \def\XINT_numer_A #1#2#3{\XINT_dsx_addzeros{#1}#2;}%
550 \def\XINT_numer_B #1#2#3{ #2}%
```

25.24. \xintDenominator

```
551 \def\xintDenominator {\romannumeral0\xintdenominator }%
552 \def\xintdenominator
553 {%
554     \expandafter\XINT_denom_fork\romannumeral0\XINT_infrac
555 }%
556 \def\XINT_denom_fork #1%
557 {%
558     \ifnum#1<\xint_c_
559       \expandafter\XINT_denom_B
560     \else
561       \expandafter\XINT_denom_A
562     \fi
563     #1.%
564 }%
565 \def\XINT_denom_A #1.#2#3{ #3}%
566 \def\XINT_denom_B -#1.#2#3{\XINT_dsx_addzeros{#1}#3;}%
```

25.25. \xintTeXFrac

Added at 1.03 (2013/04/14). Useless typesetting macro.

Modified at 1.4g (2021/05/25). Renamed from \xintFrac.

Modified at 1.4m (2022/06/10). The old name now raises an error, not a warning.

```
\ifdefined\PackageWarning
\def\xintfracTeXDeprecation#1#2{%
\PackageWarning{xintfrac}{\string#1 is deprecated. Use \string#2\MessageBreak
                          to suppress this warning}#2%
}%
\else
\edef\xintfracTeXDeprecation#1#2{{\newlinechar10
\immediate\noexpand\write128{&&JPackage xintfrac Warning: \noexpand\string#1 is
  deprecated. Use \noexpand\string#2&&J%
(xintfrac)\xintReplicate{16}{ }to suppress this warning
on input line \noexpand\the\inputlineno.&&J}}#2%
}%
\fi
\ifdefined\PackageError
\def\xintfracTeXError#1#2{%
\PackageError{xintfrac}{\string#1 has been removed.\MessageBreak
    Use \string#2 to suppress this error}%
    {I will fix it for now if you hit the `Return' key.}#2%
}%
\else
\edef\xintfracTeXError#1#2{{\newlinechar10
\errhelp{I will fix it for now if you hit the `Return' key.}%
\errmessage{Package xintfrac Error: \noexpand\string#1 has been removed.&&J%
(xintfrac)\xintReplicate{16}{ }Use \noexpand\string#2 to suppress this error}}#2%
}%
\fi
\def\xintFrac   {\xintfracTeXError\xintFrac\xintTeXFrac}%
\def\xintTeXFrac{\romannumeral0\xintfrac }%
\def\xintfrac #1%
{%
    \expandafter\XINT_fracfrac_A\romannumeral0\XINT_infrac {#1}%
}%
\def\XINT_fracfrac_A #1{\XINT_fracfrac_B #1\Z }%
\catcode`^=7
\def\XINT_fracfrac_B #1#2\Z
{%
    \xint_gob_til_zero #1\XINT_fracfrac_C 0\XINT_fracfrac_D {10^{#1#2}}%
}%
\def\XINT_fracfrac_C 0\XINT_fracfrac_D #1#2#3%
{%
    \if1\XINT_isOne {#3}%
        \xint_afterfi {\expandafter\xint_stop_atfirstoftwo\xint_gobble_ii }%
    \fi
    \space
    \frac {#2}{#3}%
}%
\def\XINT_fracfrac_D #1#2#3%
{%
    \if1\XINT_isOne {#3}\XINT_fracfrac_E\fi
    \space
```

```
    \frac {#2}{#3}#1%
}%
\def\XINT_fracfrac_E \fi\space\frac #1#2{\fi \space #1\cdot }%
```

25.26. \xintTeXsignedFrac

Modified at 1.4g (2021/05/25). Renamed from \xintSignedFrac.

Modified at 1.4m (2022/06/10). The old name now raises an error, not a warning.

```
\def\xintSignedFrac   {\xintfracTeXError\xintSignedFrac\xintTeXsignedFrac}%
\def\xintTeXsignedFrac{\romannumeral0\xintsignedfrac }%
\def\xintsignedfrac #1%
{%
    \expandafter\XINT_sgnfrac_a\romannumeral0\XINT_infrac {#1}%
}%
\def\XINT_sgnfrac_a #1#2%
{%
    \XINT_sgnfrac_b #2\Z {#1}%
}%
\def\XINT_sgnfrac_b #1%
{%
    \xint_UDsignfork
      #1\XINT_sgnfrac_N
       -{\XINT_sgnfrac_P #1}%
    \krof
}%
\def\XINT_sgnfrac_P #1\Z #2%
{%
    \XINT_fracfrac_A {#2}{#1}%
}%
\def\XINT_sgnfrac_N
{%
    \expandafter-\romannumeral0\XINT_sgnfrac_P
}%
```

25.27. \xintTeXFromSci

Added at 1.4g (2021/05/25). The main problem is how to name this and related macros.
I use \expanded here, as \xintFracToSci is not f-expandable.
Some complications as I want this to be usable on output of \xintFracToSci hence need to handle the case of a /B. After some hesitations I ended with the following which looks reasonable:
- if no scientific part, use \frac (or \over) for A/B
- if scientific part, postfix /B as \cdot B^{-1}

Modified at 1.4l (2022/05/29). Suppress external \expanded. Keep internal one.
Rename \xintTeXFromSci from \xintTeXfromSci. Keep deprecated old name for the moment.
Add \xintTeXFromScifracmacro. Make it \protected.
Nota bene: catcode of ^ is normal one here (else nothing would work).

```
\def\xintTeXfromSci{\xintfracTeXDeprecation\xintTeXfromSci\xintTeXFromSci}%
\def\xintTeXFromSci#1%
{%
    \expandafter\XINT_texfromsci\expanded{#1}/\relax/\xint:
}%
```

```
\def\XINT_texfromsci #1/#2#3/#4\xint:
{%
    \XINT_texfromsci_a #1e\relax e\xint:
    {\ifx\relax#2\xint_dothis\xint_firstofone\fi
     \xint_orthat{\xintTeXFromScifracmacro{#2#3}}}%
    {\unless\ifx\relax#2\cdot{#2#3}^{-1}\fi}%
}%
\def\XINT_texfromsci_a #1e#2#3e#4\xint:#5#6%
{%
    \ifx\relax#2#5{#1}\else#1\cdot10^{#2#3}#6\fi
}%
\ifdefined\frac
  \protected\def\xintTeXFromScifracmacro#1#2{\frac{#2}{#1}}%
\else
  \protected\def\xintTeXFromScifracmacro#1#2{{#2\over#1}}%
\fi
```

25.28. \xintTeXOver

Modified at 1.4g (2021/05/25). Renamed from \xintFwOver.

Modified at 1.4m (2022/06/10). The old name now raises an error, not a warning.

```
\def\xintFwOver {\xintfracTeXError\xintFwOver\xintTeXOver}%
\def\xintTeXOver{\romannumeral0\xintfwover }%
\def\xintfwover #1%
{%
    \expandafter\XINT_fwover_A\romannumeral0\XINT_infrac {#1}%
}%
\def\XINT_fwover_A #1{\XINT_fwover_B #1\Z }%
\def\XINT_fwover_B #1#2\Z
{%
    \xint_gob_til_zero #1\XINT_fwover_C 0\XINT_fwover_D {10^{#1#2}}%
}%
\catcode`^=11
\def\XINT_fwover_C #1#2#3#4#5%
{%
    \if0\XINT_isOne {#5}\xint_afterfi { {#4\over #5}}%
                    \else\xint_afterfi { #4}%
    \fi
}%
\def\XINT_fwover_D #1#2#3%
{%
    \if0\XINT_isOne {#3}\xint_afterfi { {#2\over #3}}%
                    \else\xint_afterfi { #2\cdot }%
    \fi
    #1%
}%
```

25.29. \xintTeXsignedOver

Modified at 1.4g (2021/05/25). Renamed from \xintSignedFwOver.

Modified at 1.4m (2022/06/10). The old name now raises an error, not a warning.

```
\def\xintSignedFwOver {\xintfracTeXError\xintSignedFwOver\xintTeXsignedOver}%
\def\xintTeXsignedOver{\romannumeral0\xintsignedfwover }%
\def\xintsignedfwover #1%
{%
    \expandafter\XINT_sgnfwover_a\romannumeral0\XINT_infrac {#1}%
}%
\def\XINT_sgnfwover_a #1#2%
{%
    \XINT_sgnfwover_b #2\Z {#1}%
}%
\def\XINT_sgnfwover_b #1%
{%
    \xint_UDsignfork
      #1\XINT_sgnfwover_N
       -{\XINT_sgnfwover_P #1}%
    \krof
}%
\def\XINT_sgnfwover_P #1\Z #2%
{%
    \XINT_fwover_A {#2}{#1}%
}%
\def\XINT_sgnfwover_N
{%
    \expandafter-\romannumeral0\XINT_sgnfwover_P
}%
```

25.30. \xintREZ

Removes trailing zeros from A and B and adjust the N in A/B[N].

The macro really doing the job \XINT_factortens was redone at 1.3a. But speed gain really noticeable only beyond about 100 digits.

```
\def\xintREZ {\romannumeral0\xintrez }%
\def\xintrez
{%
    \expandafter\XINT_rez_A\romannumeral0\XINT_infrac
}%
\def\XINT_rez_A #1#2%
{%
    \XINT_rez_AB #2\Z {#1}%
}%
\def\XINT_rez_AB #1%
{%
    \xint_UDzerominusfork
      #1-\XINT_rez_zero
      0#1\XINT_rez_neg
       0-{\XINT_rez_B #1}%
    \krof
}%
\def\XINT_rez_zero #1\Z #2#3{ 0/1[0]}%
\def\XINT_rez_neg {\expandafter-\romannumeral0\XINT_rez_B }%
\def\XINT_rez_B #1\Z
{%
```

```
    \expandafter\XINT_rez_C\romannumeral0\XINT_factortens {#1}%
}%
\def\XINT_rez_C #1.#2.#3#4%
{%
    \expandafter\XINT_rez_D\romannumeral0\XINT_factortens {#4}#3+#2.#1.%
}%
\def\XINT_rez_D #1.#2.#3.%
{%
    \expandafter\XINT_rez_E\the\numexpr #3-#2.#1.%
}%
\def\XINT_rez_E #1.#2.#3.{ #3/#2[#1]}%
```

25.31. \xintE

Added at 1.07 (2013/05/25). The fraction is the first argument contrarily to \xintTrunc and \xintRound.

Modified at 1.1 (2014/10/28). 1.1 modifies and moves \xintiiE to xint.sty.

```
\def\xintE {\romannumeral0\xinte }%
\def\xinte #1%
{%
    \expandafter\XINT_e \romannumeral0\XINT_infrac {#1}%
}%
\def\XINT_e #1#2#3#4%
{%
    \expandafter\XINT_e_end\the\numexpr #1+#4.{#2}{#3}%
}%
\def\XINT_e_end #1.#2#3{ #2/#3[#1]}%
```

25.32. \xintIrr, \xintPIrr

Modified at 1.04 (2013/04/25). fixes a buggy \xintIrr {0}.

Modified at 1.05 (2013/05/01). modifies the initial parsing and post-processing to use \xintrawwithzeros and to more quickly deal with an input denominator equal to 1.

Modified at 1.08 (2013/06/07). this version does not remove a /1 denominator.

Modified at 1.3 (2018/03/01). added \xintPIrr (partial Irr, which ignores the decimal part).

```
\def\xintIrr {\romannumeral0\xintirr }%
\def\xintPIrr{\romannumeral0\xintpirr }%
\def\xintirr #1%
{%
    \expandafter\XINT_irr_start\romannumeral0\xintrawwithzeros {#1}\Z
}%
\def\xintpirr #1%
{%
    \expandafter\XINT_pirr_start\romannumeral0\xintraw{#1}%
}%
\def\XINT_irr_start #1#2/#3\Z
{%
    \if0\XINT_isOne {#3}%
      \xint_afterfi
          {\xint_UDsignfork
               #1\XINT_irr_negative
```

```
                -{\XINT_irr_nonneg #1}%
           \krof}%
    \else
      \xint_afterfi{\XINT_irr_denomisone #1}%
    \fi
    #2\Z {#3}%
}%
\def\XINT_pirr_start #1#2/#3[%
{%
    \if0\XINT_isOne {#3}%
      \xint_afterfi
          {\xint_UDsignfork
               #1\XINT_irr_negative
                -{\XINT_irr_nonneg #1}%
           \krof}%
    \else
      \xint_afterfi{\XINT_irr_denomisone #1}%
    \fi
    #2\Z {#3}[%
}%
\def\XINT_irr_denomisone #1\Z #2{ #1/1}% changed in 1.08
\def\XINT_irr_negative   #1\Z #2{\XINT_irr_D #1\Z #2\Z -}%
\def\XINT_irr_nonneg     #1\Z #2{\XINT_irr_D #1\Z #2\Z \space}%
\def\XINT_irr_D #1#2\Z #3#4\Z
{%
    \xint_UDzerosfork
       #3#1\XINT_irr_indeterminate
       #30\XINT_irr_divisionbyzero
       #10\XINT_irr_zero
        00\XINT_irr_loop_a
    \krof
    {#3#4}{#1#2}{#3#4}{#1#2}%
}%
\def\XINT_irr_indeterminate #1#2#3#4#5%
{%
    \XINT_legacysignalcondition{DivisionUndefined}{0/0 indeterminate fraction.}{}{ 0/1}%
}%
\def\XINT_irr_divisionbyzero #1#2#3#4#5%
{%
    \XINT_legacysignalcondition{DivisionByZero}{Division by zero: #5#2/0.}{}{ 0/1}%
}%
\def\XINT_irr_zero #1#2#3#4#5{ 0/1}% changed in 1.08
\def\XINT_irr_loop_a #1#2%
{%
    \expandafter\XINT_irr_loop_d
    \romannumeral0\XINT_ndivrawpair {#1}{#2}{#1}%
}%
\def\XINT_irr_loop_d #1#2%
{%
    \XINT_irr_loop_e #2\Z
}%
\def\XINT_irr_loop_e #1#2\Z
```

```
{%
    \xint_gob_til_zero #1\XINT_irr_loop_exit0\XINT_irr_loop_a {#1#2}%
}%
\def\XINT_irr_loop_exit0\XINT_irr_loop_a #1#2#3#4%
{%
    \expandafter\XINT_irr_loop_exitb\expandafter
    {\romannumeral0\xintiiquo {#3}{#2}}%
    {\romannumeral0\xintiiquo {#4}{#2}}%
}%
\def\XINT_irr_loop_exitb #1#2%
{%
   \expandafter\XINT_irr_finish\expandafter {#2}{#1}%
}%
\def\XINT_irr_finish #1#2#3{#3#1/#2}% changed in 1.08
```

25.33. \xintifInt

```
\def\xintifInt   {\romannumeral0\xintifint }%
\def\xintifint #1{\expandafter\XINT_ifint\romannumeral0\xintrawwithzeros {#1}.}%
\def\XINT_ifint #1/#2.%
{%
    \if 0\xintiiRem {#1}{#2}%
     \expandafter\xint_stop_atfirstoftwo
    \else
     \expandafter\xint_stop_atsecondoftwo
    \fi
}%
```

25.34. \xintIsInt

Added at 1.3d only, for isint() xintexpr function.

```
\def\xintIsInt   {\romannumeral0\xintisint }%
\def\xintisint #1%
   {\expandafter\XINT_ifint\romannumeral0\xintrawwithzeros {#1}.10}%
```

25.35. \xintJrr

```
\def\xintJrr {\romannumeral0\xintjrr }%
\def\xintjrr #1%
{%
    \expandafter\XINT_jrr_start\romannumeral0\xintrawwithzeros {#1}\Z
}%
\def\XINT_jrr_start #1#2/#3\Z
{%
    \if0\XINT_isOne {#3}\xint_afterfi
          {\xint_UDsignfork
               #1\XINT_jrr_negative
                -{\XINT_jrr_nonneg #1}%
           \krof}%
    \else
      \xint_afterfi{\XINT_jrr_denomisone #1}%
    \fi
    #2\Z {#3}%
```

```
}%
\def\XINT_jrr_denomisone #1\Z #2{ #1/1}% changed in 1.08
\def\XINT_jrr_negative   #1\Z #2{\XINT_jrr_D #1\Z #2\Z -}%
\def\XINT_jrr_nonneg     #1\Z #2{\XINT_jrr_D #1\Z #2\Z \space}%
\def\XINT_jrr_D #1#2\Z #3#4\Z
{%
    \xint_UDzerosfork
       #3#1\XINT_jrr_indeterminate
       #30\XINT_jrr_divisionbyzero
       #10\XINT_jrr_zero
        00\XINT_jrr_loop_a
    \krof
    {#3#4}{#1#2}1001%
}%
\def\XINT_jrr_indeterminate #1#2#3#4#5#6#7%
{%
    \XINT_legacysignalcondition{DivisionUndefined}{0/0 indeterminate fraction.}{}{ 0/1}%
}%
\def\XINT_jrr_divisionbyzero #1#2#3#4#5#6#7%
{%
    \XINT_legacysignalcondition{DivisionByZero}{Division by zero: #7#2/0.}{}{ 0/1}%
}%
\def\XINT_jrr_zero #1#2#3#4#5#6#7{ 0/1}% changed in 1.08
\def\XINT_jrr_loop_a #1#2%
{%
    \expandafter\XINT_jrr_loop_b
    \romannumeral0\XINT_ndivrawpair {#1}{#2}{#1}%
}%
\def\XINT_jrr_loop_b #1#2#3#4#5#6#7%
{%
    \expandafter \XINT_jrr_loop_c \expandafter
        {\romannumeral0\xintiiadd{\xintiimul{#4}{#1}}{#6}}%
        {\romannumeral0\xintiiadd{\xintiimul{#5}{#1}}{#7}}%
    {#2}{#3}{#4}{#5}%
}%
\def\XINT_jrr_loop_c #1#2%
{%
    \expandafter \XINT_jrr_loop_d \expandafter{#2}{#1}%
}%
\def\XINT_jrr_loop_d #1#2#3#4%
{%
    \XINT_jrr_loop_e #3\Z {#4}{#2}{#1}%
}%
\def\XINT_jrr_loop_e #1#2\Z
{%
    \xint_gob_til_zero #1\XINT_jrr_loop_exit0\XINT_jrr_loop_a {#1#2}%
}%
\def\XINT_jrr_loop_exit0\XINT_jrr_loop_a #1#2#3#4#5#6%
{%
    \XINT_irr_finish {#3}{#4}%
}%
```

25.36. \xintTFrac

Added at 1.09i (2013/12/18). For frac in \xintexpr. And \xintFrac is already assigned. T for truncation. However, potentially not very efficient with numbers in scientific notations, with big exponents. Will have to think it again some day. I hesitated how to call the macro. Same convention as in maple, but some people reserve fractional part to x - floor(x). Also, not clear if I had to make it negative (or zero) if x < 0, or rather always positive. There should be in fact such a thing for each rounding function, trunc, round, floor, ceil.

```
\def\xintTFrac {\romannumeral0\xinttfrac }%
\def\xinttfrac #1{\expandafter\XINT_tfrac_fork\romannumeral0\xintrawwithzeros {#1}\Z }%
\def\XINT_tfrac_fork #1%
{%
    \xint_UDzerominusfork
        #1-\XINT_tfrac_zero
        0#1{\xintiiopp\XINT_tfrac_P }%
         0-{\XINT_tfrac_P #1}%
    \krof
}%
\def\XINT_tfrac_zero #1\Z { 0/1[0]}%
\def\XINT_tfrac_P #1/#2\Z {\expandafter\XINT_rez_AB
                           \romannumeral0\xintiirem{#1}{#2}\Z {0}{#2}}%
```

25.37. \xintTrunc, \xintiTrunc

Syntax: \xinttrunc{n}{X}, \xintitrunc{n}{X} (with \romannumeral0). calxfp's \CALX_fintp on A/B[N+n], {N}{A}{B} from \XINT_infrac: <s><T>.<d><z> (sign of X, digits of |trunc(X 10^n)|, first omitted digit, z = 1 if the omitted part is not zero). The post-processor (\XINT_trunc_b, or \XINT_round_b which adds 1 to T if d >= 5) hands T.<sign><n>. to the final macro \XINT_trunc_G (decimal output) or \XINT_itrunc_G (integer); the sign token is - or \space. Zero input (T = 0 and z = 0): 0.

Modified at 1.4a (2020/02/19).

Adds handling of a negative first argument.

Zero input still gives single digit 0 output as I did not want to complicate the code. But if quantization gives 0, the exponent [D] will be there. Well actually eD because of problem that sign of original is preserved in output so we can have -0 and I can not use -0[D] notation as it is not legal for strict format. So I will use -0eD hence eD generally even though this means so slight suboptimality for trunc() function in \xintexpr.

The idea to give a meaning to negative D (in the context of optional argument to \xintiexpr) was suggested a long time ago by Kpym (October 20, 2015). His suggestion was then to treat it as positive D but trim trailing zeroes. But since then, there is \xintDecToString which can be combined with \xintREZ, and I feel matters of formatting output require a whole module (or rather use existing third-party tools), and I decided to opt rather for an operation similar as the quantize() of Python Decimal module. I.e. we truncate (or round) to an integer multiple of a given power of 10.

Other reason to decide to do this is that it looks as if I don't even need to understand the original code to hack into its ending via \XINT_trunc_G or \XINT_itrunc_G. For the latter it looks as if logically I simply have to do nothing. For the former I simply have to add some eD postfix.

```
\def\xintTrunc  {\romannumeral0\xinttrunc }%
\def\xintiTrunc {\romannumeral0\xintitrunc}%
\def\xinttrunc #1{\expandafter\XINT_trunc\the\numexpr#1.\XINT_trunc_G}%
\def\xintitrunc #1{\expandafter\XINT_trunc\the\numexpr#1.\XINT_itrunc_G}%
\def\XINT_trunc #1.#2#3%
```

```
    {\expandafter\XINT_trunc_a\romannumeral0\XINT_infrac{#3}#1.\XINT_trunc_b#2}%
\def\XINT_trunc_a #1#2#3#4.#5%
    {\expandafter\XINT_trunc_c\the\numexpr#1+#4.{#2}{#3}#5#4.}%
\def\XINT_trunc_c #1.#2#3#4%
    {\expandafter#4\romannumeral0\CALX_fintp{#2}{#3}{#1}}%
\def\XINT_trunc_b #1#2#3.#4#5#6.#7%
{%
    \if0#2\if0#5\XINT_trunc_z\fi\fi
    \if-#1\xint_dothis{#7#2#3.-#6.}\fi
    \xint_orthat{#7#2#3.\space#6.}%
}%
\def\XINT_trunc_z #1\xint_orthat#2{\fi\fi\space0}%
\def\XINT_itrunc_G #1#2.#3#4.%
{%
    \if#10\xint_dothis{ 0}\fi
    \xint_orthat{#3#1}#2%
}%
\def\XINT_trunc_G #1.#2#3#4.%
{%
    \xint_gob_til_minus#3\XINT_trunc_Hc-%
    \expandafter\XINT_trunc_H
    \the\numexpr\romannumeral0\xintlength {#1}-#3#4.#3#4.{#1}#2%
}%
\def\XINT_trunc_Hc-\expandafter\XINT_trunc_H
    \the\numexpr\romannumeral0\xintlength #1.-#2.#3#4{#4#3e#2}%
\def\XINT_trunc_H #1.#2.%
{%
    \ifnum #1 > \xint_c_ \xint_dothis{\XINT_trunc_Ha {#2}}\fi
    \xint_orthat {\XINT_trunc_Hb {-#1}}% -0,--1,--2, ....
}%
\def\XINT_trunc_Ha%
{%
    \expandafter\XINT_trunc_Haa\romannumeral0\xintdecsplit
}%
\def\XINT_trunc_Haa #1#2#3{#3#1.#2}%
\def\XINT_trunc_Hb #1#2#3%
{%
    \expandafter #3\expandafter0\expandafter.%
    \romannumeral\xintreplicate{#1}0#2%
}%
```

25.38. \xintTTrunc

Added at 1.1 (2014/10/28).

```
\def\xintTTrunc {\romannumeral0\xintttrunc }%
\def\xintttrunc {\xintitrunc\xint_c_}%
```

25.39. \xintNum, \xintnum

```
\let\xintnum \xintttrunc
```

25.40. \xintRound, \xintiRound

As \xintTrunc, \xintiTrunc, with the post-processor \XINT_round_b: T + 1 (xintcore's \xintinc) if the first omitted digit is at least 5 (round half away from zero). A negative first argument is handled by \XINT_trunc_G.

```
\def\xintRound  {\romannumeral0\xintround }%
\def\xintiRound {\romannumeral0\xintiround }%
\def\xintround  #1{\expandafter\XINT_round\the\numexpr#1.\XINT_trunc_G}%
\def\xintiround #1{\expandafter\XINT_round\the\numexpr#1.\XINT_itrunc_G}%
\def\XINT_round #1.#2#3%
   {\expandafter\XINT_trunc_a\romannumeral0\XINT_infrac{#3}#1.\XINT_round_b#2}%
\def\XINT_round_b #1#2.#3#4%
{%
    \ifnum#3>\xint_c_iv
      \xint_dothis{\expandafter\XINT_round_c\romannumeral0\xintinc{#2}.#1#4}\fi
    \xint_orthat{\XINT_trunc_b #1#2.#3#4}%
}%
\def\XINT_round_c #1.#2#3{\XINT_trunc_b #2#1.0#3}%
```

25.41. \xintXTrunc

Temporarily use \xintTrunc: the former implementation was only x-expandable and more efficient. Reimplement this entry completely later.

```
\let\xintXTrunc\xintTrunc
```

25.42. \xintAdd

Modified at 1.3 (2018/03/01). Big change at 1.3: a/b+c/d uses lcm(b,d) as denominator.

```
\def\xintAdd {\romannumeral0\xintadd }%
\def\xintadd #1{\expandafter\XINT_fadd\romannumeral0\xintraw {#1}}%
\def\XINT_fadd #1{\xint_gob_til_zero #1\XINT_fadd_Azero 0\XINT_fadd_a #1}%
\def\XINT_fadd_Azero #1]{\xintraw }%
\def\XINT_fadd_a #1/#2[#3]#4%
   {\expandafter\XINT_fadd_b\romannumeral0\xintraw {#4}{#3}{#1}{#2}}%
\def\XINT_fadd_b #1{\xint_gob_til_zero #1\XINT_fadd_Bzero 0\XINT_fadd_c #1}%
\def\XINT_fadd_Bzero #1]#2#3#4{ #3/#4[#2]}%
\def\XINT_fadd_c #1/#2[#3]#4%
{%
    \expandafter\XINT_fadd_Aa\the\numexpr #4-#3.{#3}{#4}{#1}{#2}%
}%
\def\XINT_fadd_Aa #1%
{%
    \xint_UDzerominusfork
       #1-\XINT_fadd_B
        0#1\XINT_fadd_Bb
        0-\XINT_fadd_Ba
    \krof #1%
}%
\def\XINT_fadd_B   #1.#2#3#4#5#6#7{\XINT_fadd_C {#4}{#5}{#7}{#6}[#3]}%
\def\XINT_fadd_Ba  #1.#2#3#4#5#6#7%
{%
    \expandafter\XINT_fadd_C\expandafter
```

```
        {\romannumeral0\XINT_dsx_addzeros {#1}#6;}%
    {#7}{#5}{#4}[#2]%
}%
\def\XINT_fadd_Bb -#1.#2#3#4#5#6#7%
{%
    \expandafter\XINT_fadd_C\expandafter
        {\romannumeral0\XINT_dsx_addzeros {#1}#4;}%
    {#5}{#7}{#6}[#3]%
}%
\def\XINT_fadd_iszero #1[#2]{ 0/1[0]}% ou [#2] originel?
\def\XINT_fadd_C #1#2#3%
{%
    \expandafter\XINT_fadd_D_b
    \romannumeral0\XINT_ndivrawpair{#2}{#3}{#2}{#2}{#3}{#1}%
}%
```

Basically a clone of the \XINT_irr_loop_a loop. I should modify the output of \XINT_ndivrawpair perhaps to be optimized for checking if remainder vanishes.

```
\def\XINT_fadd_D_a #1#2%
{%
    \expandafter\XINT_fadd_D_b
    \romannumeral0\XINT_ndivrawpair {#1}{#2}{#1}%
}%
\def\XINT_fadd_D_b #1#2{\XINT_fadd_D_c #2\Z}%
\def\XINT_fadd_D_c #1#2\Z
{%
    \xint_gob_til_zero #1\XINT_fadd_D_exit0\XINT_fadd_D_a {#1#2}%
}%
\def\XINT_fadd_D_exit0\XINT_fadd_D_a #1#2#3%
{%
    \expandafter\XINT_fadd_E
    \romannumeral0\xintiiquo {#3}{#2}.{#2}%
}%
\def\XINT_fadd_E #1.#2#3%
{%
    \expandafter\XINT_fadd_F
    \romannumeral0\xintiimul{#1}{#3}.{\xintiiQuo{#3}{#2}}{#1}%
}%
\def\XINT_fadd_F #1.#2#3#4#5%
{%
    \expandafter\XINT_fadd_G
    \romannumeral0\xintiiadd{\xintiiMul{#2}{#4}}{\xintiiMul{#3}{#5}}/#1%
}%
\def\XINT_fadd_G #1{%
\def\XINT_fadd_G ##1{\if0##1\expandafter\XINT_fadd_iszero\fi#1##1}%
}\XINT_fadd_G{ }%
```

25.43. \xintSub

Modified at 1.3 (2018/03/01). Since 1.3 will use least common multiple of denominators.

```
\def\xintSub    {\romannumeral0\xintsub }%
\def\xintsub #1{\expandafter\XINT_fsub\romannumeral0\xintraw {#1}}%
\def\XINT_fsub #1{\xint_gob_til_zero #1\XINT_fsub_Azero 0\XINT_fsub_a #1}%
```

```
\def\XINT_fsub_Azero #1]{\xintopp }%
\def\XINT_fsub_a #1/#2[#3]#4%
   {\expandafter\XINT_fsub_b\romannumeral0\xintraw {#4}{#3}{#1}{#2}}%
\def\XINT_fsub_b #1{\xint_UDzerominusfork
                       #1-\XINT_fadd_Bzero
                        0#1\XINT_fadd_c
                        0-{\XINT_fadd_c -#1}%
                      \krof }%
```

25.44. \xintSum

There was (not documented anymore since 1.09d, 2013/10/22) a macro \xintSumExpr, but it has been deleted at 1.2l.

Empty items in the input are not accepted by this macro, but the input may be empty.

Refactored slightly at 1.4. \XINT_Sum used in xintexpr code.

```
\def\xintSum {\romannumeral0\xintsum }%
\def\xintsum #1{\expandafter\XINT_sum\romannumeral`&&@#1^}%
\def\XINT_Sum{\romannumeral0\XINT_sum}%
\def\XINT_sum#1%
{%
    \xint_gob_til_^ #1\XINT_sum_empty ^%
    \expandafter\XINT_sum_loop\romannumeral0\xintraw{#1}\xint:
}%
\def\XINT_sum_empty ^#1\xint:{ 0/1[0]}%
\def\XINT_sum_loop #1\xint:#2%
{%
    \xint_gob_til_^ #2\XINT_sum_end ^%
    \expandafter\XINT_sum_loop
    \romannumeral0\xintadd{#1}{\romannumeral0\xintraw{#2}}\xint:
}%
\def\XINT_sum_end ^#1\xintadd #2#3\xint:{ #2}%
```

25.45. \xintMul

```
\def\xintMul {\romannumeral0\xintmul }%
\def\xintmul #1{\expandafter\XINT_fmul\romannumeral0\xintraw {#1}.}%
\def\XINT_fmul #1{\xint_gob_til_zero #1\XINT_fmul_zero 0\XINT_fmul_a #1}%
\def\XINT_fmul_a #1[#2].#3%
   {\expandafter\XINT_fmul_b\romannumeral0\xintraw {#3}#1[#2.]}%
\def\XINT_fmul_b #1{\xint_gob_til_zero #1\XINT_fmul_zero 0\XINT_fmul_c #1}%
\def\XINT_fmul_c #1/#2[#3]#4/#5[#6.]%
{%
    \expandafter\XINT_fmul_d
    \expandafter{\the\numexpr #3+#6\expandafter}%
    \expandafter{\romannumeral0\xintiimul {#5}{#2}}%
    {\romannumeral0\xintiimul {#4}{#1}}%
}%
\def\XINT_fmul_d #1#2#3%
{%
    \expandafter \XINT_fmul_e \expandafter{#3}{#1}{#2}%
}%
\def\XINT_fmul_e #1#2{\XINT_outfrac {#2}{#1}}%
\def\XINT_fmul_zero #1.#2{ 0/1[0]}%
```

25.46. \xintSqr

```
\def\xintSqr {\romannumeral0\xintsqr }%
\def\xintsqr #1{\expandafter\XINT_fsqr\romannumeral0\xintraw {#1}}%
\def\XINT_fsqr #1{\xint_gob_til_zero #1\XINT_fsqr_zero 0\XINT_fsqr_a #1}%
\def\XINT_fsqr_a #1/#2[#3]%
{%
    \expandafter\XINT_fsqr_b
    \expandafter{\the\numexpr #3+#3\expandafter}%
    \expandafter{\romannumeral0\xintiisqr {#2}}%
    {\romannumeral0\xintiisqr {#1}}%
}%
\def\XINT_fsqr_b #1#2#3{\expandafter \XINT_fmul_e \expandafter{#3}{#1}{#2}}%
\def\XINT_fsqr_zero #1]{ 0/1[0]}%
```

25.47. \xintPow

ATTENTION, it gets redefined by xintexpr. Check there. 1.2f: to be coherent with the "i" convention \xintiPow should parse also its exponent via \xintNum when xintfrac.sty is loaded. This was not the case so far. Cependant le problème est que le fait d'appliquer \xintNum rend impossible certains inputs qui auraient pu être gérès par \numexpr. Le \numexpr externe est ici pour intercepter trop grand input.

```
\def\xintipow #1#2%
{%
    \expandafter\xint_pow\the\numexpr \xintNum{#2}\expandafter
    .\romannumeral0\xintnum{#1}\xint:
}%
\def\xintPow {\romannumeral0\xintpow }%
\def\xintpow #1%
{%
    \expandafter\XINT_fpow\expandafter {\romannumeral0\XINT_infrac {#1}}%
}%
\def\XINT_fpow #1#2%
{%
    \expandafter\XINT_fpow_fork\the\numexpr \xintNum{#2}\relax\Z #1%
}%
\def\XINT_fpow_fork #1#2\Z
{%
    \xint_UDzerominusfork
      #1-\XINT_fpow_zero
      0#1\XINT_fpow_neg
       0-{\XINT_fpow_pos #1}%
    \krof
    {#2}%
}%
\def\XINT_fpow_zero #1#2#3#4{ 1/1[0]}%
\def\XINT_fpow_pos #1#2#3#4#5%
{%
    \expandafter\XINT_fpow_pos_A\expandafter
    {\the\numexpr #1#2*#3\expandafter}\expandafter
    {\romannumeral0\xintiipow {#5}{#1#2}}%
    {\romannumeral0\xintiipow {#4}{#1#2}}%
}%
```

```
\def\XINT_fpow_neg #1#2#3#4%
{%
    \expandafter\XINT_fpow_pos_A\expandafter
    {\the\numexpr -#1*#2\expandafter}\expandafter
    {\romannumeral0\xintiipow {#3}{#1}}%
    {\romannumeral0\xintiipow {#4}{#1}}%
}%
\def\XINT_fpow_pos_A #1#2#3%
{%
    \expandafter\XINT_fpow_pos_B\expandafter {#3}{#1}{#2}%
}%
\def\XINT_fpow_pos_B #1#2{\XINT_outfrac {#2}{#1}}%
```

25.48. \xintFac

Factorial coefficients: variant which can be chained with other xintfrac macros.

```
\def\xintFac  {\romannumeral0\xintfac}%
\def\xintfac  #1{\expandafter\XINT_fac_fork\the\numexpr\xintNum{#1}.[0]}%
```

25.49. \xintBinomial

Added at 1.2f (2016/03/12).

```
\def\xintBinomial {\romannumeral0\xintbinomial}%
\def\xintbinomial #1#2%
{%
    \expandafter\XINT_binom_pre
    \the\numexpr\xintNum{#1}\expandafter.\the\numexpr\xintNum{#2}.[0]%
}%
```

25.50. \xintPFactorial

Added at 1.2f (2016/03/12). Partial factorial. For needs of xintexpr.sty.

```
\def\xintipfactorial #1#2%
{%
    \expandafter\XINT_pfac_fork
    \the\numexpr\xintNum{#1}\expandafter.\the\numexpr\xintNum{#2}.%
}%
\def\xintPFactorial {\romannumeral0\xintpfactorial}%
\def\xintpfactorial #1#2%
{%
    \expandafter\XINT_pfac_fork
    \the\numexpr\xintNum{#1}\expandafter.\the\numexpr\xintNum{#2}.[0]%
}%
```

25.51. \xintPrd

Refactored at 1.4. After some hesitation the routine still does not try to detect on the fly a zero item, to abort the loop. Indeed this would add some overhead generally (as we need normalizing the item before checking if it vanishes hence we must then grab things once more).

```
\def\xintPrd {\romannumeral0\xintprd }%
\def\xintprd #1{\expandafter\XINT_prd\romannumeral`&&@#1^}%
\def\XINT_Prd{\romannumeral0\XINT_prd}%
```

```
\def\XINT_prd#1%
{%
    \xint_gob_til_^ #1\XINT_prd_empty ^%
    \expandafter\XINT_prd_loop\romannumeral0\xintraw{#1}\xint:
}%
\def\XINT_prd_empty ^#1\xint:{ 1/1[0]}%
\def\XINT_prd_loop #1\xint:#2%
{%
    \xint_gob_til_^ #2\XINT_prd_end ^%
    \expandafter\XINT_prd_loop
    \romannumeral0\xintmul{#1}{\romannumeral0\xintraw{#2}}\xint:
}%
\def\XINT_prd_end ^#1\xintmul #2#3\xint:{ #2}%
```

25.52. \xintDiv

```
\def\xintDiv {\romannumeral0\xintdiv }%
\def\xintdiv #1%
{%
    \expandafter\XINT_fdiv\expandafter {\romannumeral0\XINT_infrac {#1}}%
}%
\def\XINT_fdiv #1#2%
   {\expandafter\XINT_fdiv_A\romannumeral0\XINT_infrac {#2}#1}%
\def\XINT_fdiv_A #1#2#3#4#5#6%
{%
    \expandafter\XINT_fdiv_B
    \expandafter{\the\numexpr #4-#1\expandafter}%
    \expandafter{\romannumeral0\xintiimul {#2}{#6}}%
    {\romannumeral0\xintiimul {#3}{#5}}%
}%
\def\XINT_fdiv_B #1#2#3%
{%
    \expandafter\XINT_fdiv_C
    \expandafter{#3}{#1}{#2}%
}%
\def\XINT_fdiv_C #1#2{\XINT_outfrac {#2}{#1}}%
```

25.53. \xintDivFloor

Added at 1.1 (2014/10/28). Changed at 1.2p to not append /1[0] ending but rather output a big integer in strict format, like \xintDivTrunc and \xintDivRound.

```
\def\xintDivFloor     {\romannumeral0\xintdivfloor }%
\def\xintdivfloor #1#2{\xintifloor{\xintDiv {#1}{#2}}}%
```

25.54. \xintDivTrunc

Added at 1.1 (2014/10/28).

```
\def\xintDivTrunc     {\romannumeral0\xintdivtrunc }%
\def\xintdivtrunc #1#2{\xintttrunc {\xintDiv {#1}{#2}}}%
```

25.55. \xintDivRound

1.1

```
\def\xintDivRound     {\romannumeral0\xintdivround }%
\def\xintdivround #1#2{\xintiround 0{\xintDiv {#1}{#2}}}%
```

25.56. \xintModTrunc

Added at 1.1 (2014/10/28). \xintModTrunc {q1}{q2} computes q1 - q2*t(q1/q2) with t(q1/q2) equal to the truncated division of two fractions q1 and q2.
Its former name, prior to 1.2p, was \xintMod.

Modified at 1.3 (2018/03/01). At 1.3, uses least common multiple denominator, like \xintMod (next).

```
\def\xintModTrunc {\romannumeral0\xintmodtrunc }%
\def\xintmodtrunc #1{\expandafter\XINT_modtrunc_a\romannumeral0\xintraw{#1}.}%
\def\XINT_modtrunc_a #1#2.#3%
   {\expandafter\XINT_modtrunc_b\expandafter #1\romannumeral0\xintraw{#3}#2.}%
\def\XINT_modtrunc_b #1#2% #1 de A, #2 de B.
{%
    \if0#2\xint_dothis{\XINT_modtrunc_divbyzero #1#2}\fi
    \if0#1\xint_dothis\XINT_modtrunc_aiszero\fi
    \if-#2\xint_dothis{\XINT_modtrunc_bneg #1}\fi
          \xint_orthat{\XINT_modtrunc_bpos #1#2}%
}%
\def\XINT_modtrunc_divbyzero #1#2[#3]#4.%
{%
    \XINT_legacysignalcondition{DivisionByZero}{Division by zero: #1#4/(#2[#3]).}{}{ 0/1[0]}%
}%
\def\XINT_modtrunc_aiszero #1.{ 0/1[0]}%
\def\XINT_modtrunc_bneg #1%
{%
    \xint_UDsignfork
         #1{\xintiiopp\XINT_modtrunc_pos {}}%
          -{\XINT_modtrunc_pos #1}%
    \krof
}%
\def\XINT_modtrunc_bpos #1%
{%
    \xint_UDsignfork
            #1{\xintiiopp\XINT_modtrunc_pos {}}%
             -{\XINT_modtrunc_pos #1}%
    \krof
}%
```

Attention. This crucially uses that xint's \xintiiE{x}{e} is defined to return x unchanged if e is negative (and x extended by e zeroes if e >= 0).

```
\def\XINT_modtrunc_pos #1#2/#3[#4]#5/#6[#7].%
{%
    \expandafter\XINT_modtrunc_pos_a
    \the\numexpr\ifnum#7>#4 #4\else #7\fi\expandafter.%
    \romannumeral0\expandafter\XINT_mod_D_b
    \romannumeral0\XINT_ndivrawpair{#3}{#6}{#3}{#3}{#6}%
    {#1#5}{#7-#4}{#2}{#4-#7}%
```

```
}%
\def\XINT_modtrunc_pos_a #1.#2#3#4{\xintiirem {#3}{#4}/#2[#1]}%
```

25.57. \xintDivMod

Added at 1.2p (2017/12/05). \xintDivMod{q1}{q2} outputs {floor(q1/q2)}{q1 - q2*floor(q1/q2)}. Attention that it relies on \xintiiE{x}{e} returning x if e < 0.

Modified at 1.3 (2018/03/01). Modified (like \xintAdd and \xintSub) at 1.3 to use a l.c.m for final denominator of the "mod" part.

```
\def\xintDivMod {\romannumeral0\xintdivmod }%
\def\xintdivmod #1{\expandafter\XINT_divmod_a\romannumeral0\xintraw{#1}.}%
\def\XINT_divmod_a #1#2.#3%
   {\expandafter\XINT_divmod_b\expandafter #1\romannumeral0\xintraw{#3}#2.}%
\def\XINT_divmod_b #1#2% #1 de A, #2 de B.
{%
    \if0#2\xint_dothis{\XINT_divmod_divbyzero #1#2}\fi
    \if0#1\xint_dothis\XINT_divmod_aiszero\fi
    \if-#2\xint_dothis{\XINT_divmod_bneg #1}\fi
          \xint_orthat{\XINT_divmod_bpos #1#2}%
}%
\def\XINT_divmod_divbyzero #1#2[#3]#4.%
{%
    \XINT_legacysignalcondition{DivisionByZero}{Division by zero: #1#4/(#2[#3]).}{}%
    {{0}{0/1[0]}}% à revoir...
}%
\def\XINT_divmod_aiszero #1.{{0}{0/1[0]}}%
\def\XINT_divmod_bneg #1% f // -g = (-f) // g, f % -g = - ((-f) % g)
{%
    \expandafter\XINT_divmod_bneg_finish
    \romannumeral0\xint_UDsignfork
        #1{\XINT_divmod_bpos {}}%
         -{\XINT_divmod_bpos {-#1}}%
    \krof
}%
\def\XINT_divmod_bneg_finish#1#2%
{%
    \expandafter\xint_exchangetwo_keepbraces\expandafter
    {\romannumeral0\xintiiopp#2}{#1}%
}%
\def\XINT_divmod_bpos #1#2/#3[#4]#5/#6[#7].%
{%
    \expandafter\XINT_divmod_bpos_a
    \the\numexpr\ifnum#7>#4 #4\else #7\fi\expandafter.%
    \romannumeral0\expandafter\XINT_mod_D_b
    \romannumeral0\XINT_ndivrawpair{#3}{#6}{#3}{#3}{#6}%
    {#1#5}{#7-#4}{#2}{#4-#7}%
}%
\def\XINT_divmod_bpos_a #1.#2#3#4%
{%
    \expandafter\XINT_divmod_bpos_finish
    \romannumeral0\xintiidivision{#3}{#4}{/#2[#1]}%
}%
```

```
\def\XINT_divmod_bpos_finish #1#2#3{{#1}{#2#3}}%
```

25.58. \xintMod

Added at 1.2p (2017/12/05). \xintMod{q1}{q2} computes q1 - q2*floor(q1/q2). Attention that it relies on \xintiiE{x}{e} returning x if e < 0.
Prior to 1.2p, that macro had the meaning now attributed to \xintModTrunc.

Modified at 1.3 (2018/03/01). Modified (like \xintAdd and \xintSub) at 1.3 to use a l.c.m for final denominator.

```
\def\xintMod {\romannumeral0\xintmod }%
\def\xintmod #1{\expandafter\XINT_mod_a\romannumeral0\xintraw{#1}.}%
\def\XINT_mod_a #1#2.#3%
   {\expandafter\XINT_mod_b\expandafter #1\romannumeral0\xintraw{#3}#2.}%
\def\XINT_mod_b #1#2% #1 de A, #2 de B.
{%
    \if0#2\xint_dothis{\XINT_mod_divbyzero #1#2}\fi
    \if0#1\xint_dothis\XINT_mod_aiszero\fi
    \if-#2\xint_dothis{\XINT_mod_bneg #1}\fi
          \xint_orthat{\XINT_mod_bpos #1#2}%
}%
```

Attention to not move ModTrunc code beyond that point.

```
\let\XINT_mod_divbyzero\XINT_modtrunc_divbyzero
\let\XINT_mod_aiszero  \XINT_modtrunc_aiszero
\def\XINT_mod_bneg #1% f % -g = - ((-f) % g), for g > 0
{%
    \xintiiopp\xint_UDsignfork
        #1{\XINT_mod_bpos {}}%
         -{\XINT_mod_bpos {-#1}}%
    \krof
}%
\def\XINT_mod_bpos #1#2/#3[#4]#5/#6[#7].%
{%
    \expandafter\XINT_mod_bpos_a
    \the\numexpr\ifnum#7>#4 #4\else #7\fi\expandafter.%
    \romannumeral0\expandafter\XINT_mod_D_b
    \romannumeral0\XINT_ndivrawpair{#3}{#6}{#3}{#3}{#6}%
    {#1#5}{#7-#4}{#2}{#4-#7}%
}%
\def\XINT_mod_D_a #1#2%
{%
    \expandafter\XINT_mod_D_b
    \romannumeral0\XINT_ndivrawpair {#1}{#2}{#1}%
}%
\def\XINT_mod_D_b #1#2{\XINT_mod_D_c #2\Z}%
\def\XINT_mod_D_c #1#2\Z
{%
    \xint_gob_til_zero #1\XINT_mod_D_exit0\XINT_mod_D_a {#1#2}%
}%
\def\XINT_mod_D_exit0\XINT_mod_D_a #1#2#3%
{%
    \expandafter\XINT_mod_E
    \romannumeral0\xintiiquo {#3}{#2}.{#2}%
```

```
}%
\def\XINT_mod_E #1.#2#3%
{%
    \expandafter\XINT_mod_F
    \romannumeral0\xintiimul{#1}{#3}.{\xintiiQuo{#3}{#2}}{#1}%
}%
\def\XINT_mod_F #1.#2#3#4#5#6#7%
{%
    {#1}{\xintiiE{\xintiiMul{#4}{#3}}{#5}}%
        {\xintiiE{\xintiiMul{#6}{#2}}{#7}}%
}%
\def\XINT_mod_bpos_a #1.#2#3#4{\xintiirem {#3}{#4}/#2[#1]}%
```

25.59. \xintIsOne

Added at 1.09a (2013/09/24). Could be more efficient. For fractions with big powers of tens, it is better to use \xintCmp{f}{1}. Restyled in 1.09i.

```
\def\xintIsOne   {\romannumeral0\xintisone }%
\def\xintisone #1{\expandafter\XINT_fracisone
                  \romannumeral0\xintrawwithzeros{#1}\Z }%
\def\XINT_fracisone #1/#2\Z
    {\if0\xintiiCmp {#1}{#2}\xint_afterfi{ 1}\else\xint_afterfi{ 0}\fi}%
```

25.60. \xintGeq

```
\def\xintGeq {\romannumeral0\xintgeq }%
\def\xintgeq #1%
{%
    \expandafter\XINT_fgeq\expandafter {\romannumeral0\xintabs {#1}}%
}%
\def\XINT_fgeq #1#2%
{%
    \expandafter\XINT_fgeq_A \romannumeral0\xintabs {#2}#1%
}%
\def\XINT_fgeq_A #1%
{%
    \xint_gob_til_zero #1\XINT_fgeq_Zii 0%
    \XINT_fgeq_B #1%
}%
\def\XINT_fgeq_Zii 0\XINT_fgeq_B #1[#2]#3[#4]{ 1}%
\def\XINT_fgeq_B #1/#2[#3]#4#5/#6[#7]%
{%
    \xint_gob_til_zero #4\XINT_fgeq_Zi 0%
    \expandafter\XINT_fgeq_C\expandafter
    {\the\numexpr #7-#3\expandafter}\expandafter
    {\romannumeral0\xintiimul {#4#5}{#2}}%
    {\romannumeral0\xintiimul {#6}{#1}}%
}%
\def\XINT_fgeq_Zi 0#1#2#3#4#5#6#7{ 0}%
\def\XINT_fgeq_C #1#2#3%
{%
    \expandafter\XINT_fgeq_D\expandafter
```

```
    {#3}{#1}{#2}%
}%
\def\XINT_fgeq_D #1#2#3%
{%
    \expandafter\XINT_cntSgnFork\romannumeral`&&@\expandafter\XINT_cntSgn
     \the\numexpr #2+\xintLength{#3}-\xintLength{#1}\relax\xint:
    { 0}{\XINT_fgeq_E #2\Z {#3}{#1}}{ 1}%
}%
\def\XINT_fgeq_E #1%
{%
    \xint_UDsignfork
        #1\XINT_fgeq_Fd
         -{\XINT_fgeq_Fn #1}%
    \krof
}%
\def\XINT_fgeq_Fd #1\Z #2#3%
{%
    \expandafter\XINT_fgeq_Fe
    \romannumeral0\XINT_dsx_addzeros {#1}#3;\xint:#2\xint:
}%
\def\XINT_fgeq_Fe #1\xint:#2#3\xint:{\XINT_geq_plusplus #2#1\xint:#3\xint:}%
\def\XINT_fgeq_Fn #1\Z #2#3%
{%
    \expandafter\XINT_fgeq_Fo
    \romannumeral0\XINT_dsx_addzeros {#1}#2;\xint:#3\xint:
}%
\def\XINT_fgeq_Fo #1#2\xint:#3\xint:{\XINT_geq_plusplus #1#3\xint:#2\xint:}%
```

25.61. \xintMax

```
\def\xintMax {\romannumeral0\xintmax }%
\def\xintmax #1%
{%
    \expandafter\XINT_fmax\expandafter {\romannumeral0\xintraw {#1}}%
}%
\def\XINT_fmax #1#2%
{%
    \expandafter\XINT_fmax_A\romannumeral0\xintraw {#2}#1%
}%
\def\XINT_fmax_A #1#2/#3[#4]#5#6/#7[#8]%
{%
    \xint_UDsignsfork
      #1#5\XINT_fmax_minusminus
       -#5\XINT_fmax_firstneg
       #1-\XINT_fmax_secondneg
        --\XINT_fmax_nonneg_a
    \krof
    #1#5{#2/#3[#4]}{#6/#7[#8]}%
}%
\def\XINT_fmax_minusminus --%
   {\expandafter-\romannumeral0\XINT_fmin_nonneg_b }%
\def\XINT_fmax_firstneg #1-#2#3{ #1#2}%
\def\XINT_fmax_secondneg -#1#2#3{ #1#3}%
```

```
\def\XINT_fmax_nonneg_a #1#2#3#4%
{%
    \XINT_fmax_nonneg_b {#1#3}{#2#4}%
}%
\def\XINT_fmax_nonneg_b #1#2%
{%
    \if0\romannumeral0\XINT_fgeq_A #1#2%
          \xint_afterfi{ #1}%
    \else \xint_afterfi{ #2}%
    \fi
}%
```

25.62. \xintMaxof

1.2l protects \xintMaxof against items with non terminated \the\numexpr expressions.
1.4 renders the macro compatible with an empty argument and it also defines an accessor \XINT_Maxof suitable for xintexpr usage (formerly xintexpr had its own macro handling comma separated values, but it changed internal representation at 1.4).

```
\def\xintMaxof {\romannumeral0\xintmaxof }%
\def\xintmaxof #1{\expandafter\XINT_maxof\romannumeral`&&@#1^}%
\def\XINT_Maxof{\romannumeral0\XINT_maxof}%
\def\XINT_maxof#1%
{%
    \xint_gob_til_^ #1\XINT_maxof_empty ^%
    \expandafter\XINT_maxof_loop\romannumeral0\xintraw{#1}\xint:
}%
\def\XINT_maxof_empty ^#1\xint:{ 0/1[0]}%
\def\XINT_maxof_loop #1\xint:#2%
{%
    \xint_gob_til_^ #2\XINT_maxof_e ^%
    \expandafter\XINT_maxof_loop
    \romannumeral0\xintmax{#1}{\romannumeral0\xintraw{#2}}\xint:
}%
\def\XINT_maxof_e ^#1\xintmax #2#3\xint:{ #2}%
```

25.63. \xintMin

```
\def\xintMin {\romannumeral0\xintmin }%
\def\xintmin #1%
{%
    \expandafter\XINT_fmin\expandafter {\romannumeral0\xintraw {#1}}%
}%
\def\XINT_fmin #1#2%
{%
    \expandafter\XINT_fmin_A\romannumeral0\xintraw {#2}#1%
}%
\def\XINT_fmin_A #1#2/#3[#4]#5#6/#7[#8]%
{%
    \xint_UDsignsfork
      #1#5\XINT_fmin_minusminus
       -#5\XINT_fmin_firstneg
       #1-\XINT_fmin_secondneg
        --\XINT_fmin_nonneg_a
```

```
    \krof
    #1#5{#2/#3[#4]}{#6/#7[#8]}%
}%
\def\XINT_fmin_minusminus --%
   {\expandafter-\romannumeral0\XINT_fmax_nonneg_b }%
\def\XINT_fmin_firstneg #1-#2#3{ -#3}%
\def\XINT_fmin_secondneg -#1#2#3{ -#2}%
\def\XINT_fmin_nonneg_a #1#2#3#4%
{%
    \XINT_fmin_nonneg_b {#1#3}{#2#4}%
}%
\def\XINT_fmin_nonneg_b #1#2%
{%
    \if0\romannumeral0\XINT_fgeq_A #1#2%
          \xint_afterfi{ #2}%
    \else \xint_afterfi{ #1}%
    \fi
}%
```

25.64. \xintMinof

1.2l protects \xintMinof against items with non terminated \the\numexpr expressions.
1.4 version is compatible with an empty input (empty items are handled as zero).

```
\def\xintMinof {\romannumeral0\xintminof }%
\def\xintminof #1{\expandafter\XINT_minof\romannumeral`&&@#1^}%
\def\XINT_Minof{\romannumeral0\XINT_minof}%
\def\XINT_minof#1%
{%
    \xint_gob_til_^ #1\XINT_minof_empty ^%
    \expandafter\XINT_minof_loop\romannumeral0\xintraw{#1}\xint:
}%
\def\XINT_minof_empty ^#1\xint:{ 0/1[0]}%
\def\XINT_minof_loop #1\xint:#2%
{%
    \xint_gob_til_^ #2\XINT_minof_e ^%
    \expandafter\XINT_minof_loop\romannumeral0\xintmin{#1}{\romannumeral0\xintraw{#2}}\xint:
}%
\def\XINT_minof_e ^#1\xintmin #2#3\xint:{ #2}%
```

25.65. \xintCmp

```
\def\xintCmp {\romannumeral0\xintcmp }%
\def\xintcmp #1%
{%
    \expandafter\XINT_fcmp\expandafter {\romannumeral0\xintraw {#1}}%
}%
\def\XINT_fcmp #1#2%
{%
    \expandafter\XINT_fcmp_A\romannumeral0\xintraw {#2}#1%
}%
\def\XINT_fcmp_A #1#2/#3[#4]#5#6/#7[#8]%
{%
    \xint_UDsignsfork
```

```
      #1#5\XINT_fcmp_minusminus
       -#5\XINT_fcmp_firstneg
       #1-\XINT_fcmp_secondneg
        --\XINT_fcmp_nonneg_a
    \krof
    #1#5{#2/#3[#4]}{#6/#7[#8]}%
}%
\def\XINT_fcmp_minusminus --#1#2{\XINT_fcmp_B #2#1}%
\def\XINT_fcmp_firstneg #1-#2#3{ -1}%
\def\XINT_fcmp_secondneg -#1#2#3{ 1}%
\def\XINT_fcmp_nonneg_a #1#2%
{%
    \xint_UDzerosfork
      #1#2\XINT_fcmp_zerozero
       0#2\XINT_fcmp_firstzero
       #10\XINT_fcmp_secondzero
        00\XINT_fcmp_pos
    \krof
    #1#2%
}%
\def\XINT_fcmp_zerozero   #1#2#3#4{ 0}%
\def\XINT_fcmp_firstzero  #1#2#3#4{ -1}%
\def\XINT_fcmp_secondzero #1#2#3#4{ 1}%
\def\XINT_fcmp_pos #1#2#3#4%
{%
    \XINT_fcmp_B #1#3#2#4%
}%
\def\XINT_fcmp_B #1/#2[#3]#4/#5[#6]%
{%
    \expandafter\XINT_fcmp_C\expandafter
    {\the\numexpr #6-#3\expandafter}\expandafter
    {\romannumeral0\xintiimul {#4}{#2}}%
    {\romannumeral0\xintiimul {#5}{#1}}%
}%
\def\XINT_fcmp_C #1#2#3%
{%
    \expandafter\XINT_fcmp_D\expandafter
    {#3}{#1}{#2}%
}%
\def\XINT_fcmp_D #1#2#3%
{%
    \expandafter\XINT_cntSgnFork\romannumeral`&&@\expandafter\XINT_cntSgn
    \the\numexpr #2+\xintLength{#3}-\xintLength{#1}\relax\xint:
    { -1}{\XINT_fcmp_E #2\Z {#3}{#1}}{ 1}%
}%
\def\XINT_fcmp_E #1%
{%
    \xint_UDsignfork
        #1\XINT_fcmp_Fd
         -{\XINT_fcmp_Fn #1}%
    \krof
}%
```

```
\def\XINT_fcmp_Fd #1\Z #2#3%
{%
    \expandafter\XINT_fcmp_Fe
    \romannumeral0\XINT_dsx_addzeros {#1}#3;\xint:#2\xint:
}%
\def\XINT_fcmp_Fe #1\xint:#2#3\xint:{\XINT_cmp_plusplus #2#1\xint:#3\xint:}%
\def\XINT_fcmp_Fn #1\Z #2#3%
{%
    \expandafter\XINT_fcmp_Fo
    \romannumeral0\XINT_dsx_addzeros {#1}#2;\xint:#3\xint:
}%
\def\XINT_fcmp_Fo #1#2\xint:#3\xint:{\XINT_cmp_plusplus #1#3\xint:#2\xint:}%
```

25.66. \xintAbs

```
\def\xintAbs    {\romannumeral0\xintabs }%
\def\xintabs #1{\expandafter\XINT_abs\romannumeral0\xintraw {#1}}%
```

25.67. \xintOpp

```
\def\xintOpp    {\romannumeral0\xintopp }%
\def\xintopp #1{\expandafter\XINT_opp\romannumeral0\xintraw {#1}}%
```

25.68. \xintInv

Modified at 1.3d (2019/01/06).

```
\def\xintInv    {\romannumeral0\xintinv }%
\def\xintinv #1{\expandafter\XINT_inv\romannumeral0\xintraw {#1}}%
\def\XINT_inv #1%
{%
    \xint_UDzerominusfork
      #1-\XINT_inv_iszero
      0#1\XINT_inv_a
       0-{\XINT_inv_a {}}%
    \krof #1%
}%
\def\XINT_inv_iszero #1]%
   {\XINT_legacysignalcondition{DivisionByZero}{Inverse of zero: inv(#1]).}{}{ 0/1[0]}}%
\def\XINT_inv_a #1#2/#3[#4#5]%
{%
    \xint_UDzerominusfork
      #4-\XINT_inv_expiszero
      0#4\XINT_inv_b
       0-{\XINT_inv_b -#4}%
    \krof #5.{#1#3/#2}%
}%
\def\XINT_inv_expiszero #1.#2{ #2[0]}%
\def\XINT_inv_b #1.#2{ #2[#1]}%
```

25.69. \xintSgn

```
\def\xintSgn    {\romannumeral0\xintsgn }%
\def\xintsgn #1{\expandafter\XINT_sgn\romannumeral0\xintraw {#1}\xint:}%
```

25.70. \xintSignBit

Added at 1.4l (2022/05/29).

```
\def\xintSignBit   {\romannumeral0\xintsignbit }%
\def\xintsignbit #1{\expandafter\XINT_signbit\romannumeral0\xintraw {#1}\xint:}%
\def\XINT_signbit #1#2\xint:
{%
    \xint_UDzerominusfork
      #1-{ 0}%
      0#1{ 1}%
       0-{ 0}%
    \krof
}%
```

25.71. \xintGCD

Modified at 1.4 (2020/01/31). They replace the former xintgcd macros of the same names which truncated to integers their arguments. Fraction-producing gcd() and lcm() functions were available since 1.3d xintexpr, with non-public support macros handling comma separated values.

Modified at 1.4d (2021/03/29). Somewhat strangely \xintGCD was formerly \xintGCDof used with only two arguments, as the latter directly implemented a fractionl gcd algorithm using \xintMod repeatedly for two arguments.

Now \xintGCD contains the pairwise gcd routine and \xintGCDof is only a wrapper. And the pairwise gcd is reduced to integer-only computations to hopefully reduce fraction overhead.

Each input is filtered via \xintPIrr and \xintREZ to reduce size of maniuplate integers in algebra.

But hesitation about applying \xintPIrr to output, and/or \xintREZ. (as it is applied on input).

But as the code is now used for frational lcm's we actually need to do some reduction of output else lcm's of integers will not be necessarily printed by \xinteval as integers.

Well finally I apply \xintIrr (but not \xintREZ to output). Hesitations here (thinking of inputs with large [n] parts, the output will have many zeros). So I do this only for the user macro but the core routine as used by \xintGCDof will not do it.

Also at 1.4d the code uses \expanded.

```
\def\xintGCD {\romannumeral0\xintgcd}%
\def\xintgcd #1%
{%
    \expandafter\XINT_fgcd_in
    \romannumeral0\xintrez{\xintPIrr{\xintAbs{#1}}}\xint:
}%
\def\XINT_fgcd_in #1#2\xint:#3%
{%
    \expandafter\XINT_fgcd_out
    \romannumeral0\expandafter\XINT_fgcd_chkzeros\expandafter#1%
    \romannumeral0\xintrez{\xintPIrr{\xintAbs{#3}}}\xint:#1#2\xint:
}%
\def\XINT_fgcd_out#1[#2]{\xintirr{#1[#2]}[0]}%
\def\XINT_fgcd_chkzeros #1#2%
{%
    \xint_UDzerofork
       #1\XINT_fgcd_aiszero
       #2\XINT_fgcd_biszero
        0\XINT_fgcd_main
```

```
    \krof #2%
}%
\def\XINT_fgcd_aiszero #1\xint:#2\xint:{ #1}%
\def\XINT_fgcd_biszero #1\xint:#2\xint:{ #2}%
\def\XINT_fgcd_main #1/#2[#3]\xint:#4/#5[#6]\xint:
{%
    \expandafter\XINT_fgcd_a
      \romannumeral0\xintiigcd{#2}{#5}\xint:
      #2\xint:#5\xint:#1\xint:#4\xint:#3.#6.%
}%
\def\XINT_fgcd_a #1\xint:#2\xint:
{%
    \expandafter\XINT_fgcd_b
    \romannumeral0\xintiiquo{#2}{#1}\xint:#1\xint:#2\xint:
}%
\def\XINT_fgcd_b #1\xint:#2\xint:#3\xint:#4\xint:#5\xint:#6\xint:#7.#8.%
{%
   \expanded{%
   \xintiigcd{\xintiiE{\xintiiMul{#5}{\xintiiQuo{#4}{#2}}}{#7-#8}}%
             {\xintiiE{\xintiiMul{#6}{#1}}{#8-#7}}%
    /\xintiiMul{#1}{#4}%
    [\ifnum#7>#8 #8\else #7\fi]%
   }%
}%
```

25.72. \xintGCDof

Modified at 1.4 (2020/01/31). This inherits from former non public xintexpr macro called \xintGCDof:csv, which handled comma separated items.

It handles fractions presented as braced items and is the support macro for the gcd() function in \xintexpr and \xintfloatexpr. The support macro for the gcd() function in \xintiiexpr is \xintiiGCDof, from xint.

An empty input is allowed but I have some hesitations on the return value of 1.

Modified at 1.4d (2021/03/29). Sadly the 1.4 version had multiple problems:

- broken if first argument vanished,
- broken if some argument was not in strict format, for example had leading chains of signs or zeros (\xintGCDof{2}{03}). This bug originates in the fact the original macro was used only in xintexpr sanitized context.

Also, output is now always an irreducible fraction (ending with [0]).

```
\def\xintGCDof {\romannumeral0\xintgcdof}%
\def\xintgcdof #1{\expandafter\XINT_fgcdof\romannumeral`&&@#1^}%
\def\XINT_GCDof{\romannumeral0\XINT_fgcdof}%
\def\XINT_fgcdof #1%
{%
    \expandafter\XINT_fgcdof_chkempty\romannumeral`&&@#1\xint:
}%
\def\XINT_fgcdof_chkempty #1%
{%
    \xint_gob_til_^#1\XINT_fgcdof_empty ^\XINT_fgcdof_in #1%
}%
```

```
\def\XINT_fgcdof_empty #1\xint:{ 1/1[0]}% hesitation, should it be infinity? 0?
\def\XINT_fgcdof_in #1\xint:
{%
    \expandafter\XINT_fgcd_out
    \romannumeral0\expandafter\XINT_fgcdof_loop
    \romannumeral0\xintrez{\xintPIrr{\xintAbs{#1}}}\xint:
}%
\def\XINT_fgcdof_loop #1\xint:#2%
{%
    \expandafter\XINT_fgcdof_chkend\romannumeral`&&@#2\xint:#1\xint:\xint:
}%
\def\XINT_fgcdof_chkend #1%
{%
    \xint_gob_til_^#1\XINT_fgcdof_end ^\XINT_fgcdof_loop_pair #1%
}%
\def\XINT_fgcdof_end #1\xint:#2\xint:\xint:{ #2}%
\def\XINT_fgcdof_loop_pair #1\xint:#2%
{%
    \expandafter\XINT_fgcdof_loop
    \romannumeral0\expandafter\XINT_fgcd_chkzeros\expandafter#2%
    \romannumeral0\xintrez{\xintPIrr{\xintAbs{#1}}}\xint:#2%
}%
```

25.73. \xintLCM

Same comments as for \xintGCD. Entirely redone for 1.4d. Well, actually we can express it in terms of fractional gcd.

```
\def\xintLCM {\romannumeral0\xintlcm}%
\def\xintlcm #1%
{%
    \expandafter\XINT_flcm_in
    \romannumeral0\xintrez{\xintPIrr{\xintAbs{#1}}}\xint:
}%
\def\XINT_flcm_in #1#2\xint:#3%
{%
    \expandafter\XINT_fgcd_out
    \romannumeral0\expandafter\XINT_flcm_chkzeros\expandafter#1%
    \romannumeral0\xintrez{\xintPIrr{\xintAbs{#3}}}\xint:#1#2\xint:
}%
\def\XINT_flcm_chkzeros #1#2%
{%
    \xint_UDzerofork
       #1\XINT_flcm_zero
       #2\XINT_flcm_zero
        0\XINT_flcm_main
    \krof #2%
}%
\def\XINT_flcm_zero #1\xint:#2\xint:{ 0/1[0]}%
\def\XINT_flcm_main #1/#2[#3]\xint:#4/#5[#6]\xint:
{%
    \xintinv
    {%
```

```
      \romannumeral0\XINT_fgcd_main #2/#1[-#3]\xint:#5/#4[-#6]\xint:
    }%
}%
```

25.74. \xintLCMof

See comments for \xintGCDof. xint provides the integer only \xintiiLCMof.

Modified at 1.4d (2021/03/29). Sadly, although a public xintfrac macro, it did not (since 1.4) sanitize its arguments like other xintfrac macros.

```
\def\xintLCMof {\romannumeral0\xintlcmof}%
\def\xintlcmof #1{\expandafter\XINT_flcmof\romannumeral`&&@#1^}%
\def\XINT_LCMof{\romannumeral0\XINT_flcmof}%
\def\XINT_flcmof #1%
{%
    \expandafter\XINT_flcmof_chkempty\romannumeral`&&@#1\xint:
}%
\def\XINT_flcmof_chkempty #1%
{%
    \xint_gob_til_^#1\XINT_flcmof_empty ^\XINT_flcmof_in #1%
}%
\def\XINT_flcmof_empty #1\xint:{ 0/1[0]}% hesitation
\def\XINT_flcmof_in #1\xint:
{%
    \expandafter\XINT_fgcd_out
    \romannumeral0\expandafter\XINT_flcmof_loop
    \romannumeral0\xintrez{\xintPIrr{\xintAbs{#1}}}\xint:
}%
\def\XINT_flcmof_loop #1\xint:#2%
{%
    \expandafter\XINT_flcmof_chkend\romannumeral`&&@#2\xint:#1\xint:\xint:
}%
\def\XINT_flcmof_chkend #1%
{%
    \xint_gob_til_^#1\XINT_flcmof_end ^\XINT_flcmof_loop_pair #1%
}%
\def\XINT_flcmof_end #1\xint:#2\xint:\xint:{ #2}%
\def\XINT_flcmof_loop_pair #1\xint:#2%
{%
    \expandafter\XINT_flcmof_chkzero
    \romannumeral0\expandafter\XINT_flcm_chkzeros\expandafter#2%
    \romannumeral0\xintrez{\xintPIrr{\xintAbs{#1}}}\xint:#2%
}%
\def\XINT_flcmof_chkzero #1%
{%
    \xint_gob_til_zero#1\XINT_flcmof_zero0\XINT_flcmof_loop#1%
}%
\def\XINT_flcmof_zero#1^{ 0/1[0]}%
```

25.75. Floating point macros

ATTENTION THE COMMENTS HERE ARE COMPLETELY OBSOLETE SINCE THE ADVANT OF CALX. THEY WILL BE REMOVED WHEN VERSION 2.0a1 WIL GET FINALIZED FOR RELEASE. DEFINITION OF \xintDigits and \xintSetDigits

ARE ALREADY MOVED TO calxfp. THE COMMENTED OUT PREPARATION FOR A USER INTERAFACE FOR THE EXTRA GUARD DIGITES USED INTERNALLY FOR xintlog AND xintirg WAS DELETED DURING THIS TRANSFER.

For a long time the float routines dating back to releases 1.07/1.08a (May-June 2013) were not modified.

Since 1.2f (March 2016) the four operations first round their arguments to \xinttheDigits-floats (or P-floats), not (\xinttheDigits+2)-floats or (P+2)-floats as was the case with earlier releases.

The four operations addition, subtraction, multiplication, division have always produced the correct rounding of the theoretical exact value to P or \xinttheDigits digits when the inputs are decimal numbers with at most P digits, and arbitrary decimal exponent part.

From 1.08a to 1.2j, \xintFloat (and \XINTinFloat which is used to parse inputs to other float macros) handled a fractional input A/B via an initial replacement to A'/B' where A' and B' were A and B truncated to Q+2 digits (where asked-for precision is Q), and then they correctly rounded A'/B' to Q digits. But this meant that this rounding of the input could differ (by up to one unit in the last place) from the correct rounding of the original A/B to the asked-for number of digits (which until 1.2f in uses as auxiliary to the macros for the basic operations was 2 more than the prevailing precision).

Since 1.2k all inputs are correctly rounded to the asked-for number of digits (this was, I think, the case in the 1.07 release -- there are no code comments -- but was, afaicr, not very efficiently done, and this is why the 1.08a release opeted for truncation of the numerator and denominator.)

Notice that in float expressions, the / is treated as operator, hence the above discussion makes a difference only for the special input form qfloat(A/B) or for an \xintexpr A/B\relax embedded in the float expression, with A or B having more digits than the prevailing float precision.

> Internally there is no inner representation of P-floats as such !!!!!
>
> The input parser will again compute the length of the mantissa on each use !!! This is obviously something that must be improved upon before implementation of higher functions.
>
> Currently, special tricks are used to quickly recognize inputs having no denominators, or fractions whose numerators and denominators are not too long compared to the target precision P, and in particular P-floats or quotients of two such.
>
> Another long-standing issue is that float multiplication will first compute the 2P or 2P-1 digits of the exact product, and then round it to P digits. This is sub-optimal for large P particularly as the multiplication algorithm is basically the schoolbook one, hence *worse* than quadratic in the TEX implementation which has extra cost of fetching long sequences of tokens.

Changes at 1.4e (done 2021/04/15; undone 2021/04/29)

Macros named \XINTinFloat<name> are not public user-level but were designed a long time ago for \xintfloatexpr context as a very preliminary step towards attempting to preserve some internal format, here A[N] type.

When <name> is lowercased it means it needs a \romannumeral0 trigger (\XINTinfloatS keeps an uppercase S).

Most were coded to check for an optional argument [D], and to use D=\XINTdigits in its place if absent but it turned out only \XINTinfloatpow, \XINTinfloatmul, \XINTinfloatadd were actually used with an optional argument and this happened only in macros from the very old xintseries.sty, so I changed all of them to not check for optional argument [D] anymore, keeping only some private interface for the xintseries.sty use case. Some required being used with [D], some still had names ending in "digits" indicating they would use \XINTdigits always.

Indeed basically all algebra is done "exactly" and the [D] governs rules of float-rounding on input and output.

During development of 1.4e we fleetingly experimented with letting the value used in place of D be \XINTdigitsx to 1.4e, i.e. \XINTdigits with guard digits, a situation which was motivated

by the implementation of trigonometrical functions at high level, i.e. using \xintdeffloatfunc which had no mechanism to make intermediate calculations with guard digits.

Simply doing everything "as is" but with 2 guard digits proved very good (surprisingly efficient, even) to the trigonometrical functions. However using them systematically raises many issues (for example, the correct rounding at P digits is destroyed if we obtain it a D=P+2 then round from P+2 to P digits so we definitely can not do this as default, so some interface is needed to define intermediate functions only using such guard digits and keeping them in their output).

Finally, an approach limited to the xinttrig.sty scope was used and I removed all \XINTdigits related matters from 1.4e. But this left some modifications of the interfaces of the "float" macros here which this list tries to document, mainly for the author's benefit.

Macros always using \XINTdigits and now not allowing [P] option
\XINTinFloatAdd
\XINTinFloatSub
\XINTinFloatMul
\XINTinFloatSqr
\XINTinFloatDiv
\XINTinFloatPow
\XINTinFloatPower
\XINTinFloatPFactorial
\XINTinFloatBinomial
Macros which already did not allow [P] option prior to 1.4e refactoring
\XINTinFloatFrac (renamed from \XINTinFloatFracdigits)
Macros requiring a [P]. Some of the "_wopt" named macros are renamings of macros formerly requiring [P].
\XINTinFloat
\XINTinFloatS
\XINTFloatiLogTen
\XINTinRandomFloatS (this one has only the [P] mandatory argument)
\XINTinFloatFac
\XINTinFloatSqrt
\XINTinFloatAdd_wopt, \XINTinfloatadd_wopt
\XINTinFloatSub_wopt, \XINTinfloatsub_wopt
\XINTinFloatMul_wopt, \XINTinfloatmul_wopt
\XINTinFloatSqr_wopt
\XINTinfloatpow_wopt (not FloatPow)
\XINTinFloatDiv_wopt
Specially named macros indicating usage of \XINTdigits
\XINTinFloatdigits
\XINTinFloatSdigits
\XINTFloatiLogTendigits
\XINTinRandomFloatSdigits
\XINTinFloatFacdigits
\XINTinFloatSqrtdigits

25.76. \xintFloat, \xintFloatZero

1.2f and 1.2g brought some refactoring which resulted in faster treatment of decimal inputs. 1.2i dropped use of some old routines dating back to pre 1.2 era in favor of more modern \xintDSRr for rounding. Then 1.2k improves again the handling of denominators B with few digits.

But the main change with 1.2k is a complete rewrite of the B>1 case in order to achieve again correct rounding in all cases.

The original version from 1.07 (May 2013) computed the exact rounding to P digits for all inputs. But from 1.08 on (June 2013), the macro handled A/B input by first truncating both A and B to at most

P+2 digits. This meant that decimal input (arbitrarily long, with scientific part) was correctly rounded, but in case of fractional input there could be up to 0.6 unit in the last place difference of the produced rounding to the input, hence the output could differ from the correct rounding.

Example with 16 digits (the default): \xintFloat {1/17597472569900621233}
with xintfrac 1.07: 5.682634230727187e-20
with xintfrac 1.08b--1.2j: 5.682634230727188e-20
with xintfrac 1.2k: 5.682634230727187e-20
The exact value is 5.6826342307271874999241 24...e-20, showing that 1.07 and 1.2k produce the correct rounding.

Currently the code ends in a more costly branch in about 1 case among 500, where it does some extra operations (a multiplication in particular). There is a free parameter delta (here set at 4), I have yet to make some numerical explorations, to see if it could be favorable to set it to a higher value (with delta=5, there is only 1 exceptional case in 5000, etc...).

I have always hesitated about the policy of printing 10.00...0 in case of rounding upwards to the next power of ten. Already since 1.2f \XINTinFloat always produced a mantissa with exactly P digits (except for the zero value). Starting with 1.2k, \xintFloat drops this habit of printing 10.00..0 in such cases. Side note: the rounding-up detection worked when the input A/B was with numerator A and denominator B having each less than P+2 digits, or with B=1, else, it could happen that the output was a power of ten but not detected to be a rounding up of the original fraction. The value was ok, but printed 1.0...0eN with P-1 zeroes, not 10.0...0e(N-1).

I decided it was not worth the effort to enhance the algorithm to detect with 100% fiability all cases of rounding up to next power of ten, hence 1.2k dropped this.

To avoid duplication of code, and any extra burden on \XINTinFloat, which is the macro used internally by the float macros for parsing their inputs, we simply make now \xintFloat a wrapper of \XINTinFloat.

```
\def\xintFloatZero{0.0e0}% 1.4k breaking change.  Replaces hard-coded 0.e0
\def\xintFloat   {\romannumeral0\xintfloat }%
\def\xintfloat #1{\XINT_float_chkopt #1\xint:}%
\def\XINT_float_chkopt #1%
{%
    \ifx [#1\expandafter\XINT_float_opt
       \else\expandafter\XINT_float_noopt
    \fi  #1%
}%
\def\XINT_float_noopt #1\xint:%
{%
    \expandafter\XINT_float_post
    \romannumeral0\XINTinfloat[\XINTdigits]{#1}\XINTdigits.%
}%
\def\XINT_float_opt [\xint:
{%
    \expandafter\XINT_float_opt_a\the\numexpr
}%
\def\XINT_float_opt_a #1]#2%
{%
    \expandafter\XINT_float_post
    \romannumeral0\XINTinfloat[#1]{#2}#1.%
}%
\def\XINT_float_post #1%
{%
    \xint_UDzerominusfork
      #1-\XINT_float_zero
```

```
        0#1\XINT_float_neg
        0-\XINT_float_pos
    \krof #1%
}%[
\def\XINT_float_zero #1]#2.{\expanded{ \xintFloatZero}}%
\def\XINT_float_neg-{\expandafter-\romannumeral0\XINT_float_pos}%
\def\XINT_float_pos #1#2[#3]#4.%
{%
    \expandafter\XINT_float_pos_done\the\numexpr#3+#4-\xint_c_i.#1.#2;%
}%
\def\XINT_float_pos_done #1.#2;{ #2e#1}%
```

25.77. \xintFloatBraced

Added at 1.4l (2022/05/29). Je ne le fais pas comme un wrapper au-dessus de \xintFloat car c'est pénible avec argument optionnel donc finalemnt on est obligé de rajouter overhead comme ici.
Hésitation si on obéit à \xintFloatZero ou pas. Finalement non.
Hésitation si on renvoie avec séparateur décimal ou pas.
Hésitation si on met l'exposant scientifique en premier.
Hésitation si on sépare le signe pour le mettre en premier.
Hésitation si on renvoie un exposant pour mantisse normalisée ou pas normalisée.
Finalement je décide {signe}{exposant}{mantisse sans point décimal}. Avec en fait 0 ou 1 pour signe (mais ce sign bit mais ça n'a pas grand sens en décimal...). Non finalement mantisse avec point décimal.

```
\def\xintFloatBraced{\romannumeral0\xintfloatbraced }%
\def\xintfloatbraced#1{\XINT_floatbr_chkopt #1\xint:}%
\def\XINT_floatbr_chkopt #1%
{%
    \ifx [#1\expandafter\XINT_floatbr_opt
       \else\expandafter\XINT_floatbr_noopt
    \fi  #1%
}%
\def\XINT_floatbr_noopt #1\xint:%
{%
    \expandafter\XINT_floatbr_post
    \romannumeral0\XINTinfloat[\XINTdigits]{#1}\XINTdigits.%
}%
\def\XINT_floatbr_opt [\xint:
{%
    \expandafter\XINT_floatbr_opt_a\the\numexpr
}%
\def\XINT_floatbr_opt_a #1]#2%
{%
    \expandafter\XINT_floatbr_post
    \romannumeral0\XINTinfloat[#1]{#2}#1.%
}%
\def\XINT_floatbr_post #1%
{%
    \xint_UDzerominusfork
      #1-\XINT_floatbr_zero
       0#1\XINT_floatbr_neg
       0-\XINT_floatbr_pos
```

```
    \krof #1%
}%
```

Hésitation à faire

```
   \def\XINT_floatbr_zero #1]#2.{\expandafter\XINT_floatbr_zero_a\xintFloatZero e0e\relax}
   \def\XINT_floatbr_zero_a#1e#2e#3\relax{{#1}{#2}}
```

Finalement non. Et même je décide de renvoyer autant de zéros que P. De plus depuis j'ai opté pour {sign bit}{exposant}{mantisse} Hésitation si mantisse avec ou sans le séparateur décimal. Est-ce que je devrais mettre plutôt -0+ au début?

```
\def\XINT_floatbr_zero #1]#2.{\expanded{{0}{0}{0.\xintReplicate{#2-\xint_c_i}0}}}%
\def\XINT_floatbr_neg-{\expandafter\XINT_floatbr_neg_a\romannumeral0\XINT_floatbr_pos}%
\def\XINT_floatbr_neg_a#1{{1}}%
\def\XINT_floatbr_pos #1#2[#3]#4.%
{%
    \expanded{{0}{\the\numexpr#3+#4-\xint_c_i}}{#1.#2}%
}%
```

25.78. \XINTinFloat, \XINTinFloatS

This routine is like \xintFloat but produces an output of the shape A[N] which is then parsed faster as input to other float macros. Float operations in \xintfloatexpr...\relax use internally this format.

It must be used in form \XINTinFloat[P]{f}: the optional [P] is mandatory.

Since 1.2f, the mantissa always has exactly P digits even in case of rounding up to next power of ten. This simplifies other routines.

(but the zero value must always be checked for, as it outputs 0[0])

1.2g added a variant \XINTinFloatS which, in case of decimal input with less than the asked for precision P will not add extra zeros to the mantissa. For example it may output 2[0] even if P=500, rather than the canonical representation 200...000[-499]. This is how \xintFloatMul and \xintFloatDiv parse their inputs, which speeds-up follow-up processing. But \xintFloatAdd and \xintFloatSub still use \XINTinFloat for parsing their inputs; anyway this will have to be changed again when inner structure will carry upfront at least the length of mantissa as data.

Each time \XINTinFloat is called it at least computes a length. Naturally if we had some format for floats that would be dispensed of...
something like <letterP><length of mantissa>.mantissa.exponent, etc... not yet.

Since 1.2k, \XINTinFloat always correctly rounds its argument, even if it is a fraction with very big numerator and denominator. See the discussion of \xintFloat.

```
\def\XINTinFloat {\romannumeral0\XINTinfloat }%
\def\XINTinfloat
   {\expandafter\XINT_infloat_clean\romannumeral0\XINT_infloat}%
```

Attention que ici le fait que l'on grabbe #1 est important car il pourrait y avoir un zéro (en particulier dans le cas où input est nul).

```
\def\XINT_infloat_clean #1%
   {\if #1!\xint_dothis\XINT_infloat_clean_a\fi\xint_orthat{ }#1}%
```

Ici on ajoute les zeros pour faire exactement avec P chiffres. Car le #1 = P - L avec L la longueur de #2, (ou plutôt de abs(#2), car ici le #2 peut avoir un signe) et L < P

```
\def\XINT_infloat_clean_a !#1.#2[#3]%
{%
    \expandafter\XINT_infloat_done
    \the\numexpr #3-#1\expandafter.%
    \romannumeral0\XINT_dsx_addzeros {#1}#2;;%
}%
```

```
\def\XINT_infloat_done #1.#2;{ #2[#1]}%
```

variant which allows output with shorter mantissas.

```
\def\XINTinFloatS {\romannumeral0\XINTinfloatS}%
\def\XINTinfloatS
   {\expandafter\XINT_infloatS_clean\romannumeral0\XINT_infloat}%
\def\XINT_infloatS_clean #1%
   {\if #1!\xint_dothis\XINT_infloatS_clean_a\fi\xint_orthat{ }#1}%
\def\XINT_infloatS_clean_a !#1.{ }%
```

début de la routine proprement dite, l'argument optionnel est obligatoire.

```
\def\XINT_infloat [#1]%#2%
{%
    \expandafter\XINT_infloat_a\the\numexpr #1\expandafter.%
    \romannumeral0\XINT_infrac% {#2}%
}%
```

#1=P, #2=n, #3=A, #4=B.

```
\def\XINT_infloat_a #1.#2#3#4%
{%
```

micro boost au lieu d'utiliser \XINT_isOne{#4}, mais pas bon style.

```
    \if1\XINT_is_One#4XY%
      \expandafter\XINT_infloat_sp
    \else\expandafter\XINT_infloat_fork
    \fi #3.{#1}{#2}{#4}%
}%
```

Special quick treatment of B=1 case (1.2f then again 1.2g.)
maintenant: A.{P}{N}{1} Il est possible que A soit nul.

```
\def\XINT_infloat_sp #1%
{%
    \xint_UDzerominusfork
     #1-\XINT_infloat_spzero
     0#1\XINT_infloat_spneg
      0-\XINT_infloat_sppos
    \krof #1%
}%
```

Attention surtout pas 0/1[0] ici.

```
\def\XINT_infloat_spzero 0.#1#2#3{ 0[0]}%
\def\XINT_infloat_spneg-%
    {\expandafter\XINT_infloat_spnegend\romannumeral0\XINT_infloat_sppos}%
\def\XINT_infloat_spnegend #1%
    {\if#1!\expandafter\XINT_infloat_spneg_needzeros\fi -#1}%
\def\XINT_infloat_spneg_needzeros -!#1.{!#1.-}%
```

in: A.{P}{N}{1}
out: P-L.A.P.N.

```
\def\XINT_infloat_sppos #1.#2#3#4%
{%
    \expandafter\XINT_infloat_sp_b\the\numexpr#2-\xintLength{#1}.#1.#2.#3.%
}%
```

#1= P-L. Si c'est positif ou nul il faut retrancher #1 à l'exposant, et ajouter autant de zéros.
On regarde premier token. P-L.A.P.N.

```
\def\XINT_infloat_sp_b #1%
```

```
{%
    \xint_UDzerominusfork
     #1-\XINT_infloat_sp_quick
     0#1\XINT_infloat_sp_c
      0-\XINT_infloat_sp_needzeros
    \krof #1%
}%
```

Ici P=L. Le cas usuel dans \xintfloatexpr.

```
\def\XINT_infloat_sp_quick 0.#1.#2.#3.{ #1[#3]}%
```

Ici #1=P-L est >0. L'exposant sera N-(P-L). #2=A. #3=P. #4=N.
18 mars 2016. En fait dans certains contextes il est sous-optimal d'ajouter les zéros. Par exemple quand c'est appelé par la multiplication ou la division, c'est idiot de convertir 2 en 200000...00000[-499]. Donc je redéfinis addzeros en needzeroes. Si on appelle sous la forme \XINTinFloatS, on ne fait pas l'addition de zeros.

```
\def\XINT_infloat_sp_needzeros #1.#2.#3.#4.{!#1.#2[#4]}%
```

L-P=#1.A=#2#3.P=#4.N=#5.
Ici P<L. Il va falloir arrondir. Attention si on va à la puissance de 10 suivante. En #1 on a L-P qui est >0. L'exposant final sera N+L-P, sauf dans le cas spécial, il sera alors N+L-P+1. L'ajustement final est fait par \XINT_infloat_Y.

```
\def\XINT_infloat_sp_c -#1.#2#3.#4.#5.%
{%
    \expandafter\XINT_infloat_Y
    \the\numexpr #5+#1\expandafter.%
    \romannumeral0\expandafter\XINT_infloat_sp_round
    \romannumeral0\XINT_split_fromleft
    (\xint_c_i+#4).#2#3\xint_bye2345678\xint_bye..#2%
}%
\def\XINT_infloat_sp_round #1.#2.%
{%
    \XINT_dsrr#1\xint_bye\xint_Bye3456789\xint_bye/\xint_c_x\relax.%
}%
```

General branch for A/B with B>1 inputs. It achieves correct rounding always since 1.2k (done January 2, 2017.) This branch is never taken for A=0 because \XINT_infrac will have returned B=1 then.

```
\def\XINT_infloat_fork #1%
{%
    \xint_UDsignfork
     #1\XINT_infloat_J
     -\XINT_infloat_K
    \krof #1%
}%
\def\XINT_infloat_J-{\expandafter-\romannumeral0\XINT_infloat_K }%
```

A.{P}{n}{B} avec B>1.

```
\def\XINT_infloat_K #1.#2%
{%
    \expandafter\XINT_infloat_L
    \the\numexpr\xintLength{#1}\expandafter.\the\numexpr #2+\xint_c_iv.{#1}{#2}%
}%
```

|A|.P+4.{A}{P}{n}{B}. We check if A already has length <= P+4.

```
\def\XINT_infloat_L #1.#2.%
```

```
{%
    \ifnum #1>#2
      \expandafter\XINT_infloat_Ma
    \else
      \expandafter\XINT_infloat_Mb
    \fi #1.#2.%
}%
```

```
|A|.P+4.{A}{P}{n}{B}.  We will keep only the first P+4 digits of A, denoted A'' in what follows.
  output:  u=-0.A''.junk.P+4.|A|.{A}{P}{n}{B}
```

```
\def\XINT_infloat_Ma #1.#2.#3%
{%
    \expandafter\XINT_infloat_MtoN\expandafter-\expandafter0\expandafter.%
    \romannumeral0\XINT_split_fromleft#2.#3\xint_bye2345678\xint_bye..%
    #2.#1.{#3}%
}%
```

```
|A|.P+4.{A}{P}{n}{B}.
Here A is short.  We set u = P+4-|A|, and A''=A (A' = 10^u A)
  output:  u.A''..P+4.|A|.{A}{P}{n}{B}
```

```
\def\XINT_infloat_Mb #1.#2.#3%
{%
    \expandafter\XINT_infloat_MtoN\the\numexpr#2-#1.%
    #3..#2.#1.{#3}%
}%
```

```
input u.A''.junk.P+4.|A|.{A}{P}{n}{B}
output |B|.P+4.{B}u.A''.P.|A|.n.{A}{B}
```

```
\def\XINT_infloat_MtoN #1.#2.#3.#4.#5.#6#7#8#9%
{%
   \expandafter\XINT_infloat_N
   \the\numexpr\xintLength{#9}.#4.{#9}#1.#2.#7.#5.#8.{#6}{#9}%
}%
\def\XINT_infloat_N #1.#2.%
{%
    \ifnum #1>#2
      \expandafter\XINT_infloat_Oa
    \else
      \expandafter\XINT_infloat_Ob
    \fi #1.#2.%
}%
```

```
input |B|.P+4.{B}u.A''.P.|A|.n.{A}{B}
output v=-0.B''.junk.|B|.u.A''.P.|A|.n.{A}{B}
```

```
\def\XINT_infloat_Oa #1.#2.#3%
{%
    \expandafter\XINT_infloat_P\expandafter-\expandafter0\expandafter.%
    \romannumeral0\XINT_split_fromleft#2.#3\xint_bye2345678\xint_bye..%
    #1.%
}%
```

```
output v=P+4-|B|>=0.B''.junk.|B|.u.A''.P.|A|.n.{A}{B}
```

```
\def\XINT_infloat_Ob #1.#2.#3%
{%
    \expandafter\XINT_infloat_P\the\numexpr#2-#1.#3..#1.%
}%
```

input v.B''.junk.|B|.u.A''.P.|A|.n.{A}{B}
output Q1.P.|B|.|A|.n.{A}{B}
Q1 = division euclidienne de A''.10^{u-v+P+3} par B''.
  Special detection of cases with A and B both having length at most P+4: this will happen when called from \xintFloatDiv as A and B (produced then via \XINTinFloatS) will have at most P digits. We then only need integer division with P+1 extra zeros, not P+3.

```
\def\XINT_infloat_P #1#2.#3.#4.#5.#6#7.#8.#9.%
{%
   \csname XINT_infloat_Q\if-#1\else\if-#6\else q\fi\fi\expandafter\endcsname
   \romannumeral0\xintiiquo
   {\romannumeral0\XINT_dsx_addzerosnofuss
      {#6#7-#1#2+#9+\xint_c_iii\if-#1\else\if-#6\else-\xint_c_ii\fi\fi}#8;}%
   {#3}.#9.#5.%
}%
```

«quick» branch.

```
\def\XINT_infloat_Qq #1.#2.%
{%
    \expandafter\XINT_infloat_Rq
    \romannumeral0\XINT_split_fromleft#2.#1\xint_bye2345678\xint_bye..#2.%
}%
\def\XINT_infloat_Rq #1.#2#3.%
{%
    \ifnum#2<\xint_c_v
         \expandafter\XINT_infloat_SEq
    \else\expandafter\XINT_infloat_SUp
    \fi
    {\if.#3.\xint_c_\else\xint_c_i\fi}#1.%
}%
```

standard branch which will have to handle undecided rounding, if too close to a mid-value.

```
\def\XINT_infloat_Q #1.#2.%
{%
    \expandafter\XINT_infloat_R
    \romannumeral0\XINT_split_fromleft#2.#1\xint_bye2345678\xint_bye..#2.%
}%
\def\XINT_infloat_R #1.#2#3#4#5.%
{%
    \if.#5.\expandafter\XINT_infloat_Sa\else\expandafter\XINT_infloat_Sb\fi
    #2#3#4#5.#1.%
}%
```

trailing digits.Q.P.|B|.|A|.n.{A}{B}
#1=trailing digits (they may have leading zeros.)

```
\def\XINT_infloat_Sa #1.%
{%
    \ifnum#1>500 \xint_dothis\XINT_infloat_SUp\fi
    \ifnum#1<499 \xint_dothis\XINT_infloat_SEq\fi
    \xint_orthat\XINT_infloat_X\xint_c_
}%
\def\XINT_infloat_Sb #1.%
{%
    \ifnum#1>5009 \xint_dothis\XINT_infloat_SUp\fi
    \ifnum#1<4990 \xint_dothis\XINT_infloat_SEq\fi
```

```
    \xint_orthat\XINT_infloat_X\xint_c_i
}%
```

epsilon #2=Q.#3=P.#4=|B|.#5=|A|.#6=n.{A}{B}
exposant final est n+|A|-|B|-P+epsilon

```
\def\XINT_infloat_SEq #1#2.#3.#4.#5.#6.#7#8%
{%
    \expandafter\XINT_infloat_SY
    \the\numexpr #6+#5-#4-#3+#1.#2.%
}%
\def\XINT_infloat_SY #1.#2.{ #2[#1]}%
```

initial digit #2 put aside to check for case of rounding up to next power of ten, which will need adjustment of mantissa and exponent.

```
\def\XINT_infloat_SUp #1#2#3.#4.#5.#6.#7.#8#9%
{%
    \expandafter\XINT_infloat_Y
    \the\numexpr#7+#6-#5-#4+#1\expandafter.%
    \romannumeral0\xintinc{#2#3}.#2%
}%
```

epsilon Q.P.|B|.|A|.n.{A}{B}

\xintDSH{-x}{U} multiplies U by 10^x. When x is negative, this means it truncates (i.e. it drops the last -x digits).

We don't try to optimize too much macro calls here, the odds are 2 per 1000 for this branch to be taken. Perhaps in future I will use higher free parameter d, which currently is set at 4.

#1=epsilon, #2#3=Q, #4=P, #5=|B|, #6=|A|, #7=n, #8=A, #9=B

```
\def\XINT_infloat_X #1#2#3.#4.#5.#6.#7.#8#9%
{%
   \expandafter\XINT_infloat_Y
   \the\numexpr #7+#6-#5-#4+#1\expandafter.%
   \romannumeral`&&@\romannumeral0\xintiiiflt
     {\xintDSH{#6-#5-#4+#1}{\xintDouble{#8}}}%
     {\xintiiMul{\xintInc{\xintDouble{#2#3}}}{#9}}%
   \xint_firstofone
   \xintinc{#2#3}.#2%
}%
```

check for rounding up to next power of ten.

```
\def\XINT_infloat_Y #1{%
\def\XINT_infloat_Y ##1.##2##3.##4%
{%
   \if##49\if##21\expandafter\expandafter\expandafter\XINT_infloat_Z\fi\fi
   #1##2##3[##1]%
}}\XINT_infloat_Y{ }%
```

#1=1, #2=0.

```
\def\XINT_infloat_Z #1#2#3[#4]%
{%
    \expandafter\XINT_infloat_ZZ\the\numexpr#4+\xint_c_i.#3.%
}%
\def\XINT_infloat_ZZ #1.#2.{ 1#2[#1]}%
```

25.79. \XINTFloatiLogTen

Added at 1.3e (2019/04/05). Le comportement pour un input nul est non encore finalisé. Il changera lorsque NaN, +Inf, -Inf existeront.

The optional argument [#1] is in fact mandatory and #1 is not pre-expanded in a \numexpr.

The return value here 2^31-2^15 is highly undecided.

```
\def\XINTFloatiLogTen {\the\numexpr\XINTfloatilogten}%
\def\XINTfloatilogten [#1]#2%
    {\expandafter\XINT_floatilogten\romannumeral0\XINT_infloat[#1]{#2}#1.}%
\def\XINTFloatiLogTendigits{\the\numexpr\XINTfloatilogten[\XINTdigits]}%
\def\XINT_floatilogten #1{%
    \if #10\xint_dothis\XINT_floatilogten_z\fi
    \if #1!\xint_dothis\XINT_floatilogten_a\fi
    \xint_orthat\XINT_floatilogten_b #1%
}%
\def\XINT_floatilogten_z 0[0]#1.{-"7FFF8000\relax}%
\def\XINT_floatilogten_a !#1.#2[#3]#4.{#3-#1+#4-\xint_c_i\relax}%
\def\XINT_floatilogten_b #1[#2]#3.{#2+#3-\xint_c_i\relax}%
```

25.80. \xintPFloat

Added at 1.1 (2014/10/28).

Modified at 1.4e (2021/05/05).

xint has not yet incorporated a general formatter as it was not a priority during development and external solutions exist (I did not check for a while but I think LaTeX3 has implemented a general formatter in the printf or Python ".format" spirit)

But when one starts using really the package, especially in an interactive way (xintsession 2021), one needs the default output to be as nice as possible.

The \xintPFloat macro was added at 1.1 as a "prettifying printer" for floats, basically influenced by Maple.

The rules were:

0. The input is float-rounded to either Digits or the optional argument
1. zero is printed as "0."
2. x.yz...eK is printed "as is" if K>5 or K<-5.
3. if -5<=K<=5, fixed point decimal notation is used.
4. in cases 2. and 3., no trimming of trailing zeroes.

1.4b added \xintPFloatE to customize whether to use e or E.

1.4e, with some hesitation, decided to make a breaking change and to modify the behaviour.

The new rules:

0. The input is float-rounded to either Digits or the optional argument
1. zero is printed as 0.0
2. x.yz...eK is printed in decimal fixed point if -4<=K<=+5 (notice the change, formerly K=-5 used fixed point notation in output) else it is printed in scientific notation
3. trailing zeros of the mantissa are trimmed always
4. in case of decimal fixed point for an integer, there is a trailing ".0"
5. in case of scientific notation with a one-digit trimmed mantissa there is an added ".0" too

Further, \xintPFloatE can now also be redefined as a macro with a parameter delimited by a dot, with the dot also present as final token in its output. It would then grab the scientific exponent K as explicit digit possibly prefixed by a minus sign. The macro must be f-expandable.

The macro \xintPFloat_wopt is only there for a micro gain as the package does

```
\let\xintfloatexprPrintOne\xintPFloat_wopt
```

as it knows it will be used always with a [P] argument in the xintexpr.sty context.

Modified at 1.4k (2022/05/18). Addition of customization via \xintPFloatZero, \xintPFloatLengthOneSuffix, \xintPFloatNoSciEmax, \xintPFloatNoSciEmin which replace formerly hard-coded behaviour.

Breaking change to not add ".0" suffix to integers (when scientific notation dropped) or to one-digit mantissas.

In my own practice I started being annoyed by the automatic trimming of zeros added at 1.4e.

This change had been influenced by using Python in interactive mode which since 3.1 prints floats (in decimal conversion) choosing the shortest string. In particular it trims trailing zeros, and it drops the scientific notation in favor of decimal notation for something like -4<= K <= 15, with K the scientific exponent.

At 1.4e I was still influenced by my experience with Maple and did for -4 <= K <= 5. Not very well thought anyhow (one may wish to use decimal notation when sending things to PostScript, so perhaps I should have kept with -5).

But, the main problem is with trimming trailing zeros: although in interactive sessions, this has its logic, as soon as one does tables with numbers, dropping a trailing zero upsets alignments or creates visual holes compared to other lines and this is in the end very annoying.

After much hesitation, I decided to slightly modifify only the former behaviour: trimming only if that removes at least 4 zeros. I had also experimented with another condition: trimmed mantissas should be at most 6 digits (for example) wide, else use no trimming.

Threshold customizable via \xintPFloatMinTrimmed.

Modified at 1.4l (2022/05/29). The 1.4k check for canceling the trimming of trailing zeros took over priority over the later check for being an integer when decimal fixed point notation was used (or being only with a one-digit trimmed mantissa). In particular if user set \xintPFloatMinTrimmed to the value of Digits (or P) to avoid trimming it also prevented recognition of some integers (but not all). Fixed at 1.4l

Modified at 2.0a1 (2026/11/01). The formatting core (\XINT_pfloat_fork to \XINT_pfloat_Ps, with \XINT_pfloat_a_fork) and the defaults of \xintPFloatE, \xintPFloatNoSciEmax, \xintPFloatNoSciEmin, \xintPFloatIntSuffix, \xintPFloatLengthOneSuffix, \xintPFloatZero, \xintPFloatMinTrimmed moved verbatim to calxfp.sty at 2.0a1.

```
\def\xintPFloat {\romannumeral0\xintpfloat }%
\def\xintpfloat #1{\XINT_pfloat_chkopt #1\xint:}%
\def\xintPFloat_wopt[#1]#2%
{%
    \romannumeral0\expandafter\XINT_pfloat
    \romannumeral0\XINTinfloatS[#1]{#2}#1.%
}%
\def\XINT_pfloat_chkopt #1%
{%
    \ifx [#1\expandafter\XINT_pfloat_opt
       \else\expandafter\XINT_pfloat_noopt
    \fi  #1%
}%
\def\XINT_pfloat_noopt #1\xint:%
{%
    \expandafter\XINT_pfloat\romannumeral0\XINTinfloatS[\XINTdigits]{#1}%
    \XINTdigits.%
}%
\def\XINT_pfloat_opt [\xint:{\expandafter\XINT_pfloat_opt_a\the\numexpr}%
\def\XINT_pfloat_opt_a #1]#2%
{%
    \expandafter\XINT_pfloat\romannumeral0\XINTinfloatS[#1]{#2}%
    #1.%
```

```
}%
\def\XINT_pfloat#1]%
{%
    \expandafter\XINT_pfloat_fork\romannumeral0\xintrez{#1]}%
}%
```

25.81. \xintFloatToDecimal

Added at 1.4k (2022/05/18).

```
\def\xintFloatToDecimal {\romannumeral0\xintfloattodecimal }%
\def\xintfloattodecimal #1{\XINT_floattodec_chkopt #1\xint:}%
\def\XINT_floattodec_chkopt #1%
{%
    \ifx [#1\expandafter\XINT_floattodec_opt
       \else\expandafter\XINT_floattodec_noopt
    \fi  #1%
}%
\def\XINT_floattodec_noopt #1\xint:%
{%
    \expandafter\XINT_floattodec\romannumeral0\XINTinfloatS[\XINTdigits]{#1}%
}%
\def\XINT_floattodec_opt [\xint:#1]%
{%
    \expandafter\XINT_floattodec\romannumeral0\XINTinfloatS[#1]%
}%
```

Temptation to try to use direct access to lower entry points from \xintREZ, but it dates back from very early days and uses old \Z delimiters (same remarks for the code jumping from \xintFracToSci to \xintrez)

```
\def\XINT_floattodec#1]%
{%
    \expandafter\XINT_dectostr\romannumeral0\xintrez{#1]}%
}%
```

25.82. \xintFloatAdd, \XINTinFloatAdd

The \XINTinFloatAdd interface is kept at 2.0a1 for polexpr. And \XINTinfloatadd is kept for xintseries.

First included in release 1.07.

1.09ka improved a bit the efficiency. However the add, sub, mul, div routines were provisory and supposed to be revised soon.

Which didn't happen until 1.2f. Now, the inputs are first rounded to P digits, not P+2 as earlier.

See general introduction for important changes at 1.4e relative to the \XINTinFloat<name> macros.

```
\def\xintFloatAdd  {\romannumeral0\xintfloatadd}%
\def\xintfloatadd  #1{\XINT_fladd_chkopt \xintfloat #1\xint:}%
\def\XINTinFloatAdd{\romannumeral0\XINTinfloatadd }%
\def\XINTinfloatadd{\XINT_fladd_opt_a\XINTdigits.\XINTinfloatS}%
\def\XINTinfloatadd_wopt[#1]{\expandafter\XINT_fladd_opt_a\the\numexpr#1.\XINTinfloatS}%
\def\XINT_fladd_chkopt #1#2%
{%
    \ifx [#2\expandafter\XINT_fladd_opt
```

```
        \else\expandafter\XINT_fladd_noopt
    \fi  #1#2%
}%
\def\XINT_fladd_noopt #1#2\xint:#3%
{%
    #1[\XINTdigits]%
    {\expandafter\XINT_FL_add_a
     \romannumeral0\XINTinfloat[\XINTdigits]{#2}\XINTdigits.{#3}}%
}%
\def\XINT_fladd_opt #1[\xint:#2]%#3#4%
{%
    \expandafter\XINT_fladd_opt_a\the\numexpr #2.#1%
}%
\def\XINT_fladd_opt_a #1.#2#3#4%
{%
    #2[#1]{\expandafter\XINT_FL_add_a\romannumeral0\XINTinfloat[#1]{#3}#1.{#4}}%
}%
\def\XINT_FL_add_a #1%
{%
    \xint_gob_til_zero #1\XINT_FL_add_zero 0\XINT_FL_add_b #1%
}%
\def\XINT_FL_add_zero #1.#2{#2}%[[
\def\XINT_FL_add_b #1]#2.#3%
{%
    \expandafter\XINT_FL_add_c\romannumeral0\XINTinfloat[#2]{#3}#2.#1]%
}%
\def\XINT_FL_add_c #1%
{%
    \xint_gob_til_zero #1\XINT_FL_add_zero 0\XINT_FL_add_d #1%
}%
```

Modified at 1.4p (2026/03/01). Fix long-standing regression which was introduced at 1.2f (2016/03/12).

The bug caused the rounding to not be the correct one if one argument x (after initial rounding to P digits) is +-10^n and the other y has opposite sign, scientific exponent exactly n-P-1, and |y| > 5 10^{n-P-1}. Affects \xintFloatAdd and \xintFloatSub, and also of course the \xintexpr context. Fixed by testing >1 and not>0 for the short path.

```
\def\XINT_FL_add_d #1[#2]#3.#4[#5]%
{%
    \ifnum\numexpr #2-#3-#5>\xint_c_i\xint_dothis\xint_firstoftwo\fi
    \ifnum\numexpr #5-#3-#2>\xint_c_i\xint_dothis\xint_secondoftwo\fi
    \xint_orthat\xintAdd {#1[#2]}{#4[#5]}%
}%
```

25.83. \xintFloatSub, \XINTinFloatSub

Added at 1.07 (2013/05/25).

Modified at 1.2f (2016/03/12). Starting with 1.2f the arguments undergo an intial rounding to the target precision P not P+2.

```
\def\xintFloatSub  {\romannumeral0\xintfloatsub}%
\def\xintfloatsub  #1{\XINT_flsub_chkopt \xintfloat #1\xint:}%
\def\XINT_flsub_chkopt #1#2%
```

```
{%
    \ifx [#2\expandafter\XINT_flsub_opt
       \else\expandafter\XINT_flsub_noopt
    \fi  #1#2%
}%
\def\XINT_flsub_noopt #1#2\xint:#3%
{%
    #1[\XINTdigits]%
    {\expandafter\XINT_FL_add_a
     \romannumeral0\XINTinfloat[\XINTdigits]{#2}\XINTdigits.{\xintOpp{#3}}}%
}%
\def\XINT_flsub_opt #1[\xint:#2]%#3#4%
{%
    \expandafter\XINT_flsub_opt_a\the\numexpr #2.#1%
}%
\def\XINT_flsub_opt_a #1.#2#3#4%
{%
    #2[#1]{\expandafter\XINT_FL_add_a\romannumeral0\XINTinfloat[#1]{#3}#1.{\xintOpp{#4}}}%
}%
```

25.84. \xintFloatMul, \XINTinFloatMul

The \XINTinFloatMul interface is kept at 2.0a1 for polexpr. And \XINTinfloatmul is kept for xintseries.

Added at 1.07 (2013/05/25).

Modified at 1.2d (2015/11/18). Starting with 1.2f the arguments are rounded to the target precision P not P+2.

Modified at 1.2g (2016/03/19). 1.2g handles the inputs via \XINTinFloatS which will be more efficient when the precision is large and the input is for example a small constant like 2.

```
\def\xintFloatMul  {\romannumeral0\xintfloatmul}%
\def\xintfloatmul  #1{\XINT_flmul_chkopt \xintfloat #1\xint:}%
\def\XINTinFloatMul{\romannumeral0\XINTinfloatmul}%
\def\XINTinfloatmul{\XINT_flmul_opt_a\XINTdigits.\XINTinfloatS}%
\def\XINTinfloatmul_wopt[#1]{\expandafter\XINT_flmul_opt_a\the\numexpr#1.\XINTinfloatS}%
\def\XINT_flmul_chkopt #1#2%
{%
    \ifx [#2\expandafter\XINT_flmul_opt
       \else\expandafter\XINT_flmul_noopt
    \fi  #1#2%
}%
\def\XINT_flmul_noopt #1#2\xint:#3%
{%
    #1[\XINTdigits]%
    {\expandafter\XINT_FL_mul_a
     \romannumeral0\XINTinfloatS[\XINTdigits]{#2}\XINTdigits.{#3}}%
}%
\def\XINT_flmul_opt #1[\xint:#2]%#3#4%
{%
    \expandafter\XINT_flmul_opt_a\the\numexpr #2.#1%
}%
\def\XINT_flmul_opt_a #1.#2#3#4%
{%
```

```
    #2[#1]{\expandafter\XINT_FL_mul_a\romannumeral0\XINTinfloatS[#1]{#3}#1.{#4}}%
}%
\def\XINT_FL_mul_a #1[#2]#3.#4%
{%
    \expandafter\XINT_FL_mul_b\romannumeral0\XINTinfloatS[#3]{#4}#1[#2]%
}%
\def\XINT_FL_mul_b #1[#2]#3[#4]{\xintiiMul{#3}{#1}/1[#4+#2]}%
```

25.85. \xintFloatSqr, \XINTinFloatSqr

Added at 1.4e (2021/05/05). Strangely \xintFloatSqr had never been defined so far.

An \XINTinFloatSqr{#1} was defined in xintexpr.sty directly as \XINTinFloatMul[\XINTdigits]{#1}{#1}, to support the sqr() function. The {#1}{#1} causes no problem as #1 in this context is always pre-expanded so we don't need to worry about this, and the \xintdefloatfunc mechanism should hopefully take care to add the needed argument pre-expansion if need be.

Anyway let's do this finally properly here.

```
\def\xintFloatSqr  {\romannumeral0\xintfloatsqr}%
\def\xintfloatsqr  #1{\XINT_flsqr_chkopt \xintfloat #1\xint:}%
\def\XINT_flsqr_chkopt #1#2%
{%
    \ifx [#2\expandafter\XINT_flsqr_opt
       \else\expandafter\XINT_flsqr_noopt
    \fi  #1#2%
}%
\def\XINT_flsqr_noopt #1#2\xint:
{%
    #1[\XINTdigits]%
    {\expandafter\XINT_FL_sqr_a\romannumeral0\XINTinfloatS[\XINTdigits]{#2}}%
}%
\def\XINT_flsqr_opt #1[\xint:#2]%
{%
    \expandafter\XINT_flsqr_opt_a\the\numexpr #2.#1%
}%
\def\XINT_flsqr_opt_a #1.#2#3%
{%
    #2[#1]{\expandafter\XINT_FL_sqr_a\romannumeral0\XINTinfloatS[#1]{#3}}%
}%
\def\XINT_FL_sqr_a #1[#2]{\xintiiSqr{#1}/1[#2+#2]}%
```

25.86. \xintFloatDiv, \XINTinFloatDiv

Added at 1.07 (2013/05/25).

Modified at 1.2f (2016/03/12). Starting with 1.2f the arguments are rounded to the target precision P not P+2.

Modified at 1.2g (2016/03/19). 1.2g handles the inputs via \XINTinFloatS which will be more efficient when the precision is large and the input is for example a small constant like 2.

The actual rounding of the quotient is handled via \xintfloat (or \XINTinfloatS).

Modified at 1.2k (2017/01/06). 1.2k does the same kind of improvement in \XINT_FL_div_b as for multiplication: earlier code was unnecessarily high level.

```
\def\xintFloatDiv  {\romannumeral0\xintfloatdiv}%
\def\xintfloatdiv  #1{\XINT_fldiv_chkopt \xintfloat #1\xint:}%
```

```
\def\XINT_fldiv_chkopt #1#2%
{%
    \ifx [#2\expandafter\XINT_fldiv_opt
       \else\expandafter\XINT_fldiv_noopt
    \fi  #1#2%
}%
```

1.4g adds here intercept of second argument being zero, else a low level error will arise at later stage from the the fall-back value returned by core iidivision being 0 and not having expected number of digits at \XINT_infloat_Qq and split from left returning some empty value breaking the \ifnum test in \XINT_infloat_Rq.

```
\def\XINT_fldiv_noopt #1#2\xint:#3%
{%
    #1[\XINTdigits]%
    {\expandafter\XINT_FL_div_aa
     \romannumeral0\XINTinfloatS[\XINTdigits]{#3}\XINTdigits.{#2}}%
}%
\def\XINT_FL_div_aa #1%
{%
    \xint_gob_til_zero#1\XINT_FL_div_Bzero0\XINT_FL_div_a #1%
}%
\def\XINT_FL_div_Bzero0\XINT_FL_div_a#1[#2]#3.#4%
{%
    \XINT_legacysignalcondition{DivisionByZero}{Division by zero (#1[#2]) of #4}{}{ 0[0]}%
}%
\def\XINT_fldiv_opt #1[\xint:#2]%#3#4%
{%
    \expandafter\XINT_fldiv_opt_a\the\numexpr #2.#1%
}%
```

Also here added early check at 1.4g if divisor is zero.

```
\def\XINT_fldiv_opt_a #1.#2#3#4%
{%
    #2[#1]{\expandafter\XINT_FL_div_aa\romannumeral0\XINTinfloatS[#1]{#4}#1.{#3}}%
}%
\def\XINT_FL_div_a #1[#2]#3.#4%
{%
    \expandafter\XINT_FL_div_b\romannumeral0\XINTinfloatS[#3]{#4}/#1e#2%
}%
\def\XINT_FL_div_b #1[#2]{#1e#2}%
```

25.87. \xintFloatPow, \XINTinFloatPow

Added at 1.07 (2013/05/25).
1.09j has re-organized the core loop.
2015/12/07. I have hesitated to map ^ in expressions to \xintFloatPow rather than \xintFloatPower. But for 1.234567890123456 to the power 2145678912 with P=16, using Pow rather than Power seems to bring only about 5% gain.
This routine requires the exponent x to be compatible with \numexpr parsing.

Modified at 1.2f (2016/03/12). 1.2f has rewritten the code for better efficiency. Also, now the argument A for A^x is first rounded to P digits before switching to the increased working precision (which depends upon x).

```
\def\xintFloatPow  {\romannumeral0\xintfloatpow}%
```

```
\def\xintfloatpow  #1{\XINT_flpow_chkopt \xintfloat #1\xint:}%
\def\XINTinfloatpow{\XINT_flpow_opt_a\XINTdigits.\XINTinfloatS}%
\def\XINTinfloatpow_wopt[#1]{\expandafter\XINT_flpow_opt_a\the\numexpr#1.\XINTinfloatS}%
\def\XINT_flpow_chkopt #1#2%
{%
    \ifx [#2\expandafter\XINT_flpow_opt
       \else\expandafter\XINT_flpow_noopt
    \fi
    #1#2%
}%
\def\XINT_flpow_noopt  #1#2\xint:#3%
{%
   \expandafter\XINT_flpow_checkB_a
   \the\numexpr #3.\XINTdigits.{#2}{#1[\XINTdigits]}%
}%
\def\XINT_flpow_opt #1[\xint:#2]%
{%
   \expandafter\XINT_flpow_opt_a\the\numexpr #2.#1%
}%
\def\XINT_flpow_opt_a #1.#2#3#4%
{%
   \expandafter\XINT_flpow_checkB_a\the\numexpr #4.#1.{#3}{#2[#1]}%
}%
\def\XINT_flpow_checkB_a #1%
{%
    \xint_UDzerominusfork
      #1-\XINT_flpow_BisZero
      0#1{\XINT_flpow_checkB_b -}%
       0-{\XINT_flpow_checkB_b {}#1}%
    \krof
}%
\def\XINT_flpow_BisZero .#1.#2#3{#3{1[0]}}%
\def\XINT_flpow_checkB_b #1#2.#3.%
{%
    \expandafter\XINT_flpow_checkB_c
    \the\numexpr\xintLength{#2}+\xint_c_iii.#3.#2.{#1}%
}%
\def\XINT_flpow_checkB_c #1.#2.%
{%
    \expandafter\XINT_flpow_checkB_d\the\numexpr#1+#2.#1.#2.%
}%
```

1.2f rounds input to P digits, first.

```
\def\XINT_flpow_checkB_d #1.#2.#3.#4.#5#6%
{%
    \expandafter \XINT_flpow_aa
    \romannumeral0\XINTinfloat [#3]{#6}{#2}{#1}{#4}{#5}%
}%
\def\XINT_flpow_aa #1[#2]#3%
{%
    \expandafter\XINT_flpow_ab\the\numexpr #2-#3\expandafter.%
    \romannumeral\XINT_rep #3\endcsname0.#1.%
}%
```

```
\def\XINT_flpow_ab #1.#2.#3.{\XINT_flpow_a #3#2[#1]}%
\def\XINT_flpow_a #1%
{%
    \xint_UDzerominusfork
      #1-\XINT_flpow_zero
      0#1{\XINT_flpow_b \iftrue}%
       0-{\XINT_flpow_b \iffalse#1}%
    \krof
}%
\def\XINT_flpow_zero #1[#2]#3#4#5#6%
{%
    #6{\if 1#51\xint_dothis {0[0]}\fi
       \xint_orthat
       {\XINT_legacysignalcondition{DivisionByZero}{0 raised to power -#4.}{}{ 0[0]}}%
      }%
}%
\def\XINT_flpow_b #1#2[#3]#4#5%
{%
    \XINT_flpow_loopI #5.#3.#2.#4.{#1\ifodd #5 \xint_c_i\fi\fi}%
}%
\def\XINT_flpow_truncate #1.#2.#3.%
{%
    \expandafter\XINT_flpow_truncate_a
    \romannumeral0\XINT_split_fromleft
    #3.#2\xint_bye2345678\xint_bye..#1.#3.%
}%
\def\XINT_flpow_truncate_a #1.#2.#3.{#3+\xintLength{#2}.#1.}%
\def\XINT_flpow_loopI #1.%
{%
    \ifnum #1=\xint_c_i\expandafter\XINT_flpow_ItoIII\fi
    \ifodd #1
       \expandafter\XINT_flpow_loopI_odd
    \else
       \expandafter\XINT_flpow_loopI_even
    \fi
    #1.%
}%
\def\XINT_flpow_ItoIII\ifodd #1\fi #2.#3.#4.#5.#6%
{%
    \expandafter\XINT_flpow_III\the\numexpr #6+\xint_c_.#3.#4.#5.%
}%
\def\XINT_flpow_loopI_even #1.#2.#3.%#4.%
{%
    \expandafter\XINT_flpow_loopI
    \the\numexpr #1/\xint_c_ii\expandafter.%
    \the\numexpr\expandafter\XINT_flpow_truncate
    \the\numexpr\xint_c_ii*#2\expandafter.\romannumeral0\xintiisqr{#3}.%
}%
\def\XINT_flpow_loopI_odd #1.#2.#3.#4.%
{%
    \expandafter\XINT_flpow_loopII
    \the\numexpr #1/\xint_c_ii-\xint_c_i\expandafter.%
```

```
    \the\numexpr\expandafter\XINT_flpow_truncate
    \the\numexpr\xint_c_ii*#2\expandafter.\romannumeral0\xintiisqr{#3}.#4.#2.#3.%
}%
\def\XINT_flpow_loopII #1.%
{%
    \ifnum #1 = \xint_c_i\expandafter\XINT_flpow_IItoIII\fi
    \ifodd #1
       \expandafter\XINT_flpow_loopII_odd
    \else
       \expandafter\XINT_flpow_loopII_even
    \fi
    #1.%
}%
\def\XINT_flpow_loopII_even #1.#2.#3.%#4.%
{%
    \expandafter\XINT_flpow_loopII
    \the\numexpr #1/\xint_c_ii\expandafter.%
    \the\numexpr\expandafter\XINT_flpow_truncate
    \the\numexpr\xint_c_ii*#2\expandafter.\romannumeral0\xintiisqr{#3}.%
}%
\def\XINT_flpow_loopII_odd #1.#2.#3.#4.#5.#6.%
{%
    \expandafter\XINT_flpow_loopII_odda
    \the\numexpr\expandafter\XINT_flpow_truncate
    \the\numexpr#2+#5\expandafter.\romannumeral0\xintiimul{#3}{#6}.#4.%
    #1.#2.#3.%
}%
\def\XINT_flpow_loopII_odda #1.#2.#3.#4.#5.#6.%
{%
    \expandafter\XINT_flpow_loopII
    \the\numexpr #4/\xint_c_ii-\xint_c_i\expandafter.%
    \the\numexpr\expandafter\XINT_flpow_truncate
    \the\numexpr\xint_c_ii*#5\expandafter.\romannumeral0\xintiisqr{#6}.#3.%
    #1.#2.%
}%
\def\XINT_flpow_IItoIII\ifodd #1\fi #2.#3.#4.#5.#6.#7.#8%
{%
    \expandafter\XINT_flpow_III\the\numexpr #8+\xint_c_\expandafter.%
    \the\numexpr\expandafter\XINT_flpow_truncate
    \the\numexpr#3+#6\expandafter.\romannumeral0\xintiimul{#4}{#7}.#5.%
}%
```

This ending is common with \xintFloatPower.

In the case of negative exponent we need to inverse the Q-digits mantissa. This requires no special attention now as 1.2k's \xintFloat does correct rounding of fractions hence it is easy to bound the total error. It can be checked that the algorithm after final rounding to the target precision computes a value Z whose distance to the exact theoretical will be less than 0.52 ulp(Z) (and worst cases can only be slightly worse than 0.51 ulp(Z)).

In the case of the half-integer exponent (only via the expression interface,) the computation (which proceeds via \XINTinFloatPowerH) ends with a square root. This square root extraction is done with 3 guard digits (the power operations were done with more.) Then the value is rounded to the target precision. There is thus this rounding to 3 guard digits (in the case of negative exponent the reciprocal is computed before the square-root), then the square root is (computed

with exact rounding for these 3 guard digits), and then there is the final rounding of this to the target precision. The total error (for positive as well as negative exponent) has been estimated to at worst possibly exceed slightly 0.5125 ulp(Z), and at any rate it is less than 0.52 ulp(Z).

```
\def\XINT_flpow_III #1.#2.#3.#4.#5%
{%
    \expandafter\XINT_flpow_IIIend
    \xint_UDsignfork
         #5{{1/#3[-#2]}}%
         -{{#3[#2]}}%
    \krof #1%
}%
\def\XINT_flpow_IIIend #1#2#3%
    {#3{\if#21\xint_afterfi{\expandafter-\romannumeral`&&@}\fi#1}}%
```

25.88. \xintFloatPower, \XINTinFloatPower

Added at 1.07 (2013/05/25). The core loop has been re-organized in 1.09j for some slight efficiency gain. The exponent B is given to \xintNum. The ^ in expressions is mapped to this routine.

Modified at 1.2f (2016/03/12). Same modifications as in \xintFloatPow for 1.2f.

1.2f \XINTinFloatPowerH (now moved to xintlog, and renamed). It truncated the exponent to an integer of half-integer, and in the latter case use Square-root extraction. At 1.2k this was improved as 1.2f stupidly rounded to Digits before, not after the square root extraction, 1.2k kept 3 guard digits for this last step. And the initial step was changed to a rounding rather than truncating.

Modified at 1.4e (2021/05/05). Until 1.4e this \XINTinFloatPowerH was the macro for a^b in expressions, but of course it behaved strangely for b not an integer or an half-integer! At 1.4e, the non-integer, non-half-integer exponents will be handled via log10() and pow10() support macros, see xintlog. The code has now been relocated there.

```
\def\xintFloatPower  {\romannumeral0\xintfloatpower}%
\def\xintfloatpower  #1{\XINT_flpower_chkopt \xintfloat #1\xint:}%
```

Start of macro. Check for optional argument.

```
\def\XINT_flpower_chkopt #1#2%
{%
    \ifx [#2\expandafter\XINT_flpower_opt
       \else\expandafter\XINT_flpower_noopt
    \fi
    #1#2%
}%
\def\XINT_flpower_noopt  #1#2\xint:#3%
{%
   \expandafter\XINT_flpower_checkB_a
   \romannumeral0\xintnum{#3}.\XINTdigits.{#2}{#1[\XINTdigits]}%
}%
\def\XINT_flpower_opt #1[\xint:#2]%
{%
   \expandafter\XINT_flpower_opt_a\the\numexpr #2.#1%
}%
\def\XINT_flpower_opt_a #1.#2#3#4%
{%
   \expandafter\XINT_flpower_checkB_a
```

```
    \romannumeral0\xintnum{#4}.#1.{#3}{#2[#1]}%
}%
\def\XINT_flpower_checkB_a #1%
{%
    \xint_UDzerominusfork
      #1-{\XINT_flpower_BisZero 0}%
      0#1{\XINT_flpower_checkB_b -}%
       0-{\XINT_flpower_checkB_b {}#1}%
    \krof
}%
\def\XINT_flpower_BisZero 0.#1.#2#3{#3{1[0]}}%
\def\XINT_flpower_checkB_b #1#2.#3.%
{%
    \expandafter\XINT_flpower_checkB_c
    \the\numexpr\xintLength{#2}+\xint_c_iii.#3.#2.{#1}%
}%
\def\XINT_flpower_checkB_c #1.#2.%
{%
    \expandafter\XINT_flpower_checkB_d\the\numexpr#1+#2.#1.#2.%
}%
\def\XINT_flpower_checkB_d #1.#2.#3.#4.#5#6%
{%
    \expandafter \XINT_flpower_aa
    \romannumeral0\XINTinfloat [#3]{#6}{#2}{#1}{#4}{#5}%
}%
\def\XINT_flpower_aa #1[#2]#3%
{%
    \expandafter\XINT_flpower_ab\the\numexpr #2-#3\expandafter.%
    \romannumeral\XINT_rep #3\endcsname0.#1.%
}%
\def\XINT_flpower_ab #1.#2.#3.{\XINT_flpower_a #3#2[#1]}%
\def\XINT_flpower_a #1%
{%
    \xint_UDzerominusfork
      #1-\XINT_flpow_zero
      0#1{\XINT_flpower_b \iftrue}%
       0-{\XINT_flpower_b \iffalse#1}%
    \krof
}%
\def\XINT_flpower_b #1#2[#3]#4#5%
{%
    \XINT_flpower_loopI #5.#3.#2.#4.{#1\xintiiOdd{#5}\fi}%
}%
\def\XINT_flpower_loopI #1.%
{%
    \if1\XINT_isOne {#1}\xint_dothis\XINT_flpower_ItoIII\fi
    \ifodd\xintLDg{#1} %<- intentional space
       \xint_dothis{\expandafter\XINT_flpower_loopI_odd}\fi
    \xint_orthat{\expandafter\XINT_flpower_loopI_even}%
    \romannumeral0\XINT_half
    #1\xint_bye\xint_Bye345678\xint_bye
    *\xint_c_v+\xint_c_v)/\xint_c_x-\xint_c_i\relax.%
```

```
}%
\def\XINT_flpower_ItoIII #1.#2.#3.#4.#5%
{%
    \expandafter\XINT_flpow_III\the\numexpr #5+\xint_c_.#2.#3.#4.%
}%
\def\XINT_flpower_loopI_even #1.#2.#3.#4.%
{%
    \expandafter\XINT_flpower_toloopI
    \the\numexpr\expandafter\XINT_flpow_truncate
    \the\numexpr\xint_c_ii*#2\expandafter.\romannumeral0\xintiisqr{#3}.#4.#1.%
}%
\def\XINT_flpower_toloopI #1.#2.#3.#4.{\XINT_flpower_loopI #4.#1.#2.#3.}%
\def\XINT_flpower_loopI_odd #1.#2.#3.#4.%
{%
    \expandafter\XINT_flpower_toloopII
    \the\numexpr\expandafter\XINT_flpow_truncate
    \the\numexpr\xint_c_ii*#2\expandafter.\romannumeral0\xintiisqr{#3}.#4.%
    #1.#2.#3.%
}%
\def\XINT_flpower_toloopII #1.#2.#3.#4.{\XINT_flpower_loopII #4.#1.#2.#3.}%
\def\XINT_flpower_loopII #1.%
{%
    \if1\XINT_isOne{#1}\xint_dothis\XINT_flpower_IItoIII\fi
    \ifodd\xintLDg{#1} %<- intentional space
        \xint_dothis{\expandafter\XINT_flpower_loopII_odd}\fi
    \xint_orthat{\expandafter\XINT_flpower_loopII_even}%
    \romannumeral0\XINT_half#1\xint_bye\xint_Bye345678\xint_bye
    *\xint_c_v+\xint_c_v)/\xint_c_x-\xint_c_i\relax.%
}%
\def\XINT_flpower_loopII_even #1.#2.#3.#4.%
{%
    \expandafter\XINT_flpower_toloopII
    \the\numexpr\expandafter\XINT_flpow_truncate
    \the\numexpr\xint_c_ii*#2\expandafter.\romannumeral0\xintiisqr{#3}.#4.#1.%
}%
\def\XINT_flpower_loopII_odd #1.#2.#3.#4.#5.#6.%
{%
    \expandafter\XINT_flpower_loopII_odda
    \the\numexpr\expandafter\XINT_flpow_truncate
    \the\numexpr#2+#5\expandafter.\romannumeral0\xintiimul{#3}{#6}.#4.%
    #1.#2.#3.%
}%
\def\XINT_flpower_loopII_odda #1.#2.#3.#4.#5.#6.%
{%
    \expandafter\XINT_flpower_toloopII
    \the\numexpr\expandafter\XINT_flpow_truncate
    \the\numexpr\xint_c_ii*#5\expandafter.\romannumeral0\xintiisqr{#6}.#3.%
    #4.#1.#2.%
}%
\def\XINT_flpower_IItoIII #1.#2.#3.#4.#5.#6.#7%
{%
    \expandafter\XINT_flpow_III\the\numexpr #7+\xint_c_\expandafter.%
```

```
    \the\numexpr\expandafter\XINT_flpow_truncate
    \the\numexpr#2+#5\expandafter.\romannumeral0\xintiimul{#3}{#6}.#4.%
}%
```

25.89. \xintFloatFac, \XINTFloatFac

Added at 1.2 (2015/10/10).

```
\def\xintFloatFac  {\romannumeral0\xintfloatfac}%
\def\xintfloatfac  #1{\XINT_flfac_chkopt \xintfloat #1\xint:}%
\def\XINTinFloatFac{\romannumeral0\XINTinfloatfac}%
\def\XINTinfloatfac[#1]{\expandafter\XINT_flfac_opt_a\the\numexpr#1.\XINTinfloatS}%
\def\XINT_flfac_chkopt #1#2%
{%
    \ifx [#2\expandafter\XINT_flfac_opt
       \else\expandafter\XINT_flfac_noopt
    \fi
     #1#2%
}%
\def\XINT_flfac_noopt  #1#2\xint:
{%
   \expandafter\XINT_FL_fac_fork_a
   \the\numexpr \xintNum{#2}.{\XINTdigits}{#1[\XINTdigits]}%
}%
\def\XINT_flfac_opt #1[\xint:#2]%
{%
   \expandafter\XINT_flfac_opt_a\the\numexpr #2.#1%
}%
\def\XINT_flfac_opt_a #1.#2#3%
{%
   \expandafter\XINT_FL_fac_fork_a\the\numexpr \xintNum{#3}.{#1}{#2[#1]}%
}%
\def\XINT_FL_fac_fork_a #1%
{%
     \xint_UDzerominusfork
     #1-\XINT_FL_fac_iszero
     0#1\XINT_FL_fac_isneg
      0-{\XINT_FL_fac_fork_b #1}%
    \krof
}%
\def\XINT_FL_fac_iszero #1.#2#3{#3{1[0]}}%
```

1.2f XINT_FL_fac_isneg returns 0, earlier versions used 1 here.
  calx: the factorial of an integer 0 < n < 10^8 is computed by calxfp's \CALX_ffac_fork_b (xintfrac's former engine, native output rounded to the target precision P), converted to raw by \CALX_ftoraw for the final macro (\xintfloat or \XINTinfloatS) at P.

```
\def\XINT_FL_fac_isneg  #1.#2#3%
{%
    #3{\XINT_legacysignalcondition{InvalidOperation}
                      {Factorial argument is negative: -#1.}{}{ 0[0]}}%
}%
\def\XINT_FL_fac_fork_b #1.%
{%
    \ifnum #1>\xint_c_e_viii_mone\xint_dothis\XINT_FL_fac_toobig\fi
```

```
    \xint_orthat\XINT_FL_fac_go #1.%
}%
\def\XINT_FL_fac_toobig #1.#2#3%
{%
    #3{\XINT_legacysignalcondition{InvalidOperation}
                      {Factorial argument is too large: #1>=10^8.}{}{ 0[0]}}%
}%
\def\XINT_FL_fac_go #1.#2#3%
{%
    #3{\romannumeral0\CALX_ftoraw{\romannumeral0\CALX_ffac_fork_b
       #1.\xint_c_i{#2+\xint_c_ii}\CALX_ffac_out{#2}}}%
}%
```

25.90. \xintFloatPFactorial, \XINTinFloatPFactorial

Added at 1.2f (2016/03/12) [on 2015/11/29]. Partial factorial pfactorial(a,b)=(a+1)...b, only for non-negative integers with a<=b<10^8.

Modified at 1.2h (2016/11/20). Now avoids raising \xintError:OutOfRangePFac if the condition 0<=a<=b<10^8 is violated. Same as for \xintiiPFactorial.

Modified at 1.4e (2021/05/05). 1.4e extends the precision in floating point context adding some overhead but well.

```
\def\xintFloatPFactorial  {\romannumeral0\xintfloatpfactorial}%
\def\xintfloatpfactorial  #1{\XINT_flpfac_chkopt \xintfloat #1\xint:}%
\def\XINT_flpfac_chkopt #1#2%
{%
    \ifx [#2\expandafter\XINT_flpfac_opt
       \else\expandafter\XINT_flpfac_noopt
    \fi
     #1#2%
}%
\def\XINT_flpfac_noopt  #1#2\xint:#3%
{%
   \expandafter\XINT_FL_pfac_fork
   \the\numexpr \xintNum{#2}\expandafter.%
   \the\numexpr \xintNum{#3}.\xint_c_i{\XINTdigits}{#1[\XINTdigits]}%
}%
\def\XINT_flpfac_opt #1[\xint:#2]%
{%
   \expandafter\XINT_flpfac_opt_a\the\numexpr #2.#1%
}%
\def\XINT_flpfac_opt_a #1.#2#3#4%
{%
   \expandafter\XINT_FL_pfac_fork
   \the\numexpr \xintNum{#3}\expandafter.%
   \the\numexpr \xintNum{#4}.\xint_c_i{#1}{#2[#1]}%
}%
\def\XINT_FL_pfac_fork #1#2.#3#4.%
{%
    \unless\ifnum #1#2<#3#4 \xint_dothis\XINT_FL_pfac_one\fi
    \if-#3\xint_dothis\XINT_FL_pfac_neg \fi
    \if-#1\xint_dothis\XINT_FL_pfac_zero\fi
    \ifnum #3#4>\xint_c_e_viii_mone\xint_dothis\XINT_FL_pfac_outofrange\fi
```

```
2631     \xint_orthat \XINT_FL_pfac_go #1#2.#3#4.%
2632 }%
2633 \def\XINT_FL_pfac_outofrange #1.#2.#3#4#5%
2634 {%
2635     #5{\XINT_legacysignalcondition{InvalidOperation}
2636                       {pFactorial with too large argument: #2 >= 10^8.}{}{ 0[0]}}%
2637 }%
2638 \def\XINT_FL_pfac_one  #1.#2.#3#4#5{#5{1[0]}}%
2639 \def\XINT_FL_pfac_zero #1.#2.#3#4#5{#5{0[0]}}%
2640 \def\XINT_FL_pfac_neg -#1.-#2.%
2641 {%
2642     \ifnum #1>\xint_c_e_viii\xint_dothis\XINT_FL_pfac_outofrange\fi
2643     \xint_orthat {%
2644     \ifodd\numexpr#2-#1\relax\xint_afterfi{\expandafter-\romannumeral`&&@}\fi
2645     \expandafter\XINT_FL_pfac_go}%
2646     \the\numexpr #2-\xint_c_i\expandafter.\the\numexpr#1-\xint_c_i.%
2647 }%
```

calx: the product is computed by calxfp's \CALX_fpfac_increaseP (xintfrac's former engine, native output rounded to the target precision P), converted to raw by \CALX_ftoraw for the final macro at P.

```
2648 \def\XINT_FL_pfac_go #1.#2.#3#4#5%
2649 {%
2650     #5{\romannumeral0\CALX_ftoraw{\romannumeral0\CALX_fpfac_increaseP
2651        #1.#2.#3{#4+\xint_c_ii}{#4}}}%
2652 }%
```

25.91. \xintFloatBinomial, \XINTinFloatBinomial

Added at 1.2f (2016/03/12) [on 2015/12/01]. We compute binomial(x,y) as pfac(x-y,x)/y!, where the numerator and denominator are computed with a relative error at most 4.10^{-P-2}, then rounded (once I have a float truncation, I will use truncation rather) to P+3 digits, and finally the quotient is correctly rounded to P digits. This will guarantee that the exact value X differs from the computed one Y by at most 0.6 ulp(Y).

Modified at 1.2h (2016/11/20). As for \xintiiBinomial, hard to understand why last year I coded this to raise an error if y<0 or y>x ! The question of the Gamma function is for another occasion, here x and y must be (small) integers.

1.4e: same remarks as for factorial and partial factorial about added overhead due to extra guard digits.

```
2653 \def\xintFloatBinomial  {\romannumeral0\xintfloatbinomial}%
2654 \def\xintfloatbinomial  #1{\XINT_flbinom_chkopt \xintfloat #1\xint:}%
2655 \def\XINT_flbinom_chkopt #1#2%
2656 {%
2657     \ifx [#2\expandafter\XINT_flbinom_opt
2658        \else\expandafter\XINT_flbinom_noopt
2659     \fi  #1#2%
2660 }%
2661 \def\XINT_flbinom_noopt #1#2\xint:#3%
2662 {%
2663     \expandafter\XINT_FL_binom_a
2664     \the\numexpr\xintNum{#2}\expandafter.%
2665     \the\numexpr\xintNum{#3}\expandafter.\the\numexpr\XINTdigits.#1%
2666 }%
```

```
\def\XINT_flbinom_opt #1[\xint:#2]#3#4%
{%
    \expandafter\XINT_FL_binom_a
    \the\numexpr\xintNum{#3}\expandafter.\the\numexpr\xintNum{#4}\expandafter.%
    \the\numexpr #2.#1%
}%
\def\XINT_FL_binom_a #1.#2.%
{%
    \expandafter\XINT_FL_binom_fork \the\numexpr #1-#2.#2.#1.%
}%
\def\XINT_FL_binom_fork #1#2.#3#4.#5#6.%
{%
    \if-#5\xint_dothis \XINT_FL_binom_neg\fi
    \if-#1\xint_dothis \XINT_FL_binom_zero\fi
    \if-#3\xint_dothis \XINT_FL_binom_zero\fi
    \if0#1\xint_dothis \XINT_FL_binom_one\fi
    \if0#3\xint_dothis \XINT_FL_binom_one\fi
    \ifnum #5#6>\xint_c_e_viii_mone \xint_dothis\XINT_FL_binom_toobig\fi
    \ifnum #1#2>#3#4  \xint_dothis\XINT_FL_binom_ab \fi
                      \xint_orthat\XINT_FL_binom_aa
    #1#2.#3#4.#5#6.%
}%
\def\XINT_FL_binom_neg #1.#2.#3.#4.#5%
{%
    #5[#4]{\XINT_legacysignalcondition{InvalidOperation}
                          {Binomial with negative argument: #3.}{}{ 0[0]}}%
}%
\def\XINT_FL_binom_toobig #1.#2.#3.#4.#5%
{%
    #5[#4]{\XINT_legacysignalcondition{InvalidOperation}
                          {Binomial with too large argument: #3 >= 10^8.}{}{ 0[0]}}%
}%
\def\XINT_FL_binom_one  #1.#2.#3.#4.#5{#5[#4]{1[0]}}%
\def\XINT_FL_binom_zero #1.#2.#3.#4.#5{#5[#4]{0[0]}}%
\def\XINT_FL_binom_aa #1.#2.#3.#4.#5%
   {#5[#4]{\romannumeral0\CALX_ftoraw
     {\romannumeral0\CALX_fbinom_aa #1.#2.#3.#4.}}}%
\def\XINT_FL_binom_ab #1.#2.#3.#4.#5%
   {#5[#4]{\romannumeral0\CALX_ftoraw
     {\romannumeral0\CALX_fbinom_aa #2.#1.#3.#4.}}}%
```

25.92. \xintFloatSqrt, \XINTinFloatSqrt

Added at 1.08 (2013/06/07).

Modified at 1.2f (2016/03/12).

The float version was developed at the same time as the integer one and even a bit earlier. As a result the integer variant had some sub-optimal parts. Anyway, for 1.2f I have rewritten the integer variant, and the float variant delegates all preparatory wrok for it until the last step. In particular the very low precisions are not penalized anymore from doing computations for at least 17 or 18 digits. Both the large and small precisions give quite shorter computation times.

Also, after examining more closely the achieved precision I decided to extend the float version in order for it to obtain the correct rounding (for inputs already of at most P digits with P the

precision) of the theoretical exact value.

Beyond about 500 digits of precision the efficiency decreases swiftly, as is the case generally speaking with xintcore/xint/xintfrac arithmetic macros.

Final note: with 1.2f the input is always first rounded to P significant places.

```
\def\xintFloatSqrt  {\romannumeral0\xintfloatsqrt}%
\def\xintfloatsqrt  #1{\XINT_flsqrt_chkopt \xintfloat #1\xint:}%
\def\XINT_flsqrt_chkopt #1#2%
{%
    \ifx [#2\expandafter\XINT_flsqrt_opt
       \else\expandafter\XINT_flsqrt_noopt
    \fi  #1#2%
}%
\def\XINT_flsqrt_noopt #1#2\xint:%
{%
    \expandafter\XINT_FL_sqrt_a
                \romannumeral0\XINTinfloat[\XINTdigits]{#2}\XINTdigits.#1%
}%
\def\XINT_flsqrt_opt #1[\xint:#2]%#3%
{%
    \expandafter\XINT_flsqrt_opt_a\the\numexpr #2.#1%
}%
\def\XINT_flsqrt_opt_a #1.#2#3%
{%
    \expandafter\XINT_FL_sqrt_a\romannumeral0\XINTinfloat[#1]{#3}#1.#2%
}%
\def\XINT_FL_sqrt_a #1%
{%
    \xint_UDzerominusfork
     #1-\XINT_FL_sqrt_iszero
     0#1\XINT_FL_sqrt_isneg
      0-{\XINT_FL_sqrt_pos #1}%
    \krof
}%[
\def\XINT_FL_sqrt_iszero #1]#2.#3{#3[#2]{0[0]}}%
\def\XINT_FL_sqrt_isneg #1]#2.#3%
{%
   #3[#2]{\XINT_legacysignalcondition{InvalidOperation}
                        {Square root of negative: -#1].}{}{ 0[0]}}%
}%
```

calx (s58): the square root of the positive A[N] (rounded to P digits by \XINTinfloat) is calxfp's \CALX_fsqrt (correctly rounded, as before) on its exact F value (\CALX_ffromraw), back to raw by \CALX_ftoraw for the output wrapper #4 (\xintfloat or \XINTinfloatS) at P. The former \XINT_flsqrt* chain, which used xint's big integer square root, is deleted.

```
\def\XINT_FL_sqrt_pos #1[#2]#3.#4%
   {#4[#3]{\romannumeral0\CALX_ftoraw
     {\romannumeral0\CALX_fsqrt{#3}{\romannumeral0\CALX_ffromraw{#1[#2]}}}}}%
```

25.93. \xintFloatE

Added at 1.07 (2013/05/25). The fraction is the first argument contrarily to \xintTrunc and \xintRound.

```
\def\xintFloatE   {\romannumeral0\xintfloate }%
```

```
\def\xintfloate #1{\XINT_floate_chkopt #1\xint:}%
\def\XINT_floate_chkopt #1%
{%
    \ifx [#1\expandafter\XINT_floate_opt
       \else\expandafter\XINT_floate_noopt
    \fi  #1%
}%
\def\XINT_floate_noopt #1\xint:%
{%
    \expandafter\XINT_floate_post
    \romannumeral0\XINTinfloat[\XINTdigits]{#1}\XINTdigits.%
}%
\def\XINT_floate_opt [\xint:#1]%
{%
    \expandafter\XINT_floate_opt_a\the\numexpr #1.%
}%
\def\XINT_floate_opt_a #1.#2%
{%
    \expandafter\XINT_floate_post
    \romannumeral0\XINTinfloat[#1]{#2}#1.%
}%
\def\XINT_floate_post #1%
{%
    \xint_UDzerominusfork
      #1-\XINT_floate_zero
       0#1\XINT_floate_neg
       0-\XINT_floate_pos
    \krof #1%
}%[
\def\XINT_floate_zero #1]#2.#3{ 0.e0}%
\def\XINT_floate_neg-{\expandafter-\romannumeral0\XINT_floate_pos}%
\def\XINT_floate_pos #1#2[#3]#4.#5%
{%
    \expandafter\XINT_float_pos_done\the\numexpr#3+#4+#5-\xint_c_i.#1.#2;%
}%
```

25.94. \xintifFloatInt

Added at 1.3a (2018/03/07). For ifint() function in \xintfloatexpr.

```
\def\xintifFloatInt {\romannumeral0\xintiffloatint}%
\def\xintiffloatint #1{\expandafter\XINT_iffloatint
                       \romannumeral0\xintrez{\XINTinFloatS[\XINTdigits]{#1}}}%
\def\XINT_iffloatint #1#2/1[#3]%
{%
  \if 0#1\xint_dothis\xint_stop_atfirstoftwo\fi
  \ifnum#3<\xint_c_\xint_dothis\xint_stop_atsecondoftwo\fi
  \xint_orthat\xint_stop_atfirstoftwo
}%
```

25.95. \xintFloatIsInt

Added at 1.3d (2019/01/06). For isint() function in \xintfloatexpr.

```
\def\xintFloatIsInt {\romannumeral0\xintfloatisint}%
\def\xintfloatisint #1{\expandafter\XINT_iffloatint
     \romannumeral0\xintrez{\XINTinFloatS[\XINTdigits]{#1}}10}%
```

25.96. \xintFloatIntType

Added at 1.4e (2021/05/05). For fractional powers. Expands to \xint_c_mone if argument is not an integer, to \xint_c_ if it is an even integer and to \xint_c_i if it is an odd integer.

```
\def\xintFloatIntType {\romannumeral`&&@\xintfloatinttype}%
\def\xintfloatinttype #1%
{%
    \expandafter\XINT_floatinttype
    \romannumeral0\xintrez{\XINTinFloatS[\XINTdigits]{#1}}%
}%
\def\XINT_floatinttype #1#2/1[#3]%
{%
    \if 0#1\xint_dothis\xint_c_\fi
    \ifnum#3<\xint_c_\xint_dothis\xint_c_mone\fi
    \ifnum#3>\xint_c_\xint_dothis\xint_c_\fi
    \ifodd\xintLDg{#1#2} \xint_dothis\xint_c_i\fi
    \xint_orthat\xint_c_
}%
\XINTrestorecatcodesendinput%
```

26. Package xintseries implementation

Contents

.1 Introduction 460
.2 Catcodes, ε-TEX and reload detection . . 460
.3 Package identification 461
.4 \xintSeries 461
.5 \xintiSeries 461
.6 \xintPowerSeries 462
.7 \xintPowerSeriesX 463
.8 \xintRationalSeries 463
.9 \xintRationalSeriesX 464
.10 \xintFxPtPowerSeries 465
.11 \xintFxPtPowerSeriesX 466
.12 \xintFloatPowerSeries 466
.13 \xintFloatPowerSeriesX 468

26.1. Introduction

The commenting is currently (2026/11/26) very sparse.

26.2. Catcodes, ε-TEX and reload detection

The code for reload detection was initially copied from Heiko Oberdiek's packages, then modified. The method for catcodes was also initially directly inspired by these packages.

```
\begingroup\catcode61\catcode48\catcode32=10\relax%
  \catcode13=5    % ^^M
  \endlinechar=13 %
  \catcode123=1   % {
  \catcode125=2   % }
  \catcode64=11   % @
  \catcode44=12   % ,
  \catcode46=12   % .
  \catcode58=12   % :
  \catcode94=7    % ^
  \def\empty{}\def\space{ }\newlinechar10
  \def\z{\endgroup}%
  \expandafter\let\expandafter\x\csname ver@xintseries.sty\endcsname
  \expandafter\let\expandafter\w\csname ver@xintfrac.sty\endcsname
  \expandafter\ifx\csname numexpr\endcsname\relax
    \expandafter\ifx\csname PackageWarningNoLine\endcsname\relax
      \immediate\write128{^^JPackage xintseries Warning:^^J%
                          \space\space\space\space
                          \numexpr not available, aborting input.^^J}%
    \else
      \PackageWarningNoLine{xintseries}{\numexpr not available, aborting input}%
    \fi
    \def\z{\endgroup\endinput}%
  \else
    \ifx\x\relax   % plain-TeX, first loading of xintseries.sty
      \ifx\w\relax % but xintfrac.sty not yet loaded.
         \def\z{\endgroup\input xintfrac.sty\relax}%
      \fi
    \else
      \ifx\x\empty % LaTeX, first loading,
      % variable is initialized, but \ProvidesPackage not yet seen
          \ifx\w\relax % xintfrac.sty not yet loaded.
            \def\z{\endgroup\RequirePackage{xintfrac}}%
```

```
          \fi
      \else
        \def\z{\endgroup\endinput}% xintseries already loaded.
      \fi
    \fi
  \fi
\z%
\XINTsetupcatcodes% defined in xintkernel.sty
```

26.3. Package identification

```
\XINT_providespackage
\ProvidesPackage{xintseries}%
  [2026/11/26 v2.0a2 Partial sums with xint package (JFB)]%
```

26.4. \xintSeries

```
\def\xintSeries {\romannumeral0\xintseries }%
\def\xintseries #1#2%
{%
    \expandafter\XINT_series\expandafter
    {\the\numexpr #1\expandafter}\expandafter{\the\numexpr #2}%
}%
\def\XINT_series #1#2#3%
{%
   \ifnum #2<#1
      \xint_afterfi { 0/1[0]}%
   \else
      \xint_afterfi {\XINT_series_loop {#1}{0}{#2}{#3}}%
   \fi
}%
\def\XINT_series_loop #1#2#3#4%
{%
    \ifnum #3>#1 \else \XINT_series_exit \fi
    \expandafter\XINT_series_loop\expandafter
    {\the\numexpr #1+1\expandafter }\expandafter
    {\romannumeral0\xintadd {#2}{#4{#1}}}%
    {#3}{#4}%
}%
\def\XINT_series_exit \fi #1#2#3#4#5#6#7#8%
{%
    \fi\xint_gobble_ii #6%
}%
```

26.5. \xintiSeries

```
\def\xintiSeries {\romannumeral0\xintiseries }%
\def\xintiseries #1#2%
{%
    \expandafter\XINT_iseries\expandafter
    {\the\numexpr #1\expandafter}\expandafter{\the\numexpr #2}%
}%
\def\XINT_iseries #1#2#3%
```

```
{%
   \ifnum #2<#1
      \xint_afterfi { 0}%
   \else
      \xint_afterfi {\XINT_iseries_loop {#1}{0}{#2}{#3}}%
   \fi
}%
\def\XINT_iseries_loop #1#2#3#4%
{%
    \ifnum #3>#1 \else \XINT_iseries_exit \fi
    \expandafter\XINT_iseries_loop\expandafter
    {\the\numexpr #1+1\expandafter }\expandafter
    {\romannumeral0\xintiiadd {#2}{#4{#1}}}%
    {#3}{#4}%
}%
\def\XINT_iseries_exit \fi #1#2#3#4#5#6#7#8%
{%
    \fi\xint_gobble_ii #6%
}%
```

26.6. \xintPowerSeries

The 1.03 version was very lame and created a build-up of denominators. (this was at a time \xintAdd always multiplied denominators, by the way) The Horner scheme for polynomial evaluation is used in 1.04, this cures the denominator problem and drastically improves the efficiency of the macro. Modified in 1.06 to give the indices first to a \numexpr rather than expanding twice. I just use \the\numexpr and maintain the previous code after that. 1.08a adds the forgotten optimization following that previous change.

```
\def\xintPowerSeries {\romannumeral0\xintpowerseries }%
\def\xintpowerseries #1#2%
{%
    \expandafter\XINT_powseries\expandafter
    {\the\numexpr #1\expandafter}\expandafter{\the\numexpr #2}%
}%
\def\XINT_powseries #1#2#3#4%
{%
   \ifnum #2<#1
      \xint_afterfi { 0/1[0]}%
   \else
      \xint_afterfi
      {\XINT_powseries_loop_i {#3{#2}}{#1}{#2}{#3}{#4}}%
   \fi
}%
\def\XINT_powseries_loop_i #1#2#3#4#5%
{%
    \ifnum #3>#2 \else\XINT_powseries_exit_i\fi
    \expandafter\XINT_powseries_loop_ii\expandafter
    {\the\numexpr #3-1\expandafter}\expandafter
    {\romannumeral0\xintmul {#1}{#5}}{#2}{#4}{#5}%
}%
\def\XINT_powseries_loop_ii #1#2#3#4%
{%
```

```
    \expandafter\XINT_powseries_loop_i\expandafter
    {\romannumeral0\xintadd {#4{#1}}{#2}}{#3}{#1}{#4}%
}%
\def\XINT_powseries_exit_i\fi #1#2#3#4#5#6#7#8#9%
{%
     \fi \XINT_powseries_exit_ii  #6{#7}%
}%
\def\XINT_powseries_exit_ii #1#2#3#4#5#6%
{%
    \xintmul{\xintPow {#5}{#6}}{#4}%
}%
```

26.7. \xintPowerSeriesX

Same as \xintPowerSeries except for the initial expansion of the x parameter. Modified in 1.06 to give the indices first to a \numexpr rather than expanding twice. I just use \the\numexpr and maintain the previous code after that. 1.08a adds the forgotten optimization following that previous change.

```
\def\xintPowerSeriesX {\romannumeral0\xintpowerseriesx }%
\def\xintpowerseriesx #1#2%
{%
    \expandafter\XINT_powseriesx\expandafter
    {\the\numexpr #1\expandafter}\expandafter{\the\numexpr #2}%
}%
\def\XINT_powseriesx #1#2#3#4%
{%
   \ifnum #2<#1
      \xint_afterfi { 0/1[0]}%
   \else
      \xint_afterfi
      {\expandafter\XINT_powseriesx_pre\expandafter
                  {\romannumeral`&&@#4}{#1}{#2}{#3}%
      }%
   \fi
}%
\def\XINT_powseriesx_pre #1#2#3#4%
{%
    \XINT_powseries_loop_i {#4{#3}}{#2}{#3}{#4}{#1}%
}%
```

26.8. \xintRationalSeries

This computes F(a)+...+F(b) on the basis of the value of F(a) and the ratios F(n)/F(n-1). As in \xintPowerSeries we use an iterative scheme which has the great advantage to avoid denominator build-up. This makes exact computations possible with exponential type series, which would be completely inaccessible to \xintSeries. #1=a, #2=b, #3=F(a), #4=ratio function Modified in 1.06 to give the indices first to a \numexpr rather than expanding twice. I just use \the\numexpr and maintain the previous code after that. 1.08a adds the forgotten optimization following that previous change.

```
\def\xintRationalSeries {\romannumeral0\xintratseries }%
\def\xintratseries #1#2%
{%
```

```
    \expandafter\XINT_ratseries\expandafter
    {\the\numexpr #1\expandafter}\expandafter{\the\numexpr #2}%
}%
\def\XINT_ratseries #1#2#3#4%
{%
   \ifnum #2<#1
      \xint_afterfi { 0/1[0]}%
   \else
      \xint_afterfi
      {\XINT_ratseries_loop {#2}{1}{#1}{#4}{#3}}%
   \fi
}%
\def\XINT_ratseries_loop #1#2#3#4%
{%
    \ifnum #1>#3 \else\XINT_ratseries_exit_i\fi
    \expandafter\XINT_ratseries_loop\expandafter
    {\the\numexpr #1-1\expandafter}\expandafter
    {\romannumeral0\xintadd {1}{\xintMul {#2}{#4{#1}}}}{#3}{#4}%
}%
\def\XINT_ratseries_exit_i\fi #1#2#3#4#5#6#7#8%
{%
    \fi \XINT_ratseries_exit_ii  #6%
}%
\def\XINT_ratseries_exit_ii #1#2#3#4#5%
{%
    \XINT_ratseries_exit_iii #5%
}%
\def\XINT_ratseries_exit_iii #1#2#3#4%
{%
    \xintmul{#2}{#4}%
}%
```

26.9. \xintRationalSeriesX

a,b,initial,ratiofunction,x
This computes F(a,x)+...+F(b,x) on the basis of the value of F(a,x) and the ratios F(n,x)/F(n-1,x). The argument x is first expanded and it is the value resulting from this which is used then throughout. The initial term F(a,x) must be defined as one-parameter macro which will be given x. Modified in 1.06 to give the indices first to a \numexpr rather than expanding twice. I just use \the\numexpr and maintain the previous code after that. 1.08a adds the forgotten optimization following that previous change.

```
\def\xintRationalSeriesX {\romannumeral0\xintratseriesx }%
\def\xintratseriesx #1#2%
{%
    \expandafter\XINT_ratseriesx\expandafter
    {\the\numexpr #1\expandafter}\expandafter{\the\numexpr #2}%
}%
\def\XINT_ratseriesx #1#2#3#4#5%
{%
   \ifnum #2<#1
      \xint_afterfi { 0/1[0]}%
   \else
```

```
      \xint_afterfi
      {\expandafter\XINT_ratseriesx_pre\expandafter
                  {\romannumeral`&&@#5}{#2}{#1}{#4}{#3}%
      }%
   \fi
}%
\def\XINT_ratseriesx_pre #1#2#3#4#5%
{%
    \XINT_ratseries_loop {#2}{1}{#3}{#4{#1}}{#5{#1}}%
}%
```

26.10. \xintFxPtPowerSeries

I am not two happy with this piece of code. Will make it more economical another day. Modified in 1.06 to give the indices first to a \numexpr rather than expanding twice. I just use \the\numexpr and maintain the previous code after that. 1.08a: forgot last time some optimization from the change to \numexpr.

```
\def\xintFxPtPowerSeries {\romannumeral0\xintfxptpowerseries }%
\def\xintfxptpowerseries #1#2%
{%
    \expandafter\XINT_fppowseries\expandafter
    {\the\numexpr #1\expandafter}\expandafter{\the\numexpr #2}%
}%
\def\XINT_fppowseries #1#2#3#4#5%
{%
   \ifnum #2<#1
      \xint_afterfi { 0}%
   \else
      \xint_afterfi
        {\expandafter\XINT_fppowseries_loop_pre\expandafter
           {\romannumeral0\xinttrunc {#5}{\xintPow {#4}{#1}}}%
          {#1}{#4}{#2}{#3}{#5}%
        }%
   \fi
}%
\def\XINT_fppowseries_loop_pre #1#2#3#4#5#6%
{%
    \ifnum #4>#2 \else\XINT_fppowseries_dont_i \fi
    \expandafter\XINT_fppowseries_loop_i\expandafter
    {\the\numexpr #2+\xint_c_i\expandafter}\expandafter
    {\romannumeral0\xintitrunc {#6}{\xintMul {#5{#2}}{#1}}}%
    {#1}{#3}{#4}{#5}{#6}%
}%
\def\XINT_fppowseries_dont_i \fi\expandafter\XINT_fppowseries_loop_i
    {\fi \expandafter\XINT_fppowseries_dont_ii }%
\def\XINT_fppowseries_dont_ii #1#2#3#4#5#6#7{\xinttrunc {#7}{#2[-#7]}}%
\def\XINT_fppowseries_loop_i #1#2#3#4#5#6#7%
{%
    \ifnum #5>#1 \else \XINT_fppowseries_exit_i \fi
    \expandafter\XINT_fppowseries_loop_ii\expandafter
    {\romannumeral0\xinttrunc {#7}{\xintMul {#3}{#4}}}%
    {#1}{#4}{#2}{#5}{#6}{#7}%
```

```
}%
\def\XINT_fppowseries_loop_ii #1#2#3#4#5#6#7%
{%
    \expandafter\XINT_fppowseries_loop_i\expandafter
    {\the\numexpr #2+\xint_c_i\expandafter}\expandafter
    {\romannumeral0\xintiiadd {#4}{\xintiTrunc {#7}{\xintMul {#6{#2}}{#1}}}}%
    {#1}{#3}{#5}{#6}{#7}%
}%
\def\XINT_fppowseries_exit_i\fi\expandafter\XINT_fppowseries_loop_ii
    {\fi \expandafter\XINT_fppowseries_exit_ii }%
\def\XINT_fppowseries_exit_ii #1#2#3#4#5#6#7%
{%
    \xinttrunc {#7}
    {\xintiiadd {#4}{\xintiTrunc {#7}{\xintMul {#6{#2}}{#1}}}[-#7]}%
}%
```

26.11. \xintFxPtPowerSeriesX

a,b,coeff,x,D
Modified in 1.06 to give the indices first to a \numexpr rather than expanding twice. I just use \the\numexpr and maintain the previous code after that. 1.08a adds the forgotten optimization following that previous change.

```
\def\xintFxPtPowerSeriesX {\romannumeral0\xintfxptpowerseriesx }%
\def\xintfxptpowerseriesx #1#2%
{%
    \expandafter\XINT_fppowseriesx\expandafter
    {\the\numexpr #1\expandafter}\expandafter{\the\numexpr #2}%
}%
\def\XINT_fppowseriesx #1#2#3#4#5%
{%
   \ifnum #2<#1
      \xint_afterfi { 0}%
   \else
      \xint_afterfi
        {\expandafter \XINT_fppowseriesx_pre \expandafter
         {\romannumeral`&&@#4}{#1}{#2}{#3}{#5}%
        }%
   \fi
}%
\def\XINT_fppowseriesx_pre #1#2#3#4#5%
{%
    \expandafter\XINT_fppowseries_loop_pre\expandafter
       {\romannumeral0\xinttrunc {#5}{\xintPow {#1}{#2}}}%
       {#2}{#1}{#3}{#4}{#5}%
}%
```

26.12. \xintFloatPowerSeries

1.08a. I still have to re-visit \xintFxPtPowerSeries; temporarily I just adapted the code to the case of floats.
Usage of new names \XINTinfloatpow_wopt \XINTinfloatmul_wopt, \XINTinfloatadd_wopt to track xintfrac.sty changes at 1.4e.

```
\def\xintFloatPowerSeries {\romannumeral0\xintfloatpowerseries }%
\def\xintfloatpowerseries #1{\XINT_flpowseries_chkopt #1\xint:}%
\def\XINT_flpowseries_chkopt #1%
{%
    \ifx [#1\expandafter\XINT_flpowseries_opt
       \else\expandafter\XINT_flpowseries_noopt
    \fi
    #1%
}%
\def\XINT_flpowseries_noopt  #1\xint:#2%
{%
    \expandafter\XINT_flpowseries\expandafter
    {\the\numexpr #1\expandafter}\expandafter
    {\the\numexpr #2}\XINTdigits
}%
\def\XINT_flpowseries_opt [\xint:#1]#2#3%
{%
    \expandafter\XINT_flpowseries\expandafter
    {\the\numexpr #2\expandafter}\expandafter
    {\the\numexpr #3\expandafter}{\the\numexpr #1}%
}%
\def\XINT_flpowseries #1#2#3#4#5%
{%
   \ifnum #2<#1
      \xint_afterfi { 0.e0}%
   \else
      \xint_afterfi
        {\expandafter\XINT_flpowseries_loop_pre\expandafter
           {\romannumeral0\XINTinfloatpow_wopt[#3]{#5}{#1}}%
          {#1}{#5}{#2}{#4}{#3}%
        }%
   \fi
}%
\def\XINT_flpowseries_loop_pre #1#2#3#4#5#6%
{%
    \ifnum #4>#2 \else\XINT_flpowseries_dont_i \fi
    \expandafter\XINT_flpowseries_loop_i\expandafter
    {\the\numexpr #2+\xint_c_i\expandafter}\expandafter
    {\romannumeral0\XINTinfloatmul_wopt[#6]{#5{#2}}{#1}}%
    {#1}{#3}{#4}{#5}{#6}%
}%
\def\XINT_flpowseries_dont_i \fi\expandafter\XINT_flpowseries_loop_i
    {\fi \expandafter\XINT_flpowseries_dont_ii }%
\def\XINT_flpowseries_dont_ii #1#2#3#4#5#6#7{\xintfloat [#7]{#2}}%
\def\XINT_flpowseries_loop_i #1#2#3#4#5#6#7%
{%
    \ifnum #5>#1 \else \XINT_flpowseries_exit_i \fi
    \expandafter\XINT_flpowseries_loop_ii\expandafter
    {\romannumeral0\XINTinfloatmul_wopt[#7]{#3}{#4}}%
    {#1}{#4}{#2}{#5}{#6}{#7}%
}%
\def\XINT_flpowseries_loop_ii #1#2#3#4#5#6#7%
```

```
{%
    \expandafter\XINT_flpowseries_loop_i\expandafter
    {\the\numexpr #2+\xint_c_i\expandafter}\expandafter
    {\romannumeral0\XINTinfloatadd_wopt[#7]{#4}%
                          {\XINTinfloatmul_wopt[#7]{#6{#2}}{#1}}}%
    {#1}{#3}{#5}{#6}{#7}%
}%
\def\XINT_flpowseries_exit_i\fi\expandafter\XINT_flpowseries_loop_ii
    {\fi \expandafter\XINT_flpowseries_exit_ii }%
\def\XINT_flpowseries_exit_ii #1#2#3#4#5#6#7%
{%
    \xintfloatadd[#7]{#4}{\XINTinfloatmul_wopt[#7]{#6{#2}}{#1}}%
}%
```

26.13. \xintFloatPowerSeriesX

1.08a

See \xintFloatPowerSeries for 1.4e comments.

```
\def\xintFloatPowerSeriesX {\romannumeral0\xintfloatpowerseriesx }%
\def\xintfloatpowerseriesx #1{\XINT_flpowseriesx_chkopt #1\xint:}%
\def\XINT_flpowseriesx_chkopt #1%
{%
    \ifx [#1\expandafter\XINT_flpowseriesx_opt
       \else\expandafter\XINT_flpowseriesx_noopt
    \fi
    #1%
}%
\def\XINT_flpowseriesx_noopt  #1\xint:#2%
{%
    \expandafter\XINT_flpowseriesx\expandafter
    {\the\numexpr #1\expandafter}\expandafter
    {\the\numexpr #2}\XINTdigits
}%
\def\XINT_flpowseriesx_opt [\xint:#1]#2#3%
{%
    \expandafter\XINT_flpowseriesx\expandafter
    {\the\numexpr #2\expandafter}\expandafter
    {\the\numexpr #3\expandafter}{\the\numexpr #1}%
}%
\def\XINT_flpowseriesx #1#2#3#4#5%
{%
   \ifnum #2<#1
      \xint_afterfi { 0.e0}%
   \else
      \xint_afterfi
        {\expandafter \XINT_flpowseriesx_pre \expandafter
         {\romannumeral`&&@#5}{#1}{#2}{#4}{#3}%
        }%
   \fi
}%
\def\XINT_flpowseriesx_pre #1#2#3#4#5%
{%
```

```
    \expandafter\XINT_flpowseries_loop_pre\expandafter
       {\romannumeral0\XINTinfloatpow_wopt[#5]{#1}{#2}}%
       {#2}{#1}{#3}{#4}{#5}%
}%
\XINTrestorecatcodesendinput%
```

27. Package xintcfrac implementation

Contents

.1 Introduction 470
.2 Catcodes, ε-TEX and reload detection . . 470
.3 Package identification 471
.4 \xintCFrac 471
.5 \xintGCFrac 473
.6 \xintGGCFrac 474
.7 \xintGCtoGCx 475
.8 \xintFtoCs 475
.9 \xintFtoCx 476
.10 \xintFtoC 477
.11 \xintFtoGC 477
.12 \xintFGtoC 477
.13 \xintFtoCC 478
.14 \xintCtoF, \xintCstoF 479
.15 \xintiCstoF 480
.16 \xintGCtoF 481
.17 \xintiGCtoF 482
.18 \xintCtoCv, \xintCstoCv 483
.19 \xintiCstoCv 484
.20 \xintGCtoCv 485
.21 \xintiGCtoCv 486
.22 \xintFtoCv 487
.23 \xintFtoCCv 487
.24 \xintCntoF 488
.25 \xintGCntoF 488
.26 \xintCntoCs 489
.27 \xintCntoGC 490
.28 \xintGCntoGC 491
.29 \xintCstoGC 491
.30 \xintGCtoGC 492

27.1. Introduction

The commenting is currently (2026/11/26) very sparse. Release 1.09m (2014/02/26) has modified a few things: \xintFtoCs and \xintCntoCs insert spaces after the commas, \xintCstoF and \xintCstoCv authorize spaces in the input also before the commas, \xintCntoCs does not brace the produced coefficients, new macros \xintFtoC, \xintCtoF, \xintCtoCv, \xintFGtoC, and \xintGGCFrac.

The macros \xintCstoF and \xintCstoCv use xinttools's \xintCSVtoList. Formerly, it was up to user to load xinttools to enable these macros. Starting at 1.4n the loading is automatic.

27.2. Catcodes, ε-TEX and reload detection

The code for reload detection was initially copied from Heiko Oberdiek's packages, then modified.

The method for catcodes was also initially directly inspired by these packages.

```
\begingroup\catcode61\catcode48\catcode32=10\relax%
  \catcode13=5    % ^^M
  \endlinechar=13 %
  \catcode123=1   % {
  \catcode125=2   % }
  \catcode64=11   % @
  \catcode44=12   % ,
  \catcode46=12   % .
  \catcode58=12   % :
  \catcode94=7    % ^
  \def\empty{}\def\space{ }\newlinechar10
  \def\z{\endgroup}%
  \expandafter\let\expandafter\x\csname ver@xintcfrac.sty\endcsname
  \expandafter\let\expandafter\w\csname ver@xintfrac.sty\endcsname
  \expandafter\let\expandafter\t\csname ver@xinttools.sty\endcsname
  \expandafter\ifx\csname numexpr\endcsname\relax
    \expandafter\ifx\csname PackageWarningNoLine\endcsname\relax
      \immediate\write128{^^JPackage xintcfrac Warning:^^J%
                          \space\space\space\space
                          \numexpr not available, aborting input.^^J}%
```

```
    \else
      \PackageWarningNoLine{xintcfrac}{\numexpr not available, aborting input}%
    \fi
    \def\z{\endgroup\endinput}%
  \else
    \ifx\x\relax   % not LaTeX, first loading of xintcfrac.sty
      \ifx\w\relax % but xintfrac.sty not yet loaded.
         \expandafter\def\expandafter\z\expandafter
                     {\z\input xintfrac.sty\relax}%
      \fi
      \ifx\t\relax % but xinttools.sty not yet loaded.
         \expandafter\def\expandafter\z\expandafter
                     {\z\input xinttools.sty\relax}%
      \fi
    \else
      \ifx\x\empty % LaTeX, first loading,
      % variable is initialized, but \ProvidesPackage not yet seen
          \ifx\w\relax % xintfrac not yet loaded.
            \expandafter\def\expandafter\z\expandafter
                        {\z\RequirePackage{xintfrac}}%
          \fi
          \ifx\t\relax % xinttools not yet loaded.
            \expandafter\def\expandafter\z\expandafter
                        {\z\RequirePackage{xinttools}}%
          \fi
      \else
        \def\z{\endgroup\endinput}% xintcfrac already loaded.
      \fi
    \fi
  \fi
\z%
\XINTsetupcatcodes% defined in xintkernel.sty
```

27.3. Package identification

```
\XINT_providespackage
\ProvidesPackage{xintcfrac}%
  [2026/11/26 v2.0a2 Continued fractions with xint package (JFB)]%
```

27.4. \xintCFrac

```
\def\xintCFrac {\romannumeral0\xintcfrac }%
\def\xintcfrac #1%
{%
    \XINT_cfrac_opt_a #1\xint:
}%
\def\XINT_cfrac_opt_a #1%
{%
    \ifx[#1\XINT_cfrac_opt_b\fi \XINT_cfrac_noopt #1%
}%
\def\XINT_cfrac_noopt #1\xint:
{%
    \expandafter\XINT_cfrac_A\romannumeral0\xintrawwithzeros {#1}\Z
```

```
    \relax\relax
}%
\def\XINT_cfrac_opt_b\fi\XINT_cfrac_noopt [\xint:#1]%
{%
    \fi\csname XINT_cfrac_opt#1\endcsname
}%
\def\XINT_cfrac_optl #1%
{%
    \expandafter\XINT_cfrac_A\romannumeral0\xintrawwithzeros {#1}\Z
    \relax\hfill
}%
\def\XINT_cfrac_optc #1%
{%
    \expandafter\XINT_cfrac_A\romannumeral0\xintrawwithzeros {#1}\Z
    \relax\relax
}%
\def\XINT_cfrac_optr #1%
{%
    \expandafter\XINT_cfrac_A\romannumeral0\xintrawwithzeros {#1}\Z
    \hfill\relax
}%
\def\XINT_cfrac_A #1/#2\Z
{%
    \expandafter\XINT_cfrac_B\romannumeral0\xintiidivision {#1}{#2}{#2}%
}%
\def\XINT_cfrac_B #1#2%
{%
    \XINT_cfrac_C #2\Z {#1}%
}%
\def\XINT_cfrac_C #1%
{%
    \xint_gob_til_zero #1\XINT_cfrac_integer 0\XINT_cfrac_D #1%
}%
\def\XINT_cfrac_integer 0\XINT_cfrac_D 0#1\Z #2#3#4#5{ #2}%
\def\XINT_cfrac_D #1\Z #2#3{\XINT_cfrac_loop_a {#1}{#3}{#1}{{#2}}}%
\def\XINT_cfrac_loop_a
{%
    \expandafter\XINT_cfrac_loop_d\romannumeral0\XINT_ndivrawpair
}%
\def\XINT_cfrac_loop_d #1#2%
{%
    \XINT_cfrac_loop_e #2.{#1}%
}%
\def\XINT_cfrac_loop_e #1%
{%
    \xint_gob_til_zero #1\xint_cfrac_loop_exit0\XINT_cfrac_loop_f #1%
}%
\def\XINT_cfrac_loop_f #1.#2#3#4%
{%
    \XINT_cfrac_loop_a {#1}{#3}{#1}{{#2}#4}%
}%
\def\xint_cfrac_loop_exit0\XINT_cfrac_loop_f #1.#2#3#4#5#6%
```

```
    {\XINT_cfrac_T #5#6{#2}#4\Z }%
\def\XINT_cfrac_T #1#2#3#4%
{%
  \xint_gob_til_Z #4\XINT_cfrac_end\Z\XINT_cfrac_T #1#2{#4+\cfrac{#11#2}{#3}}%
}%
\def\XINT_cfrac_end\Z\XINT_cfrac_T #1#2#3%
{%
    \XINT_cfrac_end_b #3%
}%
\def\XINT_cfrac_end_b \Z+\cfrac#1#2{ #2}%
```

27.5. \xintGCFrac

Updated at 1.4g to follow-up on renaming of \xintFrac into \xintTeXFrac.

```
\def\xintGCFrac {\romannumeral0\xintgcfrac }%
\def\xintgcfrac #1{\XINT_gcfrac_opt_a #1\xint:}%
\def\XINT_gcfrac_opt_a #1%
{%
    \ifx[#1\XINT_gcfrac_opt_b\fi \XINT_gcfrac_noopt #1%
}%
\def\XINT_gcfrac_noopt #1\xint:%
{%
    \XINT_gcfrac #1+!/\relax\relax
}%
\def\XINT_gcfrac_opt_b\fi\XINT_gcfrac_noopt [\xint:#1]%
{%
    \fi\csname XINT_gcfrac_opt#1\endcsname
}%
\def\XINT_gcfrac_optl #1%
{%
    \XINT_gcfrac #1+!/\relax\hfill
}%
\def\XINT_gcfrac_optc #1%
{%
    \XINT_gcfrac #1+!/\relax\relax
}%
\def\XINT_gcfrac_optr #1%
{%
    \XINT_gcfrac #1+!/\hfill\relax
}%
\def\XINT_gcfrac
{%
    \expandafter\XINT_gcfrac_enter\romannumeral`&&@%
}%
\def\XINT_gcfrac_enter {\XINT_gcfrac_loop {}}%
\def\XINT_gcfrac_loop #1#2+#3/%
{%
    \xint_gob_til_exclam #3\XINT_gcfrac_endloop!%
    \XINT_gcfrac_loop {{#3}{#2}#1}%
}%
\def\XINT_gcfrac_endloop!\XINT_gcfrac_loop #1#2#3%
{%
```

```
    \XINT_gcfrac_T #2#3#1!!%
}%
\def\XINT_gcfrac_T #1#2#3#4{\XINT_gcfrac_U #1#2{\xintTeXFrac{#4}}}%
\def\XINT_gcfrac_U #1#2#3#4#5%
{%
    \xint_gob_til_exclam #5\XINT_gcfrac_end!\XINT_gcfrac_U
               #1#2{\xintTeXFrac{#5}%
                \ifcase\xintSgn{#4}
                +\or+\else-\fi
                \cfrac{#1\xintTeXFrac{\xintAbs{#4}}#2}{#3}}%
}%
\def\XINT_gcfrac_end!\XINT_gcfrac_U #1#2#3%
{%
    \XINT_gcfrac_end_b #3%
}%
\def\XINT_gcfrac_end_b #1\cfrac#2#3{ #3}%
```

27.6. \xintGGCFrac

New with 1.09m

```
\def\xintGGCFrac {\romannumeral0\xintggcfrac }%
\def\xintggcfrac #1{\XINT_ggcfrac_opt_a #1\xint:}%
\def\XINT_ggcfrac_opt_a #1%
{%
    \ifx[#1\XINT_ggcfrac_opt_b\fi \XINT_ggcfrac_noopt #1%
}%
\def\XINT_ggcfrac_noopt #1\xint:
{%
    \XINT_ggcfrac #1+!/\relax\relax
}%
\def\XINT_ggcfrac_opt_b\fi\XINT_ggcfrac_noopt [\xint:#1]%
{%
    \fi\csname XINT_ggcfrac_opt#1\endcsname
}%
\def\XINT_ggcfrac_optl #1%
{%
    \XINT_ggcfrac #1+!/\relax\hfill
}%
\def\XINT_ggcfrac_optc #1%
{%
    \XINT_ggcfrac #1+!/\relax\relax
}%
\def\XINT_ggcfrac_optr #1%
{%
    \XINT_ggcfrac #1+!/\hfill\relax
}%
\def\XINT_ggcfrac
{%
    \expandafter\XINT_ggcfrac_enter\romannumeral`&&@%
}%
\def\XINT_ggcfrac_enter {\XINT_ggcfrac_loop {}}%
\def\XINT_ggcfrac_loop #1#2+#3/%
```

```
{%
    \xint_gob_til_exclam #3\XINT_ggcfrac_endloop!%
    \XINT_ggcfrac_loop {{#3}{#2}#1}%
}%
\def\XINT_ggcfrac_endloop!\XINT_ggcfrac_loop #1#2#3%
{%
    \XINT_ggcfrac_T #2#3#1!!%
}%
\def\XINT_ggcfrac_T #1#2#3#4{\XINT_ggcfrac_U #1#2{#4}}%
\def\XINT_ggcfrac_U #1#2#3#4#5%
{%
    \xint_gob_til_exclam #5\XINT_ggcfrac_end!\XINT_ggcfrac_U
              #1#2{#5+\cfrac{#1#4#2}{#3}}%
}%
\def\XINT_ggcfrac_end!\XINT_ggcfrac_U #1#2#3%
{%
    \XINT_ggcfrac_end_b #3%
}%
\def\XINT_ggcfrac_end_b #1\cfrac#2#3{ #3}%
```

27.7. \xintGCtoGCx

```
\def\xintGCtoGCx {\romannumeral0\xintgctogcx }%
\def\xintgctogcx #1#2#3%
{%
    \expandafter\XINT_gctgcx_start\expandafter {\romannumeral`&&@#3}{#1}{#2}%
}%
\def\XINT_gctgcx_start #1#2#3{\XINT_gctgcx_loop_a {}{#2}{#3}#1+!/}%
\def\XINT_gctgcx_loop_a #1#2#3#4+#5/%
{%
    \xint_gob_til_exclam #5\XINT_gctgcx_end!%
    \XINT_gctgcx_loop_b {#1{#4}}{#2{#5}#3}{#2}{#3}%
}%
\def\XINT_gctgcx_loop_b #1#2%
{%
    \XINT_gctgcx_loop_a {#1#2}%
}%
\def\XINT_gctgcx_end!\XINT_gctgcx_loop_b #1#2#3#4{ #1}%
```

27.8. \xintFtoCs

Modified in 1.09m: a space is added after the inserted commas.

```
\def\xintFtoCs {\romannumeral0\xintftocs }%
\def\xintftocs #1%
{%
    \expandafter\XINT_ftc_A\romannumeral0\xintrawwithzeros {#1}\Z
}%
\def\XINT_ftc_A #1/#2\Z
{%
    \expandafter\XINT_ftc_B\romannumeral0\xintiidivision {#1}{#2}{#2}%
}%
\def\XINT_ftc_B #1#2%
{%
```

```
    \XINT_ftc_C #2.{#1}%
}%
\def\XINT_ftc_C #1%
{%
    \xint_gob_til_zero #1\XINT_ftc_integer 0\XINT_ftc_D #1%
}%
\def\XINT_ftc_integer 0\XINT_ftc_D 0#1.#2#3{ #2}%
\def\XINT_ftc_D #1.#2#3{\XINT_ftc_loop_a {#1}{#3}{#1}{#2, }}% 1.09m adds a space
\def\XINT_ftc_loop_a
{%
    \expandafter\XINT_ftc_loop_d\romannumeral0\XINT_ndivrawpair
}%
\def\XINT_ftc_loop_d #1#2%
{%
    \XINT_ftc_loop_e #2.{#1}%
}%
\def\XINT_ftc_loop_e #1%
{%
    \xint_gob_til_zero #1\xint_ftc_loop_exit0\XINT_ftc_loop_f #1%
}%
\def\XINT_ftc_loop_f #1.#2#3#4%
{%
    \XINT_ftc_loop_a {#1}{#3}{#1}{#4#2, }% 1.09m has an added space here
}%
\def\xint_ftc_loop_exit0\XINT_ftc_loop_f #1.#2#3#4{ #4#2}%
```

27.9. \xintFtoCx

```
\def\xintFtoCx {\romannumeral0\xintftocx }%
\def\xintftocx #1#2%
{%
    \expandafter\XINT_ftcx_A\romannumeral0\xintrawwithzeros {#2}\Z {#1}%
}%
\def\XINT_ftcx_A #1/#2\Z
{%
    \expandafter\XINT_ftcx_B\romannumeral0\xintiidivision {#1}{#2}{#2}%
}%
\def\XINT_ftcx_B #1#2%
{%
    \XINT_ftcx_C #2.{#1}%
}%
\def\XINT_ftcx_C #1%
{%
    \xint_gob_til_zero #1\XINT_ftcx_integer 0\XINT_ftcx_D #1%
}%
\def\XINT_ftcx_integer 0\XINT_ftcx_D 0#1.#2#3#4{ #2}%
\def\XINT_ftcx_D #1.#2#3#4{\XINT_ftcx_loop_a {#1}{#3}{#1}{{#2}#4}{#4}}%
\def\XINT_ftcx_loop_a
{%
    \expandafter\XINT_ftcx_loop_d\romannumeral0\XINT_ndivrawpair
}%
\def\XINT_ftcx_loop_d #1#2%
{%
```

```
    \XINT_ftcx_loop_e #2.{#1}%
}%
\def\XINT_ftcx_loop_e #1%
{%
    \xint_gob_til_zero #1\xint_ftcx_loop_exit0\XINT_ftcx_loop_f #1%
}%
\def\XINT_ftcx_loop_f #1.#2#3#4#5%
{%
    \XINT_ftcx_loop_a {#1}{#3}{#1}{#4{#2}#5}{#5}%
}%
\def\xint_ftcx_loop_exit0\XINT_ftcx_loop_f #1.#2#3#4#5{ #4{#2}}%
```

27.10. \xintFtoC

New in 1.09m: this is the same as \xintFtoCx with empty separator. I had temporarily during preparation of 1.09m removed braces from \xintFtoCx, but I recalled later why that was useful (see doc), thus let's just here do \xintFtoCx {}

```
\def\xintFtoC {\romannumeral0\xintftoc }%
\def\xintftoc {\xintftocx {}}%
```

27.11. \xintFtoGC

```
\def\xintFtoGC {\romannumeral0\xintftogc }%
\def\xintftogc {\xintftocx {+1/}}%
```

27.12. \xintFGtoC

New with 1.09m of 2014/02/26. Computes the common initial coefficients for the two fractions f and g, and outputs them as a sequence of braced items.

```
\def\xintFGtoC {\romannumeral0\xintfgtoc}%
\def\xintfgtoc#1%
{%
    \expandafter\XINT_fgtc_a\romannumeral0\xintrawwithzeros {#1}\Z
}%
\def\XINT_fgtc_a #1/#2\Z #3%
{%
    \expandafter\XINT_fgtc_b\romannumeral0\xintrawwithzeros {#3}\Z #1/#2\Z { }%
}%
\def\XINT_fgtc_b #1/#2\Z
{%
    \expandafter\XINT_fgtc_c\romannumeral0\xintiidivision {#1}{#2}{#2}%
}%
\def\XINT_fgtc_c #1#2#3#4/#5\Z
{%
    \expandafter\XINT_fgtc_d\romannumeral0\xintiidivision
                                          {#4}{#5}{#5}{#1}{#2}{#3}%
}%
\def\XINT_fgtc_d #1#2#3#4%#5#6#7%
{%
    \xintifEq {#1}{#4}{\XINT_fgtc_da {#1}{#2}{#3}{#4}}%
                      {\xint_thirdofthree}%
}%
```

```
\def\XINT_fgtc_da #1#2#3#4#5#6#7%
{%
     \XINT_fgtc_e {#2}{#5}{#3}{#6}{#7{#1}}%
}%
\def\XINT_fgtc_e #1%
{%
    \xintiiifZero {#1}{\expandafter\xint_firstofone\xint_gobble_iii}%
                     {\XINT_fgtc_f {#1}}%
}%
\def\XINT_fgtc_f #1#2%
{%
   \xintiiifZero {#2}{\xint_thirdofthree}{\XINT_fgtc_g {#1}{#2}}%
}%
\def\XINT_fgtc_g #1#2#3%
{%
    \expandafter\XINT_fgtc_h\romannumeral0\XINT_ndivrawpair {#1}{#3}{#1}{#2}%
}%
\def\XINT_fgtc_h #1#2#3#4#5%
{%
    \expandafter\XINT_fgtc_d\romannumeral0\XINT_ndivrawpair
                        {#4}{#5}{#4}{#1}{#2}{#3}%
}%
```

27.13. \xintFtoCC

```
\def\xintFtoCC {\romannumeral0\xintftocc }%
\def\xintftocc #1%
{%
    \expandafter\XINT_ftcc_A\expandafter {\romannumeral0\xintrawwithzeros {#1}}%
}%
\def\XINT_ftcc_A #1%
{%
    \expandafter\XINT_ftcc_B
    \romannumeral0\xintrawwithzeros {\xintAdd {1/2[0]}{#1[0]}}\Z {#1[0]}%
}%
\def\XINT_ftcc_B #1/#2\Z
{%
    \expandafter\XINT_ftcc_C\expandafter {\romannumeral0\xintiiquo {#1}{#2}}%
}%
\def\XINT_ftcc_C #1#2%
{%
    \expandafter\XINT_ftcc_D\romannumeral0\xintsub {#2}{#1}\Z {#1}%
}%
\def\XINT_ftcc_D #1%
{%
    \xint_UDzerominusfork
      #1-\XINT_ftcc_integer
      0#1\XINT_ftcc_En
       0-{\XINT_ftcc_Ep #1}%
    \krof
}%
\def\XINT_ftcc_Ep #1\Z #2%
{%
```

```
    \expandafter\XINT_ftcc_loop_a\expandafter
    {\romannumeral0\xintdiv {1[0]}{#1}}{#2+1/}%
}%
\def\XINT_ftcc_En #1\Z #2%
{%
    \expandafter\XINT_ftcc_loop_a\expandafter
    {\romannumeral0\xintdiv {1[0]}{#1}}{#2+-1/}%
}%
\def\XINT_ftcc_integer #1\Z #2{ #2}%
\def\XINT_ftcc_loop_a #1%
{%
    \expandafter\XINT_ftcc_loop_b
    \romannumeral0\xintrawwithzeros {\xintAdd {1/2[0]}{#1}}\Z {#1}%
}%
\def\XINT_ftcc_loop_b #1/#2\Z
{%
    \expandafter\XINT_ftcc_loop_c\expandafter
    {\romannumeral0\xintiiquo {#1}{#2}}%
}%
\def\XINT_ftcc_loop_c #1#2%
{%
    \expandafter\XINT_ftcc_loop_d
    \romannumeral0\xintsub {#2}{#1[0]}\Z {#1}%
}%
\def\XINT_ftcc_loop_d #1%
{%
    \xint_UDzerominusfork
      #1-\XINT_ftcc_end
      0#1\XINT_ftcc_loop_N
       0-{\XINT_ftcc_loop_P #1}%
    \krof
}%
\def\XINT_ftcc_end #1\Z #2#3{ #3#2}%
\def\XINT_ftcc_loop_P #1\Z #2#3%
{%
    \expandafter\XINT_ftcc_loop_a\expandafter
    {\romannumeral0\xintdiv {1[0]}{#1}}{#3#2+1/}%
}%
\def\XINT_ftcc_loop_N #1\Z #2#3%
{%
    \expandafter\XINT_ftcc_loop_a\expandafter
    {\romannumeral0\xintdiv {1[0]}{#1}}{#3#2+-1/}%
}%
```

27.14. \xintCtoF, \xintCstoF

1.09m uses xinttools's \xintCSVtoList on the argument of \xintCstoF to allow spaces also before the commas. And the original \xintCstoF code became the one of the new \xintCtoF dealing with a braced rather than comma separated list.

```
\def\xintCstoF {\romannumeral0\xintcstof }%
\def\xintcstof #1%
{%
```

```
    \expandafter\XINT_ctf_prep \romannumeral0\xintcsvtolist{#1}!%
}%
\def\xintCtoF {\romannumeral0\xintctof }%
\def\xintctof #1%
{%
    \expandafter\XINT_ctf_prep \romannumeral`&&@#1!%
}%
\def\XINT_ctf_prep
{%
    \XINT_ctf_loop_a 1001%
}%
\def\XINT_ctf_loop_a #1#2#3#4#5%
{%
    \xint_gob_til_exclam #5\XINT_ctf_end!%
    \expandafter\XINT_ctf_loop_b
    \romannumeral0\xintrawwithzeros {#5}.{#1}{#2}{#3}{#4}%
}%
\def\XINT_ctf_loop_b #1/#2.#3#4#5#6%
{%
    \expandafter\XINT_ctf_loop_c\expandafter
    {\romannumeral0\xintiimul{#2}{#4}}%
    {\romannumeral0\xintiimul{#2}{#3}}%
    {\romannumeral0\xintiiadd {\xintiimul{#2}{#6}}%
                              {\xintiimul{#1}{#4}}}%
    {\romannumeral0\xintiiadd {\xintiimul{#2}{#5}}%
                              {\xintiimul{#1}{#3}}}%
}%
\def\XINT_ctf_loop_c #1#2%
{%
    \expandafter\XINT_ctf_loop_d\expandafter {\expandafter{#2}{#1}}%
}%
\def\XINT_ctf_loop_d #1#2%
{%
    \expandafter\XINT_ctf_loop_e\expandafter {\expandafter{#2}#1}%
}%
\def\XINT_ctf_loop_e #1#2%
{%
    \expandafter\XINT_ctf_loop_a\expandafter{#2}#1%
}%
\def\XINT_ctf_end #1.#2#3#4#5{\xintrawwithzeros {#2/#3}}% 1.09b removes [0]
```

27.15. \xintiCstoF

```
\def\xintiCstoF {\romannumeral0\xinticstof }%
\def\xinticstof #1%
{%
    \expandafter\XINT_icstf_prep \romannumeral`&&@#1,!,%
}%
\def\XINT_icstf_prep
{%
    \XINT_icstf_loop_a 1001%
}%
\def\XINT_icstf_loop_a #1#2#3#4#5,%
```

```
{%
    \xint_gob_til_exclam #5\XINT_icstf_end!%
    \expandafter
    \XINT_icstf_loop_b \romannumeral`&&@#5.{#1}{#2}{#3}{#4}%
}%
\def\XINT_icstf_loop_b #1.#2#3#4#5%
{%
    \expandafter\XINT_icstf_loop_c\expandafter
    {\romannumeral0\xintiiadd {#5}{\xintiimul{#1}{#3}}}%
    {\romannumeral0\xintiiadd {#4}{\xintiimul{#1}{#2}}}%
    {#2}{#3}%
}%
\def\XINT_icstf_loop_c #1#2%
{%
    \expandafter\XINT_icstf_loop_a\expandafter {#2}{#1}%
}%
\def\XINT_icstf_end#1.#2#3#4#5{\xintrawwithzeros {#2/#3}}% 1.09b removes [0]
```

27.16. \xintGCtoF

```
\def\xintGCtoF {\romannumeral0\xintgctof }%
\def\xintgctof #1%
{%
    \expandafter\XINT_gctf_prep \romannumeral`&&@#1+!/%
}%
\def\XINT_gctf_prep
{%
    \XINT_gctf_loop_a 1001%
}%
\def\XINT_gctf_loop_a #1#2#3#4#5+%
{%
    \expandafter\XINT_gctf_loop_b
    \romannumeral0\xintrawwithzeros {#5}.{#1}{#2}{#3}{#4}%
}%
\def\XINT_gctf_loop_b #1/#2.#3#4#5#6%
{%
    \expandafter\XINT_gctf_loop_c\expandafter
    {\romannumeral0\xintiimul{#2}{#4}}%
    {\romannumeral0\xintiimul{#2}{#3}}%
    {\romannumeral0\xintiiadd {\xintiimul{#2}{#6}}%
                              {\xintiimul{#1}{#4}}}%
    {\romannumeral0\xintiiadd {\xintiimul{#2}{#5}}%
                              {\xintiimul{#1}{#3}}}%
}%
\def\XINT_gctf_loop_c #1#2%
{%
    \expandafter\XINT_gctf_loop_d\expandafter {\expandafter{#2}{#1}}%
}%
\def\XINT_gctf_loop_d #1#2%
{%
    \expandafter\XINT_gctf_loop_e\expandafter {\expandafter{#2}#1}%
}%
\def\XINT_gctf_loop_e #1#2%
```

```
{%
    \expandafter\XINT_gctf_loop_f\expandafter {\expandafter{#2}#1}%
}%
\def\XINT_gctf_loop_f #1#2/%
{%
    \xint_gob_til_exclam #2\XINT_gctf_end!%
    \expandafter\XINT_gctf_loop_g
    \romannumeral0\xintrawwithzeros {#2}.#1%
}%
\def\XINT_gctf_loop_g #1/#2.#3#4#5#6%
{%
    \expandafter\XINT_gctf_loop_h\expandafter
    {\romannumeral0\xintiimul{#1}{#6}}%
    {\romannumeral0\xintiimul{#1}{#5}}%
    {\romannumeral0\xintiimul{#2}{#4}}%
    {\romannumeral0\xintiimul{#2}{#3}}%
}%
\def\XINT_gctf_loop_h #1#2%
{%
    \expandafter\XINT_gctf_loop_i\expandafter {\expandafter{#2}{#1}}%
}%
\def\XINT_gctf_loop_i #1#2%
{%
    \expandafter\XINT_gctf_loop_j\expandafter {\expandafter{#2}#1}%
}%
\def\XINT_gctf_loop_j #1#2%
{%
    \expandafter\XINT_gctf_loop_a\expandafter {#2}#1%
}%
\def\XINT_gctf_end #1.#2#3#4#5{\xintrawwithzeros {#2/#3}}% 1.09b removes [0]
```

27.17. \xintiGCtoF

```
\def\xintiGCtoF {\romannumeral0\xintigctof }%
\def\xintigctof #1%
{%
    \expandafter\XINT_igctf_prep \romannumeral`&&@#1+!/%
}%
\def\XINT_igctf_prep
{%
    \XINT_igctf_loop_a 1001%
}%
\def\XINT_igctf_loop_a #1#2#3#4#5+%
{%
    \expandafter\XINT_igctf_loop_b
    \romannumeral`&&@#5.{#1}{#2}{#3}{#4}%
}%
\def\XINT_igctf_loop_b #1.#2#3#4#5%
{%
    \expandafter\XINT_igctf_loop_c\expandafter
    {\romannumeral0\xintiiadd {#5}{\xintiimul{#1}{#3}}}%
    {\romannumeral0\xintiiadd {#4}{\xintiimul{#1}{#2}}}%
    {#2}{#3}%
```

```
}%
\def\XINT_igctf_loop_c #1#2%
{%
    \expandafter\XINT_igctf_loop_f\expandafter {\expandafter{#2}{#1}}%
}%
\def\XINT_igctf_loop_f #1#2#3#4/%
{%
    \xint_gob_til_exclam #4\XINT_igctf_end!%
    \expandafter\XINT_igctf_loop_g
    \romannumeral`&&@#4.{#2}{#3}#1%
}%
\def\XINT_igctf_loop_g #1.#2#3%
{%
    \expandafter\XINT_igctf_loop_h\expandafter
    {\romannumeral0\xintiimul{#1}{#3}}%
    {\romannumeral0\xintiimul{#1}{#2}}%
}%
\def\XINT_igctf_loop_h #1#2%
{%
    \expandafter\XINT_igctf_loop_i\expandafter {#2}{#1}%
}%
\def\XINT_igctf_loop_i #1#2#3#4%
{%
    \XINT_igctf_loop_a {#3}{#4}{#1}{#2}%
}%
\def\XINT_igctf_end #1.#2#3#4#5{\xintrawwithzeros {#4/#5}}% 1.09b removes [0]
```

27.18. \xintCtoCv, \xintCstoCv

1.09m uses xinttools's \xintCSVtoList on the argument of \xintCstoCv to allow spaces also before the commas. The original \xintCstoCv code became the one of the new \xintCtoF dealing with a braced rather than comma separated list.

```
\def\xintCstoCv {\romannumeral0\xintcstocv }%
\def\xintcstocv #1%
{%
    \expandafter\XINT_ctcv_prep\romannumeral0\xintcsvtolist{#1}!%
}%
\def\xintCtoCv {\romannumeral0\xintctocv }%
\def\xintctocv #1%
{%
    \expandafter\XINT_ctcv_prep\romannumeral`&&@#1!%
}%
\def\XINT_ctcv_prep
{%
    \XINT_ctcv_loop_a {}1001%
}%
\def\XINT_ctcv_loop_a #1#2#3#4#5#6%
{%
    \xint_gob_til_exclam #6\XINT_ctcv_end!%
    \expandafter\XINT_ctcv_loop_b
    \romannumeral0\xintrawwithzeros {#6}.{#2}{#3}{#4}{#5}{#1}%
}%
```

```
\def\XINT_ctcv_loop_b #1/#2.#3#4#5#6%
{%
    \expandafter\XINT_ctcv_loop_c\expandafter
    {\romannumeral0\xintiimul{#2}{#4}}%
    {\romannumeral0\xintiimul{#2}{#3}}%
    {\romannumeral0\xintiiadd {\xintiimul{#2}{#6}}%
                              {\xintiimul{#1}{#4}}}%
    {\romannumeral0\xintiiadd {\xintiimul{#2}{#5}}%
                              {\xintiimul{#1}{#3}}}%
}%
\def\XINT_ctcv_loop_c #1#2%
{%
    \expandafter\XINT_ctcv_loop_d\expandafter {\expandafter{#2}{#1}}%
}%
\def\XINT_ctcv_loop_d #1#2%
{%
    \expandafter\XINT_ctcv_loop_e\expandafter {\expandafter{#2}#1}%
}%
\def\XINT_ctcv_loop_e #1#2%
{%
    \expandafter\XINT_ctcv_loop_f\expandafter{#2}#1%
}%
\def\XINT_ctcv_loop_f #1#2#3#4#5%
{%
    \expandafter\XINT_ctcv_loop_g\expandafter
    {\romannumeral0\xintrawwithzeros {#1/#2}}{#5}{#1}{#2}{#3}{#4}%
}%
\def\XINT_ctcv_loop_g #1#2{\XINT_ctcv_loop_a {#2{#1}}}% 1.09b removes [0]
\def\XINT_ctcv_end #1.#2#3#4#5#6{ #6}%
```

27.19. \xintiCstoCv

```
\def\xintiCstoCv {\romannumeral0\xinticstocv }%
\def\xinticstocv #1%
{%
    \expandafter\XINT_icstcv_prep \romannumeral`&&@#1,!,%
}%
\def\XINT_icstcv_prep
{%
    \XINT_icstcv_loop_a {}1001%
}%
\def\XINT_icstcv_loop_a #1#2#3#4#5#6,%
{%
    \xint_gob_til_exclam #6\XINT_icstcv_end!%
    \expandafter
    \XINT_icstcv_loop_b \romannumeral`&&@#6.{#2}{#3}{#4}{#5}{#1}%
}%
\def\XINT_icstcv_loop_b #1.#2#3#4#5%
{%
    \expandafter\XINT_icstcv_loop_c\expandafter
    {\romannumeral0\xintiiadd {#5}{\xintiimul{#1}{#3}}}%
    {\romannumeral0\xintiiadd {#4}{\xintiimul{#1}{#2}}}%
    {{#2}{#3}}%
```

```
}%
\def\XINT_icstcv_loop_c #1#2%
{%
    \expandafter\XINT_icstcv_loop_d\expandafter {#2}{#1}%
}%
\def\XINT_icstcv_loop_d #1#2%
{%
    \expandafter\XINT_icstcv_loop_e\expandafter
    {\romannumeral0\xintrawwithzeros {#1/#2}}{{#1}{#2}}%
}%
\def\XINT_icstcv_loop_e #1#2#3#4{\XINT_icstcv_loop_a {#4{#1}}#2#3}%
\def\XINT_icstcv_end #1.#2#3#4#5#6{ #6}%  1.09b removes [0]
```

27.20. \xintGCtoCv

```
\def\xintGCtoCv {\romannumeral0\xintgctocv }%
\def\xintgctocv #1%
{%
    \expandafter\XINT_gctcv_prep \romannumeral`&&@#1+!/%
}%
\def\XINT_gctcv_prep
{%
    \XINT_gctcv_loop_a {}1001%
}%
\def\XINT_gctcv_loop_a #1#2#3#4#5#6+%
{%
    \expandafter\XINT_gctcv_loop_b
    \romannumeral0\xintrawwithzeros {#6}.{#2}{#3}{#4}{#5}{#1}%
}%
\def\XINT_gctcv_loop_b #1/#2.#3#4#5#6%
{%
    \expandafter\XINT_gctcv_loop_c\expandafter
    {\romannumeral0\xintiimul{#2}{#4}}%
    {\romannumeral0\xintiimul{#2}{#3}}%
    {\romannumeral0\xintiiadd {\xintiimul{#2}{#6}}%
                              {\xintiimul{#1}{#4}}}%
    {\romannumeral0\xintiiadd {\xintiimul{#2}{#5}}%
                              {\xintiimul{#1}{#3}}}%
}%
\def\XINT_gctcv_loop_c #1#2%
{%
    \expandafter\XINT_gctcv_loop_d\expandafter {\expandafter{#2}{#1}}%
}%
\def\XINT_gctcv_loop_d #1#2%
{%
    \expandafter\XINT_gctcv_loop_e\expandafter {\expandafter{#2}{#1}}%
}%
\def\XINT_gctcv_loop_e #1#2%
{%
    \expandafter\XINT_gctcv_loop_f\expandafter {#2}#1%
}%
\def\XINT_gctcv_loop_f #1#2%
{%
```

```
    \expandafter\XINT_gctcv_loop_g\expandafter
    {\romannumeral0\xintrawwithzeros {#1/#2}}{{#1}{#2}}%
}%
\def\XINT_gctcv_loop_g #1#2#3#4%
{%
    \XINT_gctcv_loop_h {#4{#1}}{#2#3}% 1.09b removes [0]
}%
\def\XINT_gctcv_loop_h #1#2#3/%
{%
    \xint_gob_til_exclam #3\XINT_gctcv_end!%
    \expandafter\XINT_gctcv_loop_i
    \romannumeral0\xintrawwithzeros {#3}.#2{#1}%
}%
\def\XINT_gctcv_loop_i #1/#2.#3#4#5#6%
{%
    \expandafter\XINT_gctcv_loop_j\expandafter
    {\romannumeral0\xintiimul{#1}{#6}}%
    {\romannumeral0\xintiimul{#1}{#5}}%
    {\romannumeral0\xintiimul{#2}{#4}}%
    {\romannumeral0\xintiimul{#2}{#3}}%
}%
\def\XINT_gctcv_loop_j #1#2%
{%
    \expandafter\XINT_gctcv_loop_k\expandafter {\expandafter{#2}{#1}}%
}%
\def\XINT_gctcv_loop_k #1#2%
{%
    \expandafter\XINT_gctcv_loop_l\expandafter {\expandafter{#2}#1}%
}%
\def\XINT_gctcv_loop_l #1#2%
{%
    \expandafter\XINT_gctcv_loop_m\expandafter {\expandafter{#2}#1}%
}%
\def\XINT_gctcv_loop_m #1#2{\XINT_gctcv_loop_a {#2}#1}%
\def\XINT_gctcv_end #1.#2#3#4#5#6{ #6}%
```

27.21. \xintiGCtoCv

```
\def\xintiGCtoCv {\romannumeral0\xintigctocv }%
\def\xintigctocv #1%
{%
    \expandafter\XINT_igctcv_prep \romannumeral`&&@#1+!/%
}%
\def\XINT_igctcv_prep
{%
    \XINT_igctcv_loop_a {}1001%
}%
\def\XINT_igctcv_loop_a #1#2#3#4#5#6+%
{%
    \expandafter\XINT_igctcv_loop_b
    \romannumeral`&&@#6.{#2}{#3}{#4}{#5}{#1}%
}%
\def\XINT_igctcv_loop_b #1.#2#3#4#5%
```

```
{%
    \expandafter\XINT_igctcv_loop_c\expandafter
    {\romannumeral0\xintiiadd {#5}{\xintiimul{#1}{#3}}}%
    {\romannumeral0\xintiiadd {#4}{\xintiimul{#1}{#2}}}%
    {{#2}{#3}}%
}%
\def\XINT_igctcv_loop_c #1#2%
{%
    \expandafter\XINT_igctcv_loop_f\expandafter {\expandafter{#2}{#1}}%
}%
\def\XINT_igctcv_loop_f #1#2#3#4/%
{%
    \xint_gob_til_exclam #4\XINT_igctcv_end_a!%
    \expandafter\XINT_igctcv_loop_g
    \romannumeral`&&@#4.#1#2{#3}%
}%
\def\XINT_igctcv_loop_g #1.#2#3#4#5%
{%
    \expandafter\XINT_igctcv_loop_h\expandafter
    {\romannumeral0\xintiimul{#1}{#5}}%
    {\romannumeral0\xintiimul{#1}{#4}}%
    {{#2}{#3}}%
}%
\def\XINT_igctcv_loop_h #1#2%
{%
    \expandafter\XINT_igctcv_loop_i\expandafter {\expandafter{#2}{#1}}%
}%
\def\XINT_igctcv_loop_i #1#2{\XINT_igctcv_loop_k #2{#2#1}}%
\def\XINT_igctcv_loop_k #1#2%
{%
    \expandafter\XINT_igctcv_loop_l\expandafter
    {\romannumeral0\xintrawwithzeros {#1/#2}}%
}%
\def\XINT_igctcv_loop_l #1#2#3{\XINT_igctcv_loop_a {#3{#1}}#2}%1.09i removes [0]
\def\XINT_igctcv_end_a #1.#2#3#4#5%
{%
    \expandafter\XINT_igctcv_end_b\expandafter
    {\romannumeral0\xintrawwithzeros {#2/#3}}%
}%
\def\XINT_igctcv_end_b #1#2{ #2{#1}}% 1.09b removes [0]
```

27.22. \xintFtoCv

Still uses \xinticstocv \xintFtoCs rather than \xintctocv \xintFtoC.

```
\def\xintFtoCv {\romannumeral0\xintftocv }%
\def\xintftocv #1%
{%
    \xinticstocv {\xintFtoCs {#1}}%
}%
```

27.23. \xintFtoCCv

```
\def\xintFtoCCv {\romannumeral0\xintftoccv }%
```

```
\def\xintftoccv #1%
{%
    \xintigctocv {\xintFtoCC {#1}}%
}%
```

27.24. \xintCntoF

Modified in 1.06 to give the N first to a \numexpr rather than expanding twice. I just use \the\numexpr and maintain the previous code after that.

```
\def\xintCntoF {\romannumeral0\xintcntof }%
\def\xintcntof #1%
{%
    \expandafter\XINT_cntf\expandafter {\the\numexpr #1}%
}%
\def\XINT_cntf #1#2%
{%
   \ifnum #1>\xint_c_
      \xint_afterfi {\expandafter\XINT_cntf_loop\expandafter
                     {\the\numexpr #1-1\expandafter}\expandafter
                     {\romannumeral`&&@#2{#1}}{#2}}%
   \else
      \xint_afterfi
         {\ifnum #1=\xint_c_
              \xint_afterfi {\expandafter\space \romannumeral`&&@#2{0}}%
          \else \xint_afterfi { }% 1.09m now returns nothing.
          \fi}%
   \fi
}%
\def\XINT_cntf_loop #1#2#3%
{%
    \ifnum #1>\xint_c_ \else \XINT_cntf_exit \fi
    \expandafter\XINT_cntf_loop\expandafter
    {\the\numexpr #1-1\expandafter }\expandafter
    {\romannumeral0\xintadd {\xintDiv {1[0]}{#2}}{#3{#1}}}%
    {#3}%
}%
\def\XINT_cntf_exit \fi
    \expandafter\XINT_cntf_loop\expandafter
    #1\expandafter #2#3%
{%
    \fi\xint_gobble_ii #2%
}%
```

27.25. \xintGCntoF

Modified in 1.06 to give the N argument first to a \numexpr rather than expanding twice. I just use \the\numexpr and maintain the previous code after that.

```
\def\xintGCntoF {\romannumeral0\xintgcntof }%
\def\xintgcntof #1%
{%
    \expandafter\XINT_gcntf\expandafter {\the\numexpr #1}%
}%
```

```
\def\XINT_gcntf #1#2#3%
{%
   \ifnum #1>\xint_c_
      \xint_afterfi {\expandafter\XINT_gcntf_loop\expandafter
                     {\the\numexpr #1-1\expandafter}\expandafter
                     {\romannumeral`&&@#2{#1}}{#2}{#3}}%
   \else
      \xint_afterfi
         {\ifnum #1=\xint_c_
              \xint_afterfi {\expandafter\space\romannumeral`&&@#2{0}}%
          \else \xint_afterfi { }% 1.09m now returns nothing rather than 0/1[0]
          \fi}%
   \fi
}%
\def\XINT_gcntf_loop #1#2#3#4%
{%
    \ifnum #1>\xint_c_ \else \XINT_gcntf_exit \fi
    \expandafter\XINT_gcntf_loop\expandafter
    {\the\numexpr #1-1\expandafter }\expandafter
    {\romannumeral0\xintadd {\xintDiv {#4{#1}}{#2}}{#3{#1}}}%
    {#3}{#4}%
}%
\def\XINT_gcntf_exit \fi
    \expandafter\XINT_gcntf_loop\expandafter
    #1\expandafter #2#3#4%
{%
    \fi\xint_gobble_ii #2%
}%
```

27.26. \xintCntoCs

Modified in 1.09m: added spaces after the commas in the produced list. Moreover the coefficients are not braced anymore. A slight induced limitation is that the macro argument should not contain some explicit comma (cf. \XINT_cntcs_exit_b), hence \xintCntoCs {\macro,} with \def\macro,#1{<stuff>} would crash. Not a very serious limitation, I believe.

```
\def\xintCntoCs {\romannumeral0\xintcntocs }%
\def\xintcntocs #1%
{%
    \expandafter\XINT_cntcs\expandafter {\the\numexpr #1}%
}%
\def\XINT_cntcs #1#2%
{%
   \ifnum #1<0
      \xint_afterfi { }% 1.09i: a 0/1[0] was here, now the macro returns nothing
   \else
      \xint_afterfi {\expandafter\XINT_cntcs_loop\expandafter
                     {\the\numexpr #1-\xint_c_i\expandafter}\expandafter
                     {\romannumeral`&&@#2{#1}}{#2}}% produced coeff not braced
   \fi
}%
\def\XINT_cntcs_loop #1#2#3%
{%
```

```
    \ifnum #1>-\xint_c_i \else \XINT_cntcs_exit \fi
    \expandafter\XINT_cntcs_loop\expandafter
    {\the\numexpr #1-\xint_c_i\expandafter}\expandafter
    {\romannumeral`&&@#3{#1}, #2}{#3}% space added, 1.09m
}%
\def\XINT_cntcs_exit \fi
    \expandafter\XINT_cntcs_loop\expandafter
    #1\expandafter #2#3%
{%
    \fi\XINT_cntcs_exit_b #2%
}%
\def\XINT_cntcs_exit_b #1,{}% romannumeral stopping space already there
```

27.27. \xintCntoGC

Modified in 1.06 to give the N first to a \numexpr rather than expanding twice. I just use \the\numexpr and maintain the previous code after that.

1.09m maintains the braces, as the coeff are allowed to be fraction and the slash can not be naked in the GC format, contrarily to what happens in \xintCntoCs. Also the separators given to \xintGCtoGCx may then fetch the coefficients as argument, as they are braced.

```
\def\xintCntoGC {\romannumeral0\xintcntogc }%
\def\xintcntogc #1%
{%
    \expandafter\XINT_cntgc\expandafter {\the\numexpr #1}%
}%
\def\XINT_cntgc #1#2%
{%
   \ifnum #1<0
      \xint_afterfi { }% 1.09i there was as strange 0/1[0] here, removed
   \else
      \xint_afterfi {\expandafter\XINT_cntgc_loop\expandafter
                     {\the\numexpr #1-\xint_c_i\expandafter}\expandafter
                     {\expandafter{\romannumeral`&&@#2{#1}}}{#2}}%
   \fi
}%
\def\XINT_cntgc_loop #1#2#3%
{%
    \ifnum #1>-\xint_c_i \else \XINT_cntgc_exit \fi
    \expandafter\XINT_cntgc_loop\expandafter
    {\the\numexpr #1-\xint_c_i\expandafter }\expandafter
    {\expandafter{\romannumeral`&&@#3{#1}}+1/#2}{#3}%
}%
\def\XINT_cntgc_exit \fi
    \expandafter\XINT_cntgc_loop\expandafter
    #1\expandafter #2#3%
{%
    \fi\XINT_cntgc_exit_b #2%
}%
\def\XINT_cntgc_exit_b #1+1/{ }%
```

27.28. \xintGCntoGC

Modified in 1.06 to give the N first to a \numexpr rather than expanding twice. I just use \the\numexpr and maintain the previous code after that.

```
\def\xintGCntoGC {\romannumeral0\xintgcntogc }%
\def\xintgcntogc #1%
{%
    \expandafter\XINT_gcntgc\expandafter {\the\numexpr #1}%
}%
\def\XINT_gcntgc #1#2#3%
{%
   \ifnum #1<0
      \xint_afterfi { }% 1.09i now returns nothing
   \else
      \xint_afterfi {\expandafter\XINT_gcntgc_loop\expandafter
                     {\the\numexpr #1-\xint_c_i\expandafter}\expandafter
                     {\expandafter{\romannumeral`&&@#2{#1}}}{#2}{#3}}%
   \fi
}%
\def\XINT_gcntgc_loop #1#2#3#4%
{%
    \ifnum #1>-\xint_c_i \else \XINT_gcntgc_exit \fi
    \expandafter\XINT_gcntgc_loop_b\expandafter
    {\expandafter{\romannumeral`&&@#4{#1}}/#2}{#3{#1}}{#1}{#3}{#4}%
}%
\def\XINT_gcntgc_loop_b #1#2#3%
{%
    \expandafter\XINT_gcntgc_loop\expandafter
    {\the\numexpr #3-\xint_c_i \expandafter}\expandafter
    {\expandafter{\romannumeral`&&@#2}+#1}%
}%
\def\XINT_gcntgc_exit \fi
    \expandafter\XINT_gcntgc_loop_b\expandafter #1#2#3#4#5%
{%
    \fi\XINT_gcntgc_exit_b #1%
}%
\def\XINT_gcntgc_exit_b #1/{ }%
```

27.29. \xintCstoGC

```
\def\xintCstoGC {\romannumeral0\xintcstogc }%
\def\xintcstogc #1%
{%
    \expandafter\XINT_cstc_prep \romannumeral`&&@#1,!,%
}%
\def\XINT_cstc_prep #1,{\XINT_cstc_loop_a {{#1}}}%
\def\XINT_cstc_loop_a #1#2,%
{%
    \xint_gob_til_exclam #2\XINT_cstc_end!%
    \XINT_cstc_loop_b {#1}{#2}%
}%
\def\XINT_cstc_loop_b #1#2{\XINT_cstc_loop_a {#1+1/{#2}}}%
\def\XINT_cstc_end!\XINT_cstc_loop_b #1#2{ #1}%
```

27.30. \xintGCtoGC

```
\def\xintGCtoGC {\romannumeral0\xintgctogc }%
\def\xintgctogc #1%
{%
    \expandafter\XINT_gctgc_start \romannumeral`&&@#1+!/%
}%
\def\XINT_gctgc_start {\XINT_gctgc_loop_a {}}%
\def\XINT_gctgc_loop_a #1#2+#3/%
{%
    \xint_gob_til_exclam #3\XINT_gctgc_end!%
    \expandafter\XINT_gctgc_loop_b\expandafter
    {\romannumeral`&&@#2}{#3}{#1}%
}%
\def\XINT_gctgc_loop_b #1#2%
{%
    \expandafter\XINT_gctgc_loop_c\expandafter
    {\romannumeral`&&@#2}{#1}%
}%
\def\XINT_gctgc_loop_c #1#2#3%
{%
    \XINT_gctgc_loop_a {#3{#2}+{#1}/}%
}%
\def\XINT_gctgc_end!\expandafter\XINT_gctgc_loop_b
{%
    \expandafter\XINT_gctgc_end_b
}%
\def\XINT_gctgc_end_b #1#2#3{ #3{#1}}%
\XINTrestorecatcodesendinput%
```

28. Package xintexpr implementation

Added at calx 2.0a1.

Contents

.1 Catcodes, ε-TEX and reload detection . . 493
.2 Package identification 494
.3 \XINTinFloatdigits, \XINTinFloatSdigits 494
.4 Top level user TEX interface: \xinteval . 494
.4.1 \xintexpr, \xintiexpr 495
.4.2 \XINT_expr_wrap, \XINT_iexpr_wrap . . . 496
.4.3 \XINTexprprint, \XINTiexprprint 496
.4.4 \xinttheexpr, \xinttheiexpr 496
.4.5 \xintbareeval 496
.4.6 \xintthebareeval 496
.4.7 \xinteval, \xintieval 497
.4.8 \xintboolexpr, \XINT_boolexpr_print, \xintboolexpr 497
.4.9 \xintifboolexpr 497
.4.10 \xintifsgnexpr 498
.4.11 \xint_FracToSci_x 498
.5 Raw input of the float operations 500
.6 Expansion spanning; opening and closing parentheses 500
.7 The comma as binary operator 500
.8 The minus as prefix operator of variable precedence level 501
.9 The * as Python-like «unpacking» prefix operator 501
.10 \xintPow 501
.10.1 &&, ||, //, /:, +, −, *, /, ^, **, 'and', 'or', 'xor', and 'mod' 502
.10.2 .., ..[, and].. for a..b and a..[b]..c syntax . 503
.10.3 <, >, ==, <=, >=, != with Python-like chaining 504
.10.4 [...] as «oneple» constructor 504
.10.5 [...] brackets and : operator for NumPy-like slicing and item indexing syntax 504
.11 Support for raw A/B[N] 504
.12 ? as two-way and ?? as three-way «short-circuit» conditionals 504
.13 ! as postfix factorial operator 505
.14 User defined variables: \xintdefvar . . . 505
.15 \qfrac() of \xinteval 505
.16 Regular built-in functions 505
.17 User declared functions 509
.17.1 \xintunassignexprfunc 510
.17.2 \xintNewExpr, \xintNewIExpr, \xintNewBoolExpr, \xintNewFloatExpr, \xintNewIIExpr 510

xintexpr.sty: **WIP** Currently an empty wrapper loading calx.sty and xintfrac.sty. In future \xinteval will be defined by this file not by calx.sty.

28.1. Catcodes, ε-TEX and reload detection

```
\begingroup\catcode61\catcode48\catcode32=10\relax%
  \catcode13=5    % ^^M
  \endlinechar=13 %
  \catcode123=1   % {
  \catcode125=2   % }
  \catcode64=11   % @
  \catcode44=12   % ,
  \catcode46=12   % .
  \catcode58=12   % :
  \catcode94=7    % ^
  \catcode35=6    % #
  \def\odef#1{\expandafter\def\expandafter#1\expandafter}%
  \def\empty{}\def\space{ }\newlinechar10
  \def\z{\endgroup}%
  \expandafter\let\expandafter\x\csname ver@xintexpr.sty\endcsname
  \expandafter\let\expandafter\f\csname ver@xintfrac.sty\endcsname
  \expandafter\let\expandafter\c\csname ver@calx.sty\endcsname
  % I assume engines do not exist providing \expanded but not \numexpr
  \expandafter\ifx\csname expanded\endcsname\relax
```

```
    \expandafter\ifx\csname PackageWarningNoLine\endcsname\relax
      \immediate\write128{^^JPackage xintexpr Warning:^^J%
                          \space\space\space\space
                          \expanded not available, aborting input.^^J}%
    \else
      \PackageWarningNoLine{xintexpr}{\expanded not available, aborting input}%
    \fi
    \def\z{\endgroup\endinput}%
  \else
    \ifx\x\relax   % not LaTeX (perhaps Plain+miniltx), first loading of xintexpr.sty
      \ifx\c\relax % but calx.sty not yet loaded.
         \odef\z{\z\input calx.sty\relax}%
      \fi
      \ifx\f\relax % but xintfrac.sty not yet loaded (made miniltx robust 2022/06/09)
         \odef\z{\z\input xintfrac.sty\relax}%
      \fi
    \else
      \ifx\x\empty % LaTeX, first loading,
      % variable is initialized, but \ProvidesPackage not yet seen
          \ifx\c\relax % calx not yet loaded.
            \odef\z{\z\RequirePackage{calx}}%
          \fi
          \ifx\f\relax % xintfrac not yet loaded.
            \odef\z{\z\RequirePackage{xintfrac}}%
          \fi
      \else
        \def\z{\endgroup\endinput}% xintexpr already loaded.
      \fi
    \fi
  \fi
\z%
\XINTsetupcatcodes%
```

28.2. Package identification

```
\XINT_providespackage
\ProvidesPackage{xintexpr}%
  [2026/11/26 v2.0a2 Numeric and symbolic expression parsers (JFB)]%
\catcode`! 11
```

28.3. \XINTinFloatdigits, \XINTinFloatSdigits

```
\def\XINTinFloatdigits {\XINTinFloat [\XINTdigits]}%
\def\XINTinFloatSdigits{\XINTinFloatS[\XINTdigits]}%
```

28.4. Top level user TEX interface: \xinteval

Contents

.4.1 \xintexpr, \xintiexpr 495
.4.2 \XINT_expr_wrap, \XINT_iexpr_wrap 496
.4.3 \XINTexprprint, \XINTiexprprint 496
.4.4 \xinttheexpr, \xintttheiexpr 496
.4.5 \xintbareeval 496

.4.6 \xintthebareeval 496
.4.7 \xinteval, \xintieval 497
.4.8 \xintboolexpr, \XINT_boolexpr_print, \xinttheboolexpr 497
.4.9 \xintifboolexpr 497
.4.10 \xintifsgnexpr 498
.4.11 \xint_FracToSci_x 498

28.4.1. \xintexpr, \xintiexpr

\xintiexpr has an optional argument since 1.1.

ATTENTION! 1.3d renamed \xinteval to \xintexpro etc... (thus this old \xinteval isn't our current \xinteval).

Usage of \xintiRound{0} for \xintiexpr without optional [D] means that \xintiexpr ... \relax wrapper can be used to insert rounded-to-integers values in \xintiiexpr context: no post-fix [0] which would break it. ^^^^^^ REVISE THE ABOVE BECAUSE NOW WE HAVE [0] ^^^^^

1.4a add support for the optional argument [D] for \xintiexpr being negative D, with same meaning as the 1.4a modified \xintRound from xintfrac.sty.

\xintiexpr mechanism was refactored at 1.4e so that rounding due to [D] optional argument uses raw format, not fixed point format on output, delegating fixed point conversion to an \XINTiexprprint now separated from \XINTexprprint.

In case of negative [D], \xintiexpr [D]...\relax internally has the [0] post-fix so it can not be inserted as sub-expression in \xintiiexpr without a num() or \xintiexpr ...\relax (extra) wrapper.

Modified at 1.4o (2025/09/06).

```
\def\xintexpr       {\romannumeral0\xintexpro      }%
\def\xintiexpr      {\romannumeral0\xintiexpro     }%
\def\xintexpro   {\csname XINT_expr_wrap\expandafter\endcsname
                  \romannumeral0\xintbareeval }%
\def\xintiexpro #1%
{%
    \ifx [#1\expandafter\XINT_iexpr_withopt\else\expandafter\XINT_iexpr_noopt
    \fi #1%
}%
\def\XINT_iexpr_noopt
{%
    \csname XINT_iexpr_iiround\expandafter\endcsname\romannumeral0\xintbareeval
}%
\def\XINT_iexpr_iiround
{%
    \expandafter\XINT_expr_wrap
    \expanded
    \XINT:NEhook:x:mapwithin\XINT:expr:mapwithin{\XINTiRoundzero_braced}%
}%
```

FIX THIS (for calx 2.0a1)

```
\def\XINTiRoundzero_braced#1{{\xintiRound{0}{#1}}}%
\def\XINT_iexpr_withopt [#1]%
{%
    \csname XINT_iexpr_round\expandafter\endcsname
    \the\numexpr \xint_zapspaces #1 \xint_gobble_i\expandafter.%
    \romannumeral0\xintbareeval
}%
\def\XINT_iexpr_round #1.%
```

```
{%
    \ifnum#1=\xint_c_\xint_dothis{\XINT_iexpr_iiround}\fi
    \xint_orthat{\XINT_iexpr_round_a #1.}%
}%
\def\XINT_iexpr_round_a #1.%
{%
    \expandafter\XINT_iexpr_wrap
    \expanded
    \XINT:NEhook:x:mapwithin\XINT:expr:mapwithin{\XINTiRound_braced{#1}}%
}%
\def\XINTiRound_braced#1#2%
  {{\xintiRound{#1}{#2}[\the\numexpr\ifnum#1<\xint_c_i0\else-#1\fi]}}%
```

28.4.2. \XINT_expr_wrap, \XINT_iexpr_wrap

1.3e removes some leading space tokens which served nothing.

1.4e has \XINT_iexpr_wrap separated from \XINT_expr_wrap, thus simplifying internal matters as output printer for \xintexpr will not have to handle fixed point input but only extended-raw type input (i.e. A, A/B, A[N] or A/B[N]).

```
\def\XINT_expr_wrap    {\XINTfstop\XINTexprprint.}%
\def\XINT_iexpr_wrap   {\XINTfstop\XINTiexprprint.}%
```

28.4.3. \XINTexprprint, \XINTiexprprint

See comments in calx.

```
\protected\def\XINTexprprint.%
    {\XINT:NEhook:x:toblist\XINT:expr:toblistwith\xintexprPrintOne}%
\protected\def\XINTiexprprint.%
    {\XINT:NEhook:x:toblist\XINT:expr:toblistwith\xintiexprPrintOne}%
\let\xintiexprPrintOne\xintDecToString
\protected\def\XINTboolexprprint.%
    {\XINT:NEhook:x:toblist\XINT:expr:toblistwith\xintboolexprPrintOne}%
\def\xintboolexprPrintOne#1{\calxifNotZero{#1}{true}{false}}%
```

28.4.4. \xinttheexpr, \xinttheiexpr

See comments in calx.

```
\def\xinttheexpr
   {\expanded\csname XINTexprprint\expandafter\endcsname
    \expandafter.\romannumeral0\xintbareeval}%
\def\xinttheiexpr{\expanded\expandafter\xint_gobble_i\romannumeral0\xintiexpro}%
```

28.4.5. \xintbareeval

See comments in calx.

```
\def\xintbareeval     {\XINT_expr_start  }%
```

28.4.6. \xintthebareeval

See comments in calx.

```
\def\xintthebareeval{%
  \romannumeral0\csname xint_firstofone\expandafter\endcsname
  \romannumeral0\xintbareeval
```

```
}%
```

28.4.7. \xinteval, \xintieval

See comments in calx.

```
\def\xinteval #1{%
    \expanded\csname XINTexprprint\expandafter\endcsname
    \expandafter.\romannumeral0\xintbareeval#1\relax
}%
```

Should I use #{ rather delimited argument? (as for \xintiieval at 1.4n).

```
\def\xintieval{\expanded\expandafter\xint_ieval_chkopt\string}%
\def\xint_ieval_chkopt #1%
{%
    \ifx [#1\expandafter\xint_ieval_opt
       \else\expandafter\xint_ieval_noopt
    \fi  #1%
}%
```

Modified at 1.4p (2026/03/01). Here too I don't know why I had been using \romannumeral`&&@\xintiexpr. Replaced with \romannumeral0\xintiexpro. And I couldn't resist the temptation to immediately jump to _withopt. I did resist replacing _withopt with its expansion... this expansion will inject the \csname for Babel.

```
\def\xint_ieval_opt [#1]#2{%
    \expandafter\xint_gobble_i\romannumeral0\XINT_iexpr_withopt[#1]#2\relax
}%
\def\xint_ieval_noopt #1{\expandafter\xint_ieval\expandafter{\iffalse}\fi}%
```

Attention that here the optional argument can come from the expression itself (legacy syntax).

```
\def\xint_ieval#1{%
    \expandafter\xint_gobble_i\romannumeral0\xintiexpro#1\relax
}%
```

28.4.8. \xintboolexpr, \XINT_boolexpr_print, \xinttheboolexpr

Attention, the conversion to 1 or 0 is done only by the print macro. Perhaps I should force it also inside raw result.

```
\def\xintboolexpr{%
    \romannumeral0\expandafter\XINT_boolexpr_done\romannumeral0\xintexpro
}%
\def\XINT_boolexpr_done #1.{\XINTfstop\XINTboolexprprint.}%
```

Modified at 1.4o (2025/09/06). A \csname for \xinttheboolexpr for Babel.

```
\def\xinttheboolexpr{%
    \expanded\csname XINTboolexprprint\expandafter\endcsname
    \expandafter.\romannumeral0\xintbareeval
}%
```

28.4.9. \xintifboolexpr

See comments in calx. FIXME CHECK STATUS OF \xintiiifnotzero WHOSE USAGE HERE WAS AN ABUSE ORIGINALLY

```
\def\xintifboolexpr #1{%
    \romannumeral0\expandafter\CALX_ifnz
```

```
    \romannumeral0\csname xint_firstofone\expandafter\endcsname
    \romannumeral0\xintbareeval #1\relax
}%
```

28.4.10. \xintifsgnexpr

```
\def\xintifsgnexpr #1{%
    \romannumeral0\expandafter\CALX_ifsgn
    \romannumeral0\csname xint_firstofone\expandafter\endcsname
    \romannumeral0\xintbareeval #1\relax
}%
```

28.4.11. \xint_FracToSci_x

Added at 1.4 (2020/01/31). TODO CLEAN UP COMMENTS WHICH HAVE MULTIPLE OBSOLETE COMPONENTS Under the name of \xintFracToSci and defined in xintfrac.

Modified at 1.4e (2021/05/05). Refactored and much simplified

It only needs to be x-expandable, and indeed the implementation here is only x-expandable.

Finally for 1.4e release I modify. This is breaking change for all \xinteval output in case of scientific notation: it will not be with an integer mantissa, but with regular scientific notation, using the same rules as \xintPFloat.

Of course no float rounding! Also, as [0] will always or almost always be present from an \xinteval, we want then to use integer not scientific notation. But expression contained decimal fixed point input, or uses scientific functions, then probably the N will not be zero and this will trigger usage of scientific notation in output.

Implementing these changes sort of ruin our previous efforts to minimize grabbing the argument, but well. So the rules now are

Input: A, A/B, A[N], A/B[N]

Output: A, A/B, A if N=0, A/B if N=0

If N is not zero, scientific notation like \xintPFloat, i.e. behaviour like \xintfloateval apart from the rounding to significands of width Digits. At 1.4k, trimming of zeros from A is always done, i.e. the \xintPFloatMinTrimmed is ignored to keep behaviour. unchanged. Trailing zeros of B are kept as is.

The zero gives 0, except in A[N] and A/B[N] cases, it may give 0.0

Modified at 1.4k (2022/05/18). Moved from xintfrac to xintexpr.

Modified at 1.4l (2022/05/29). Renamed to \xint_FracToSci_x to make it private and provide in xintfrac another \xintFracToSci with same output but which behaves like other macros there: f-expandable and accepting the whole range of inputs accepted by the xintfrac public macros.

The private x-expandable macro here will have an empty output for an empty input but is never used for an empty input (see \xintexprEmptyItem).

```
\def\xint_FracToSci_x #1%
   {\expandafter\CALX_fts\romannumeral`&&@#1/\W[\R}%
\def\XINT_FracToSci_x #1/#2#3[#4%
{%
    \xint_gob_til_W #2\XINT_FracToSci_x_noslash\W
    \xint_gob_til_R #4\XINT_FracToSci_x_slash_noN\R
    \XINT_FracToSci_x_slash_N #1/#2#3[#4%
}%
\def\XINT_FracToSci_x_noslash#1\XINT_FracToSci_x_slash_N #2[#3%
{%
    \xint_gob_til_R #3\XINT_FracToSci_x_noslash_noN\R
    \XINT_FracToSci_x_noslash_N #2[#3%
```

```
}%
\def\XINT_FracToSci_x_noslash_noN\R\XINT_FracToSci_x_noslash_N #1/\W[\R{#1}%
\def\XINT_FracToSci_x_noslash_N #1[#2]/\W[\R%
{%
    \ifnum#2=\xint_c_ #1\else
      \romannumeral0\expandafter\XINT_pfloat_a_fork\romannumeral0\xintrez{#1[#2]}%
    \fi
}%
\def\XINT_FracToSci_x_slash_noN\R\XINT_FracToSci_x_slash_N #1#2/#3/\W[\R%
{%
    #1\xint_gob_til_zero#1\expandafter\iffalse\xint_gobble_ii0\iftrue
    #2\if\XINT_isOne{#3}1\else/#3\fi\fi
}%
\def\XINT_FracToSci_x_slash_N #1#2/#3[#4]/\W[\R%
{%
    \ifnum#4=\xint_c_ #1#2\else
      \romannumeral0\expandafter\XINT_pfloat_a_fork\romannumeral0\xintrez{#1#2[#4]}%
    \fi
    \if\XINT_isOne{#3}1\else\if#10\else/#3\fi\fi
}%
```

\xint_FracToSci_x also prints the native values of calx which reach \xinteval (nested \xintfloatexpr or \xintiiexpr, float or ii variables and functions): \CALX_fts tests the first token, a tag has character code above `9. I+, I-, I? are printed as they are; an M value is converted by calxint's \CALX_mtoraw (all digits), an F value by calxfp's \CALX_ftoraw (A[N]), then printed by \XINT_FracToSci_x.

```
\def\CALX_fts #1%
   {\ifnum`#1>57 \expandafter\CALX_fts_t\fi\XINT_FracToSci_x #1}%
\def\CALX_fts_t\XINT_FracToSci_x #1#2/\W[\R
   {\if I#1\xint_dothis{#1#2}\fi
    \if M#1\xint_dothis{\expandafter\XINT_FracToSci_x
       \romannumeral0\CALX_mtoraw{#1#2}/\W[\R}\fi
    \xint_orthat{\expandafter\XINT_FracToSci_x
       \romannumeral0\CALX_ftoraw{#1#2}/\W[\R}}%
\let\xintexprPrintOne\xint_FracToSci_x
```

See calx.

```
\def\XINT_tmpa #1#2#3{%
  \def #1##1%
  {%
    \csname
      XINT_\ifcsname XINT_#3_func_##1\endcsname
        #3_func_##1\expandafter\endcsname\romannumeral`&&@\expandafter#2%
      \romannumeral\else
      \ifcsname XINT_#3_onliteral_##1\endcsname
        #3_onliteral_##1\expandafter\endcsname\romannumeral
      \else
      \ifcsname XINT_expr_var*_##1\endcsname
        expr_var*_##1\expandafter\endcsname\romannumeral
      \else
        #3_func_\XINT_expr_unknown_function {##1}%
                \expandafter\endcsname\romannumeral`&&@\expandafter#2%
      \romannumeral
      \fi\fi\fi\xint_c_
```

```
  }%
}%
    \expandafter\XINT_tmpa
                \csname XINT_expr_op_`\endcsname
                \XINT_expr_oparen
                {expr}%
```

28.5. Raw input of the float operations

Syntax (as in calxfp, where \CALX_fr_x is the raw input branch of \CALX_fround):
  \CALX_fr_x <P>.<X>\xint:
        -> <space>Y\xint: (Y: X rounded to P digits, native float)
calxfp accepts as raw values only A[N], A/B[N] (integers A, B without leading zeros, B > 0). With xintexpr.sty, X is any xintfrac input, as in xint 1.4: the arguments of macros made by \xintNewExpr and similar, the argument of qfloat() and the arguments of the \calxFp macros are not parsed by the expression parser. xintfrac's \XINTinFloat rounds X to P digits (its A[N] branch does not look for a decimal mark or an exponent), \CALX_ffromraw converts the A[N] result, and calxfp's \CALX_fr_f finishes as for a native input.

```
\def\CALX_fr_x #1.#2\xint:%
   {\expandafter\CALX_fr_f\the\numexpr#1\expandafter.%
    \romannumeral`&&@\CALX_ffromraw{\XINTinFloat[#1]{#2}}\xint:}%
```

Integer exponent detection for non-native input: use the legacy format normalization (trailing zeros only, no division or GCD).

```
\def\CALX_fip_raw #1\xint:%
   {\expandafter\CALX_fip_norm\romannumeral0\xintrez{#1}\xint:}%
```

28.6. Expansion spanning; opening and closing parentheses

See comments in calx.

```
\catcode`) 11
    \expandafter\XINT_def_start_and_check
    \csname XINT_expr_start\expandafter\endcsname
    \csname XINT_expr_check\expandafter\endcsname
    \csname XINT_expr_checkp\expandafter\endcsname
    \csname XINT_expr_op_-xii\expandafter\endcsname
    \csname XINT_expr_extra_)\endcsname
    {expr}%
    \expandafter\XINT_defop_oparen
    \csname XINT_expr_op_(\expandafter\endcsname
    \csname XINT_expr_op_)\expandafter\endcsname
    \csname XINT_expr_oparen\expandafter\endcsname
    \csname XINT_expr_check-_)\expandafter\endcsname
    \csname XINT_expr_checkp_)\expandafter\endcsname
    \csname XINT_expr_op_-xii\endcsname
    {expr}%
\catcode`) 12
```

28.7. The comma as binary operator

New with 1.09a. Refactored at 1.4.

```
\expandafter\XINT_defop_comma
    \csname XINT_expr_op_,\expandafter\endcsname
```

```
    \csname XINT_expr_exec_,\expandafter\endcsname
    \csname XINT_expr_check-_,\expandafter\endcsname
    \csname XINT_expr_checkp_,\expandafter\endcsname
    \csname XINT_expr_op_-xii\endcsname {expr}%
```

28.8. The minus as prefix operator of variable precedence level

```
\xintApplyInline{\XINT_defop_minus {expr}\xintOpp}{{xii}{xiv}{xvii}}%
```

28.9. The * as Python-like «unpacking» prefix operator

```
    \expandafter\XINT_defpost_unpack\csname XINT_expr_op_0\expandafter\endcsname
                            \csname XINT_expr_until_unpack\endcsname {expr}%
```

28.10. \xintPow

`\xintPow{a}{b}` -> raw value of a^b
xintexpr.sty redefines xintfrac's `\xintPow`: `^`, `**` and `pow()` of `\xinteval` use it. b = 0: 1[0]. b an integer: exact power (xintfrac) if numerator and denominator of the result have at most about 10000 digits, else xintfrac's float power at `\XINTdigits`. b not an integer (half-integers included): `\calxFpPow`, the power of `\xintfloateval`, its result converted to raw. a = 0, b < 0: InvalidOperation, value 0 (as 1/0 in `\xinteval`).

```
\def\xintPow{\romannumeral0\xintpow}%
\def\xintpow#1#2%
{%
    \expandafter\XINT_scipow_a\romannumeral0\xintrez{#2}\XINT_pow_int{#1}%
}%
\def\XINT_scipow_a #1%
{%
    \xint_gob_til_zero#1\XINT_scipow_Biszero0\XINT_scipow_b#1%
}%
\def\XINT_scipow_Biszero#1]#2#3{ 1[0]}%
\def\XINT_scipow_b #1#2/#3[#4]#5%
{%
    \unless\if1\XINT_is_One#3XY\xint_dothis\XINT_scipow_c\fi
    \ifnum#4<\xint_c_ \xint_dothis\XINT_scipow_c\fi
    \xint_orthat#5#1#2/#3[#4]%
}%
\def\XINT_scipow_c #1[#2]#3%
{%
    \ifnum\xintSgn{#3}=\xint_c_ \expandafter\XINT_scipow_z
    \else\expandafter\XINT_scipow_e\fi #1[#2]{#3}%
}%
\def\XINT_scipow_e #1[#2]#3{\CALX_ftoraw{\calxFpPow{#3}{#1[#2]}}}%
\def\XINT_scipow_z #1#2[#3]#4%
{%
    \if-#1\xint_dothis{\XINT_legacysignalcondition{InvalidOperation}%
        {0 raised to power #1#2[#3].}{}{ 0[0]}}\fi
    \xint_orthat{ 0[0]}%
}%
\def\XINT_pow_int #1/1[#2]%
{%
  \expandafter\XINT_pow_int_a\romannumeral0\XINT_dsx_addzeros{#2}#1;.%
```

```
}%
\def\XINT_pow_int_a #1#2.%
{%
  \ifnum\if-#1\xintLength{#2}\else\xintLength{#1#2}\fi>\xint_c_iv
       \expandafter\XINT_pow_bigint
  \else\expandafter\XINT_pow_int_b
  \fi #1#2.%
}%
\def\XINT_pow_bigint #1.#2%
{%
    \XINT_flpower_checkB_a#1.\XINTdigits.{#2}{\XINTinfloatS[\XINTdigits]}%
}%
\def\XINT_pow_int_b #1.#2%
{%
    \expandafter\XINT_pow_int_c\romannumeral0\xintraw{#2}\xint:#1\xint:
}%
\def\XINT_pow_int_c#1#2/#3[#4]\xint:#5\xint:
{%
    \if0\ifnum\numexpr\xint_c_e_iv/%
           (\xintLength{#1#2}\if-#1-\xint_c_i\fi)<\XINT_Abs#5 %
       1\else
        \ifnum\numexpr\xint_c_e_iv/\xintLength{#3}<\XINT_Abs#5 %
       1\else
    0\fi\fi
      \expandafter\XINT_fpow_fork\else\expandafter\XINT_pow_bigint_i
    \fi
    #5\Z{#4}{#1#2}{#3}%
}%
\def\XINT_pow_bigint_i#1\Z#2#3#4%
{%
    \XINT_flpower_checkB_a#1.\XINTdigits.{#3/#4[#2]}{\XINTinfloatS[\XINTdigits]}%
}%
\def\XINT_expr_func_pow #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:two
    {\romannumeral`&&@\xintPow#3}}%
}%
```

28.10.1. &&, ||, //, /:, +, −, *, /, ^, **, 'and', 'or', 'xor', and 'mod'

```
\XINT_expr_defbin_a {expr}  {||} {6} {6} \calxOR
\catcode`& 12
\XINT_expr_defbin_a {expr}  {&&} {8} {8} \calxAND
\catcode`& 7
\XINT_expr_defbin_a {expr}  {xor}{6} {6} \calxXOR
\XINT_expr_defbin_a {expr}  {//} {14}{14}\xintDivFloor
\XINT_expr_defbin_a {expr}  {/:} {14}{14}\xintMod
\XINT_expr_defbin_a {expr}   +   {12}{12}\xintAdd
\XINT_expr_defbin_a {expr}   -   {12}{12}\xintSub
\XINT_expr_defbin_a {expr}   *   {14}{14}\xintMul
\XINT_expr_defbin_a {expr}   /   {14}{14}\xintDiv
\XINT_expr_defbin_a {expr}   ^   {18}{17}\xintPow
```

```
\catcode`& 12
   \expandafter\let\csname XINT_expr_op_and\expandafter\endcsname
                   \csname XINT_expr_op_&&\endcsname
   \expandafter\let\csname XINT_expr_op_or\expandafter\endcsname
                   \csname XINT_expr_op_||\endcsname
   \expandafter\let\csname XINT_expr_op_mod\expandafter\endcsname
                   \csname XINT_expr_op_/:\endcsname
\catcode`& 7
```

28.10.2. .., ..[, and].. for a..b and a..[b]..c syntax

a..[d]..b of \xinteval. Contrarily to a..b which is limited to small integers, this works with a, b, and d (big) fractions. It will produce a «nil» list, if a>b and d<0 or a<b and d>0.

\xintSeqA

```
\def\xintSeqA        {\expandafter\XINT_SeqA\romannumeral0\xintraw}%
\def\XINT_SeqA  #1]#2{\expandafter\XINT_SeqA_a\romannumeral0\xintraw {#2}#1]}%
\def\XINT_SeqA_a #1{\xint_UDzerominusfork
                                  #1-{z}%
                                  0#1{n}%
                                  0-{p}%
                    \krof #1}%
```

\xintSeqB:tl:x At 1.4, delayed expansion of start and step done here and not before, for matters of \xintdeffunc and «NEhooks».

The float variant at 1.4 is made identical to the exact variant. I.e. stepping is exact and comparison to the range limit too. But recall that a/b input will be converted to a float. To handle 1/3 step for example still better to use \xintexpr 1..1/3..10\relax for example inside the \xintfloateval.

```
\def\xintSeqB:tl:x  #1{\expandafter\XINT_SeqB:tl:x\romannumeral`&&@\xintSeqA#1}%
\def\XINT_SeqB:tl:x #1{\csname XINT_SeqB#1:tl:x\endcsname}%
\def\XINT_SeqBz:tl:x #1]#2]#3{{#2]}}%
\def\XINT_SeqBp:tl:x #1]#2]#3%
    {\expandafter\XINT_SeqBp:tl:x_a\romannumeral0\xintraw{#3}#2]#1]}%
\def\XINT_SeqBp:tl:x_a #1]#2]#3]%
{%
    \xintifCmp{#1]}{#2]}%
    {}{{#2]}}{{#2]}\expandafter\XINT_SeqBp:tl:x_b\romannumeral0\xintadd{#3]}{#2]}#1]#3]}%
}%
\def\XINT_SeqBp:tl:x_b #1]#2]#3]%
{%
    \xintifCmp{#1]}{#2]}%
    {{#1]}\expandafter\XINT_SeqBp:tl:x_b\romannumeral0\xintadd{#3]}{#1]}#2]#3]}{{#1]}}{}%
}%
\def\XINT_SeqBn:tl:x #1]#2]#3%
   {\expandafter\XINT_SeqBn:tl:x_a\romannumeral0\xintraw{#3}#2]#1]}%
\def\XINT_SeqBn:tl:x_a #1]#2]#3]%
{%
    \xintifCmp{#1]}{#2]}%
    {{#2]}\expandafter\XINT_SeqBn:tl:x_b\romannumeral0\xintadd{#3]}{#2]}#1]#3]}{{#2]}}{}%
}%
\def\XINT_SeqBn:tl:x_b #1]#2]#3]%
```

```
{%
    \xintifCmp{#1]}{#2]}%
    {}{{#1]}}{{#1]}\expandafter\XINT_SeqBn:tl:x_b\romannumeral0\xintadd{#3]}{#1]}#2]#3]}%
}%
\XINT_expr_defbin_b {expr}%
\XINT_expr_defbin_c {expr}   {..}\xintSeq:tl:x
\XINT_expr_defbin_c {expr}  {]..}\xintSeqB:tl:x
```

28.10.3. <, >, ==, <=, >=, != with Python-like chaining

```
\XINT_expr_defbin_d {expr}  <\xintLt
\XINT_expr_defbin_d {expr}  >\xintGt
\XINT_expr_defbin_d {expr}  {==}\xintEq
\XINT_expr_defbin_d {expr}  {<=}\xintLtorEq
\XINT_expr_defbin_d {expr}  {>=}\xintGtorEq
\XINT_expr_defbin_d {expr}  {!=}\xintNotEq
```

28.10.4. [...] as «oneple» constructor

```
\XINT_expr_defbin_e {expr}%
```

28.10.5. [...] brackets and : operator for NumPy-like slicing and item indexing syntax

```
\XINT_expr_defbin_f {expr}%
\XINT_expr_defbin_g {expr}%
```

28.11. Support for raw A/B[N]

See comments in calx.

```
\catcode0 11
\let\XINT_expr_precedence_&&@ \xint_c_xiv
\def\XINT_expr_op_&&@ #1#2]%
{%
    \expandafter\XINT_expr_put_op_first
    \expanded{{{\xintE#1{\xint_zapspaces #2 \xint_gobble_i}}}%
    \expandafter}\romannumeral`&&@\XINT_expr_getop
}%
\catcode2 11
\let\XINT_expr_op_&&B\XINT_flexpr_op_&&B
\catcode2 12
\catcode3 11
\let\XINT_expr_op_&&C \XINT_flexpr_op_&&C
\catcode3 12
\catcode0 12
```

28.12. ? as two-way and ?? as three-way «short-circuit» conditionals

```
\catcode`- 11
\def\XINT_expr_op_?  {\XINT_expr_op__? \XINT_expr_op_-xii}%
\catcode`- 12
```

28.13. ! as postfix factorial operator

```
\def\XINT_expr_op_! #1%
{%
    \expandafter\XINT_expr_put_op_first
    \expanded{{\romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\xintFac#1}}\expandafter}\romannumeral`&&@\XINT_expr_getop
}%
```

28.14. User defined variables: \xintdefvar

```
\def\xintdefvar     {\xintexprSafeCatcodes\xintdefvar_a}%
\def\xintdefvar_a#1={\XINT_expr_defvar\xintthebareeval{#1}}%
```

28.15. \qfrac() of \xinteval

The legacy qfrac() (\xintRaw: any xintfrac input, exact) for \xinteval only; the \xintfloateval and \xintiieval ones are calx.sty's (strict A/B) and are not changed.

```
\def\XINT_expr_onliteral_qfrac #1)%
    {\expandafter\XINT_expr_put_op_first\expanded{{{\xintRaw{#1}}}\expandafter
     }\romannumeral`&&@\XINT_expr_getop}%
```

28.16. Regular built-in functions

```
\let\XINT_expr_func_num\XINT_iiexpr_func_num
```

reduce() and preduce() (xintfrac's \xintIrr, \xintPIrr) of \xinteval; calx gives none for \xintfloateval: xintexpr.sty adds them, the same macros. reduce(): \xintIrrz, \xintIrr's output with [0] appended (the [0] follows the input of \XINT_irr_start, as the [N] of \xintPIrr). Not user level.

```
\def\xintIrrz {\romannumeral0\xintirrz }%
\def\xintirrz #1%
{%
    \expandafter\XINT_irr_start\romannumeral0\xintrawwithzeros {#1}\Z[0]%
}%
\def\XINT_expr_func_reduce #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\xintIrrz#3}}%
}%
\def\XINT_expr_func_preduce #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\xintPIrr#3}}%
}%
\let\XINT_flexpr_func_reduce\XINT_expr_func_reduce
\let\XINT_flexpr_func_preduce\XINT_expr_func_preduce
\def\XINT_expr_func_abs #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\xintAbs#3}}%
```

```
}%
\let\XINT_expr_func_sgn\XINT_iiexpr_func_sgn
\def\XINT_expr_func_frac #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\xintTFrac#3}}%
}%
\def\XINT_expr_func_floor #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\xintFloor#3}}%
}%
\def\XINT_expr_func_ceil #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\xintCeil#3}}%
}%
\def\XINT_expr_func_sqr #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\xintSqr#3}}%
}%
\let\XINT_expr_func_? \XINT_iiexpr_func_?
\let\XINT_expr_func_! \XINT_iiexpr_func_!
\let\XINT_expr_func_not \XINT_iiexpr_func_not
\def\XINT_expr_func_odd #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\xintOdd#3}}%
}%
\def\XINT_expr_func_even #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\xintEven#3}}%
}%
\def\XINT_expr_func_isint #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\xintIsInt#3}}%
}%
\def\XINT_expr_func_isone #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\xintIsOne#3}}%
```

```
}%
\def\XINT_expr_func_factorial #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:and:opt:direct
    \XINT:expr:f:one:and:opt #3,!\xintFac\XINTinFloatFac
    }}%
}%
\let\XINT_expr_func_sqrt\XINT_flexpr_func_sqrt
\def\XINT_expr_func_inv #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\xintInv#3}}%
}%
```

Modified at 2.0a1 (2026/11/01).

Eversince 1.4a and the support for negative arguments the functions round(x,n) and trunc(x,n) were half-broken in both \xinteval and \xintfloateval because the underlying \xintRound and \xintTrunc with a negative rounding argument produced scientific notation which is not acceptable in leaves which apply \detokenize hence a catcode 12 e breaking \xintfloateval printer.

```
\def\xintRoundToRaw#1{\romannumeral0\expandafter\xintroundtoraw\the\numexpr#1.}%
\def\xintroundtoraw#1.#2{\xintiround{#1}{#2}[-#1]}%
\def\XINT_expr_func_round #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:tacitzeroifone:direct
    \XINT:expr:f:tacitzeroifone #3,!\xintiRound\xintRoundToRaw
    }}%
}%
\def\xintTruncToRaw#1{\romannumeral0\expandafter\xinttrunctoraw\the\numexpr#1.}%
\def\xinttrunctoraw#1.#2{\xintitrunc{#1}{#2}[-#1]}%
\def\XINT_expr_func_trunc #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:tacitzeroifone:direct
    \XINT:expr:f:tacitzeroifone #3,!\xintiTrunc\xintTruncToRaw
    }}%
}%
\def\XINT_expr_func_float #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:and:opt:direct
    \XINT:expr:f:one:and:opt #3,!\XINTinFloatdigits\XINTinFloat
    }}%
}%
\def\XINT_expr_func_float_dgt #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\XINTinFloatdigits#3}}%
}%
\def\XINT_expr_func_sfloat #1#2#3%
```

```
{%
    \expandafter #1\expandafter #2\expandafter{\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:and:opt:direct
    \XINT:expr:f:one:and:opt #3,!\XINTinFloatSdigits\XINTinFloatS
    }}%
}%
\expandafter\def\csname XINT_expr_func_ilog10\endcsname #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:and:opt:direct
    \XINT:expr:f:one:and:opt #3,!\xintiLogTen\XINTFloatiLogTen
    }}%
}%
\def\XINT_expr_func_divmod #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{\romannumeral`&&@%
    \XINT:NEhook:f:one:from:two
    {\romannumeral`&&@\xintDivMod #3}}%
}%
\def\XINT_expr_func_mod #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{\romannumeral`&&@%
    \XINT:NEhook:f:one:from:two
    {\romannumeral`&&@\xintMod#3}}%
}%
\def\XINT_expr_func_binomial #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{\romannumeral`&&@%
    \XINT:NEhook:f:one:from:two
    {\romannumeral`&&@\xintBinomial #3}}%
}%
\def\XINT_expr_func_pfactorial #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{\romannumeral`&&@%
    \XINT:NEhook:f:one:from:two
    {\romannumeral`&&@\xintPFactorial #3}}%
}%
\let\XINT_expr_func_randrange\XINT_flexpr_func_randrange
```

gcd() and lcm() of \xinteval: xintfrac's \XINT_GCDof, \XINT_LCMof (fractions, raw A/B[0] output). calx gives none for \xintfloateval: xintexpr.sty adds them, the same macros (items converted to raw by xintfrac's input parser, \CALX_infr; output left raw as legacy).

```
\def\XINT_expr_func_gcd #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{\expandafter
    {\romannumeral`&&@\XINT:NEhook:f:from:delim:u\XINT_GCDof#3^}}%
}%
\def\XINT_expr_func_lcm #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{\expandafter
    {\romannumeral`&&@\XINT:NEhook:f:from:delim:u\XINT_LCMof#3^}}%
}%
\let\XINT_flexpr_func_gcd\XINT_expr_func_gcd
```

```
\let\XINT_flexpr_func_lcm\XINT_expr_func_lcm
\def\XINT_expr_func_max #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{\expandafter
    {\romannumeral`&&@\XINT:NEhook:f:from:delim:u\XINT_Maxof#3^}}%
}%
\def\XINT_expr_func_min #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{\expandafter
    {\romannumeral`&&@\XINT:NEhook:f:from:delim:u\XINT_Minof#3^}}%
}%
\expandafter
\def\csname XINT_expr_func_+\endcsname #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{\expandafter
    {\romannumeral`&&@\XINT:NEhook:f:from:delim:u\XINT_Sum#3^}}%
}%
\expandafter
\def\csname XINT_expr_func_*\endcsname #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{\expandafter
    {\romannumeral`&&@\XINT:NEhook:f:from:delim:u\XINT_Prd#3^}}%
}%
\let\XINT_expr_func_all\XINT_iiexpr_func_all
\let\XINT_expr_func_any\XINT_iiexpr_func_any
\let\XINT_expr_func_xor\XINT_iiexpr_func_xor
\let\XINT_expr_func_len\XINT_iiexpr_func_len
\let\XINT_expr_func_first\XINT_flexpr_func_first
\let\XINT_expr_func_last\XINT_flexpr_func_last
\let\XINT_expr_func_reversed\XINT_flexpr_func_reversed
\let\XINT_expr_func_if\XINT_flexpr_func_if
```

ifint() and ifone() of \xinteval: xintfrac's \xintifInt, \xintifOne, for its values (any xintfrac input, fractions); the calx ones of \xintiieval and \xintfloateval are for the native values.

```
\def\XINT_expr_func_ifint #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:branch{\romannumeral`&&@\xintifInt #3}}%
}%
\def\XINT_expr_func_ifone #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:branch{\romannumeral`&&@\xintifOne #3}}%
}%
\let\XINT_expr_func_ifsgn\XINT_flexpr_func_ifsgn
\let\XINT_expr_func_nuple\XINT_flexpr_func_nuple
\let\XINT_expr_func_unpack\XINT_flexpr_func_unpack
\let\XINT_expr_func_flat\XINT_flexpr_func_flat
\let\XINT_expr_func_zip\XINT_flexpr_func_zip
```

28.17. User declared functions

The mysterious \XINT_NewFunc is defined at bottom of this file. For comments see calx.

```
\def\xintdeffunc{\xintexprSafeCatcodes\xintdeffunc_a}%
\XINT_def_deffunc
   \xintdeffunc_a{expr}\XINT_NewFunc{expr}\xintdeffunc_b
\def\xintdefufunc{\xintexprSafeCatcodes\xintdefufunc_a}%
\XINT_def_defufunc
   \xintdefufunc_a{expr}\XINT_NewFunc{expr}\xintdefufunc_b\xintdefufunc_c
```

28.17.1. \xintunassignexprfunc

```
\XINT_def_unassignfunc{expr}%
```

28.17.2. \xintNewExpr, \xintNewIExpr, \xintNewBoolExpr, \xintNewFloatExpr, \xintNewIIExpr

1.2c modifications to accomodate \XINT_expr_deffunc_newexpr etc..
1.2f adds token \XINT_newexpr_clean to be able to have a different \XINT_newfunc_clean.
As \XINT_NewExpr always execute \XINT_expr_redefineprints since 1.3e whether with \xintNewExpr or \XINT_NewFunc, it has been moved from argument to hardcoded in replacement text.
NO MORE \XINT_expr_redefineprints at 1.4 ! This allows better support for \xinteval, \xinttheexpr as sub-entities inside an \xintNewExpr. And the «cleaning» will remove the new \XINTfstop , to maintain backwards compatibility with former behaviour that created macros expand to explicit digits and not an encapsulated result.
(obsolete:) The #2#3 in clean stands for \noexpand\XINTfstop.

Modified at 1.4n (2025/09/05). The #2#3 in previous paragraph is obsolete, the pattern used for \XINT_newexpr_clean was formerly #1>#2#3 (not optimal, but legacy), but is now simply #1\XINTfstop. This is simpler and allows compatibility with LuaMetaTeX whose \meaning output differs from the one of other engines.

```
\catcode`\~=12
\def\XINT_flexpr_getmacropar #1{%
\def\XINT_flexpr_getmacropar ##1%
{%
    \expanded{{{~xintRaw{#1##1}}}\expandafter}%
    \romannumeral`&&@\XINT_expr_getop
}%
}\expandafter\XINT_flexpr_getmacropar\string#%
\def\XINT_iiexpr_getmacropar #1{%
\def\XINT_iiexpr_getmacropar ##1%
{%
    \expanded{{{~xintiNum{#1##1}}}\expandafter}%
    \romannumeral`&&@\XINT_expr_getop
}%
}\expandafter\XINT_iiexpr_getmacropar\string#%
\catcode`\~=3
\def\xintNewFloatExpr{\begingroup
    \let\XINT_expr_getmacropar\XINT_flexpr_getmacropar
    \XINT_NewExpr_a\xint_firstofone\xintfloatexpr\XINT_newexpr_clean
}%
\def\xintNewIIExpr{\begingroup
    \let\XINT_expr_getmacropar\XINT_iiexpr_getmacropar
    \XINT_NewExpr_a\xint_firstofone\xintiiexpr\XINT_newexpr_clean
}%
\def\XINT_newexpr_clean #1\XINTfstop{\noexpand\expanded\noexpand\xintNEprinthook}%
\def\xintNEprinthook#1.#2{\expanded{\xint_noxpd{#1.}{#2}}}%
```

```
\def\xintNewExpr      {\XINT_NewExpr\xint_firstofone\xintexpr      \XINT_newexpr_clean}%
\def\xintNewIExpr     {\XINT_NewExpr\xint_firstofone\xintiexpr     \XINT_newexpr_clean}%
\def\xintNewBoolExpr {\XINT_NewExpr\xint_firstofone\xintboolexpr \XINT_newexpr_clean}%
\def\XINT_NewFunc     {\XINT_NewExpr\xint_gobble_i\xintthebareeval\XINT_newfunc_clean}%
\let\XINT_tmpa\undefined \let\XINT_tmpb\undefined \let\XINT_tmpc\undefined
\let\XINT_tmpd\undefined \let\XINT_tmpe\undefined
\XINTrestorecatcodesendinput%
```

29. Package calxcore implementation

Added at calx 2.0a1.

Contents

.1 Catcodes, ε-TEX and reload detection . . 512
.2 Package identification 513
.3 \xintNum, \xintiNum 513
.4 \xintiiSgn 514
.5 \xintiiOpp 515
.6 \xintiiCmp 515
.7 Exceptional conditions: signals, flags, handlers, traps 516
.7.1 \CALX_signalcondition 519
.7.2 \calxifflagraised, \calxresetflag, \calxresetflags 519
.7.3 \xintifNonFiniteSignalled 520
.7.4 \calxsetup 521
.8 Blocks of eight digits 522
.8.1 \XINT_cuz_small 522
.8.2 \XINT_zeroes_forviii 522
.8.3 \XINT_rsepbyviii 522
.8.4 \XINT_sepandrev_andcount 523
.8.5 \XINT_rev_nounsep 523
.8.6 \XINT_unrevbyviii 524
.8.7 Aliases for native callers 524
.9 Core arithmetic 524
.9.1 \CALX_add_a 524
.9.2 \CALX_mul_loop, \CALX_smallmul, \CALX_verysmallmul 526
.9.3 \CALX_smalldivx_a 527
.10 Native integer format 529
.10.1 \CALX_nfromraw 530
.10.2 \CALX_ntoraw 530
.10.3 \CALX_nskip 531
.11 Sign and zero tests of values 531
.11.1 \CALX_ifnz, \CALX_ifsgn 531
.12 Native addition and subtraction 533
.12.1 \CALX_nadd 533
.12.2 \CALX_nsub 536
.12.3 \CALX_nnega, \CALX_nnegb, \CALX_nnorm, \CALX_nlast, \CALX_ntake 537
.13 Native multiplication 538
.13.1 \CALX_nmul 538
.14 Native division 542
.14.1 \CALX_nudiv 542
.14.2 \CALX_nrevk 545
.14.3 \CALX_nilv 546
.14.4 \CALX_nds 546
.14.5 \CALX_ndab 549
.14.6 \CALX_ndrx 549
.14.7 \CALX_ndtw 551
.14.8 \CALX_ndts 553
.14.9 \CALX_ndth 556
.14.10 \CALX_ndfo 559
.15 Native signed division 562
.15.1 \CALX_ndivtrunc, \CALX_nmodtrunc, \CALX_ndivfloor, \CALX_nmod, \CALX_nquo, \CALX_nrem, \CALX_ndivround, \CALX_ndivision, \CALX_ndivmod 562
.15.2 \CALX_nsig 564
.15.3 \CALX_nq, \CALX_nqinc, \CALX_nrz, \CALX_nrs, \CALX_nrb, \CALX_npair 564
.16 Already expanded native arithmetic . . . 567
.17 Native power 567
.18 Native partial factorial and factorial . . . 568
.19 \XINT_split_fromleft 571

The eight-digit block utilities shared by calx and legacy xint arithmetic are defined here under their original XINT_ names. The native conversions and arithmetic kernels remain in the CALX_ namespace. The legacy xintcore.sty depends on calxcore.sty for the shared utilities.

29.1. Catcodes, ε-TEX and reload detection

The code of this subsection is copied over from the xint packages, with the package names changed.

```
\begingroup\catcode61\catcode48\catcode32=10\relax%
  \catcode13=5    % ^^M
  \endlinechar=13 %
  \catcode123=1   % {
  \catcode125=2   % }
  \catcode64=11   % @
  \catcode44=12   % ,
  \catcode46=12   % .
  \catcode58=12   % :
```

```
  \catcode94=7    % ^
  \def\empty{}\def\space{ }\newlinechar10
  \def\z{\endgroup}%
  \expandafter\let\expandafter\x\csname ver@calxcore.sty\endcsname
  \expandafter\let\expandafter\w\csname ver@xintkernel.sty\endcsname
  \expandafter\ifx\csname numexpr\endcsname\relax
    \expandafter\ifx\csname PackageWarningNoLine\endcsname\relax
      \immediate\write128{^^JPackage calxcore Warning:^^J%
                          \space\space\space\space
                          \numexpr not available, aborting input.^^J}%
    \else
      \PackageWarningNoLine{calxcore}{\numexpr not available, aborting input}%
    \fi
    \def\z{\endgroup\endinput}%
  \else
    \ifx\x\relax   % plain-TeX, first loading of calxcore.sty
      \ifx\w\relax % but xintkernel.sty not yet loaded.
         \def\z{\endgroup\input xintkernel.sty\relax}%
      \fi
    \else
      \ifx\x\empty % LaTeX, first loading,
      % variable is initialized, but \ProvidesPackage not yet seen
          \ifx\w\relax % xintkernel.sty not yet loaded.
            \def\z{\endgroup\RequirePackage{xintkernel}}%
          \fi
      \else
        \def\z{\endgroup\endinput}% xintkernel already loaded.
      \fi
    \fi
  \fi
\z%
\XINTsetupcatcodes% defined in xintkernel.sty
```

29.2. Package identification

```
\XINT_providespackage
\ProvidesPackage{calxcore}%
  [2026/11/26 v2.0a2 A calx component (JFB)]%
```

29.3. \xintNum, \xintiNum

For example \xintNum {----+-+++---+----000000000000003}
  Very old routine got completely rewritten at 1.2l.
  New code uses \numexpr governed expansion and fixes some issues of former version particularly regarding inputs of the \numexpr...\relax type without \the or \number prefix, and/or possibly no terminating \relax.
  \xintiNum{\numexpr 1}\foo in earlier versions caused premature expansion of \foo.
  \xintiNum{\the\numexpr 1} was ok, but a bit luckily so.
  Also, up to 1.2k inclusive, the macro fetched tokens eight by eight, and not nine by nine as is done now. I have no idea why.

```
\def\xintiNum {\romannumeral0\xintinum }%
\def\xintinum #1%
{%
```

```
    \expandafter\XINT_num_cleanup\the\numexpr\expandafter\XINT_num_loop
    \romannumeral`&&@#1\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\Z
}%
\def\xintNum {\romannumeral0\xintnum }%
\let\xintnum\xintinum
```

Attention \xintnum (hence \xintNum) gets redefined by xintfrac. Click on names to see the redefinition there.

```
\def\XINT_num #1%
{%
    \expandafter\XINT_num_cleanup\the\numexpr\XINT_num_loop
    #1\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\Z
}%
\def\XINT_num_loop #1#2#3#4#5#6#7#8#9%
{%
    \xint_gob_til_xint: #9\XINT_num_end\xint:
    #1#2#3#4#5#6#7#8#9%
    \ifnum \numexpr #1#2#3#4#5#6#7#8#9+\xint_c_ = \xint_c_
```

means that so far only signs encountered, (if syntax is legal) then possibly zeroes or a terminated or not terminated \numexpr evaluating to zero In that latter case a correct zero will be produced in the end.

```
      \expandafter\XINT_num_loop
    \else
```

non terminated \numexpr (with nine tokens total) are safe as after \fi, there is then \xint:

```
     \expandafter\relax
    \fi
}%
\def\XINT_num_end\xint:#1\xint:{#1+\xint_c_\xint:}% empty input ok
\def\XINT_num_cleanup #1\xint:#2\Z { #1}%
```

29.4. \xintiiSgn

Moved to calxcore for STUPID REASONS. Too painful to comment. 1.21 made \xintiiSgn robust against non terminated input.

1.2o deprecates here \xintSgn (it requires xintfrac.sty).

```
\def\xintiiSgn {\romannumeral0\xintiisgn }%
\def\xintiisgn #1%
{%
    \expandafter\XINT_sgn \romannumeral`&&@#1\xint:
}%
\def\XINT_sgn #1#2\xint:
{%
    \xint_UDzerominusfork
      #1-{ 0}%
      0#1{-1}%
       0-{ 1}%
    \krof
}%
\def\XINT_Sgn #1#2\xint:
{%
    \xint_UDzerominusfork
      #1-{0}%
```

```
      0#1{-1}%
       0-{1}%
    \krof
}%
\def\XINT_cntSgn #1#2\xint:
{%
    \xint_UDzerominusfork
      #1-\xint_c_
      0#1\xint_c_mone
       0-\xint_c_i
    \krof
}%
```

29.5. \xintiiOpp

Moved to calxcore for STUPID REASONS. Too painful to comment. Attention, \xintiiOpp non robust against non terminated inputs. Reason is I don't want to have to grab a delimiter at the end, as everything happens "upfront".

```
\def\xintiiOpp {\romannumeral0\xintiiopp }%
\def\xintiiopp #1%
{%
    \expandafter\XINT_opp \romannumeral`&&@#1%
}%
\def\XINT_Opp #1{\romannumeral0\XINT_opp #1}%
\def\XINT_opp #1%
{%
    \xint_UDzerominusfork
      #1-{ 0}%      zero
      0#1{ }%     negative
       0-{ -#1}%  positive
    \krof
}%
```

29.6. \xintiiCmp

The reason for it to be here is a completely silly situation which has cost the author perhaps 6 hours due to incompetent assistant preferring to work in secret and keep major decisions to itself. It may be provisory but will not be modified for 2.0a1.

Modified at 1.4m (2022/06/10). Now uses the \xintstrcmp engine primitive.

```
\def\xintiiCmp   {\romannumeral0\xintiicmp }%
\def\xintiicmp #1{\expandafter\XINT_iicmp\romannumeral`&&@#1\xint:}%
\def\XINT_iicmp #1#2\xint:#3%
{%
    \expandafter\XINT_cmp_nfork\expandafter #1\romannumeral`&&@#3\xint:#2\xint:
}%
\def\XINT_cmp_nfork #1#2%
{%
    \xint_UDzerofork
      #1\XINT_cmp_firstiszero
      #2\XINT_cmp_secondiszero
       0{}%
    \krof
```

```
    \xint_UDsignsfork
          #1#2\XINT_cmp_minusminus
           #1-\XINT_cmp_minusplus
           #2-\XINT_cmp_plusminus
            --\XINT_cmp_plusplus
    \krof #1#2%
}%
\def\XINT_cmp_firstiszero  #1\krof 0#2#3\xint:#4\xint:
{%
    \xint_UDzerominusfork
      #2-{ 0}%
      0#2{ 1}%
       0-{ -1}%
    \krof
}%
\def\XINT_cmp_secondiszero #1\krof #20#3\xint:#4\xint:
{%
    \xint_UDzerominusfork
      #2-{ 0}%
      0#2{ -1}%
       0-{ 1}%
    \krof
}%
\def\XINT_cmp_plusminus    #1\xint:#2\xint:{ 1}%
\def\XINT_cmp_minusplus    #1\xint:#2\xint:{ -1}%
\def\XINT_cmp_minusminus
    --{\expandafter\XINT_opp\romannumeral0\XINT_cmp_plusplus {}{}}%
```

The \romannumeral0 trigger induces some complications here to terminate nicely without grabbing too many tokens in the stream or deteriorating expansion quality of the non-equal-length branches. \expanded simplifies things.

```
\def\XINT_cmp_plusplus #1#2#3\xint:#4\xint:{\expanded{ %
    \ifcase\expandafter\XINT_cntSgn\the\numexpr\xintLength{#1#4}-\xintLength{#2#3}\xint:
        \xintstrcmp{#1#4}{#2#3}\or1\else-1\fi
    }%
}%
```

29.7. Exceptional conditions: signals, flags, handlers, traps

The code of this section comes from the unfinished signal code of xintcore.sty (dating back to 1.2l), renamed, kept in the core module, and completed. As per the Mike Cowlishaw/IBM's General Decimal Arithmetic Specification

http://speleotrove.com/decimal/decarith.html

and the Python3 implementation in its Decimal module.

Clamped, ConversionSyntax, DivisionByZero, DivisionImpossible, DivisionUndefined, Inexact, InsufficientStorage, InvalidContext, InvalidOperation, Overflow, Inexact, Rounded, Subnormal, Underflow.

X3.274 rajoute LostDigits

Python rajoute FloatOperation (et n'inclut pas InsufficientStorage)

quote de decarith.pdf: The Clamped, Inexact, Rounded, and Subnormal conditions can coincide with each other or with other conditions. In these cases then any trap enabled for another condition takes precedence over (is handled before) all of these, any Subnormal trap takes precedence

over Inexact, any Inexact trap takes precedence over Rounded, and any Rounded trap takes precedence over Clamped.

This was initiated at 1.2l (2017/07/26), but the possibility to customize handlers was never finished nor documented for users. Completely redone for 2.0a1 (2026/11/01).

Model: the decimal module of Python (as for the xintcore code), with the IEEE 754 default exception handling as default configuration: a condition raises a flag and is handled, the handler returns a default result, and NO TeX error is emitted. Emitting an (expandable) error is an opt-in, per signal, via \calxsetup.

Vocabulary (that of Python's decimal module):

- a CONDITION is what a calx macro detects and reports;
- each condition belongs to a SIGNAL; for most conditions, condition and signal are the same; five conditions belong to signal InvalidOperation:
  - ConversionSyntax (input which is not a number)
  - DivisionImpossible (integer division result does not fit)
  - DivisionUndefined (0/0)
  - InvalidContext (invalid configuration)
  - ControversialOperation (argument too large for a periodic function)
- there are eight signals: Clamped, DivisionByZero, Inexact, InvalidOperation, Overflow, Rounded, Subnormal, Underflow;
- each SIGNAL has a FLAG (raised or not) and a TRAP (set or not);
- each CONDITION has a HANDLER.

What happens when calx code signals a condition C of signal S, via

\CALX_signalcondition{C}{<message>}{<handler argument>}{<trap value>}

(this is \CALX_signalcondition_a below, Python's Context._raise_error):

- 1. if S is "ignored" (control sequence \CALX_S_isignoredflag defined; Python's Context._ignored_flags), the handler of S is applied to <handler argument>, and that is all: no flag is raised;
- 2. otherwise the flag of S is raised; then
- 3. if S is not trapped (the default for all signals), the handler of C is applied to <handler argument>, and its result is the result;
- 4. if S is trapped, an expandable error is emitted with <message>, and the result is <trap value>, the value the calling macro would return without the signal machinery (e.g. what xint returns: 0 on division by zero).

No calx code sets the "ignored" status at this stage; it is kept from the xintcore code (and Python) for internal computations which must not leave traces in the flags. It is set by \let\CALX_S_isignoredflag\relax (via \csname) and cleared by \let to \XINT_undefined.

Default handlers (Python's decimal: the "handle" method of each condition class; a class without "handle" returns None in Python, meaning the caller keeps the result it has computed, which here is the handler argument):

condition	handler argument	result
InvalidOperation	(ignored)	I? (NaN)
ConversionSyntax	(ignored)	I?
DivisionImpossible	(ignored)	I?
DivisionUndefined	(ignored)	I?
InvalidContext	(ignored)	I?
ControversialOperation	(ignored)	I?
DivisionByZero	sign + or -	I+ or I- (+-Infinity)
Overflow	sign + or -	I+ or I-
Underflow	computed result	the argument
Clamped	computed result	the argument
Inexact	computed result	the argument

```
   Rounded            computed result            the argument
   Subnormal          computed result            the argument
```

I, ?, + and - in results are catcode 12 characters, the special value markers of the calx internal format (I+ is +Infinity, I- is -Infinity, I? is NaN). The sign handed to the DivisionByZero and Overflow handlers must be a catcode 12 + or -. Printing the special values as NaN, Infinity, -Infinity is not done here: it is the task of the output macros of the arithmetic modules.

Overflow: Python returns +-Infinity or the largest finite number, depending on the rounding mode. calx rounds half away from zero, for which Python (ROUND_HALF_UP) returns +-Infinity.

FLAGS. A flag of signal S is raised when \CALX_SFlag_ON is defined. It is raised EXPANDABLY by

```
\expandafter\xint_gobble_i\csname CALX_SFlag_ON\endcsname
```

which uses the fact that \csname...\endcsname, if the name is undefined, \let's it to \relax as a side effect. This assignment is ALWAYS LOCAL to the current group (checked under etex: even with \globaldefs=1). Hence:

- a flag raised by a computation done inside a group (\begingroup or braces, an \hbox, an alignment cell, ...) is down again after the group ends;
- a flag is raised only when the computation is actually expanded: inside an \edef, an \expanded, a \message, a \csname, or f-expansion; storing an unexpanded computation in a macro via \def raises nothing.

A user wanting a flag to survive a group tests it before the group ends and makes the information global by his own means.

Testing (expandable) and lowering (NOT expandable):

```
\calxifflagraised{S}{<yes>}{<no>}
\calxresetflag{S}
\calxresetflags   (the eight flags)
```

Lowering obeys \ifxintglobaldefs (via \XINT_global): it is global if \xintglobaldefstrue, local otherwise.

CONFIGURATION, \calxsetup. Same syntax and method as \bnumsetup of bnumexpr: a comma separated list of key=value; spaces around keys are removed; a final comma does no harm; each key performs a \let; assignments are global if \xintglobaldefstrue, local otherwise (\XINT_global); with \xintverbosetrue each assignment is reported on the terminal and in the log (\xintMessage).

Keys:

```
<S>.trap    = true   (or false)          for each of the eight signals
<C>.handler = \macro                     for each of the thirteen conditions
```

e.g.

```
\calxsetup{DivisionByZero.trap = true, InvalidOperation.trap = true}
```

gives back the behaviour of xint (expandable error, then xint's default result) for these two signals.

A handler is a macro with ONE argument, the handler argument; it must expand to the result (in the contexts where calx macros use it, it is expanded by f-expansion). For example a user preferring that conversion errors give zero, in the calx internal format of zero, would \let the ConversionSyntax handler to a macro expanding to that. The value of <C>.handler must be a single control sequence (the \let receives only that).

An unknown key, or a value other than true and false for a .trap key, is reported by a (non expandable) TeX error, \calxsetup being itself not expandable.

```
\catcode`- 11
\def\CALX_ConversionSyntax-signal  {{InvalidOperation}}%
\let\CALX_DivisionImpossible-signal\CALX_ConversionSyntax-signal
\let\CALX_DivisionUndefined-signal \CALX_ConversionSyntax-signal
\let\CALX_InvalidContext-signal    \CALX_ConversionSyntax-signal
\let\CALX_ControversialOperation-signal\CALX_ConversionSyntax-signal
\catcode`- 12
```

29.7.1. \CALX_signalcondition

Syntax (f-expandable):
  \CALX_signalcondition{C}{M}{H}{V} -> <result>
C a signal or condition name (InvalidOperation, DivisionByZero, ...), M the error message, H the handler argument, V the value returned if the signal is trapped; <result>: the handler's value (or V).
  \CALX_signalcondition: identical to \XINT_signalcondition (xintcore.sty of xint 1.4p), renamed. The first \romannumeral0 step turns #1 into {<signal>}{<condition>}, then \CALX_signalcondition_a receives
    #1 = signal, #2 = condition, #3 = error message,
    #4 = handler argument, #5 = value returned if the signal is trapped.
Note that, as in Python, the "ignored" branch applies the handler of the SIGNAL, the untrapped branch the handler of the CONDITION.
  The expansion is complete (handler included) under f-expansion. The handlers defined here expand in one step to their result.

```
\def\CALX_signalcondition #1{\expandafter\CALX_signalcondition_a
    \romannumeral0\ifcsname CALX_#1-signal\endcsname
                    \xint_dothis{\csname CALX_#1-signal\endcsname}%
                  \fi\xint_orthat{{#1}}{#1}}%
\def\CALX_signalcondition_a %
    #1#2#3#4#5{% copied over from Python Decimal module
\ifcsname CALX_#1_isignoredflag\endcsname
       \xint_dothis{\csname CALX_#1.handler\endcsname {#4}}%
    \fi
    \expandafter\xint_gobble_i\csname CALX_#1Flag_ON\endcsname
    \unless\ifcsname CALX_#1_istrapped\endcsname
       \xint_dothis{\csname CALX_#2.handler\endcsname {#4}}%
    \fi
    \xint_orthat{%
       % the flag raised is named after the signal #1, but we show condition
       % #2
       \XINT_expandableerror{#3}%
       \xint_firstofone{#5}%
    }%
}%
\let\XINT_signalcondition\CALX_signalcondition
```

Flags. \CALX_RaiseFlag is xintcore's \XINT_RaiseFlag, for direct use by calx code (raising a flag without going through a handler). The public test and reset macros are xintcore's \XINT_ifFlagRaised, \XINT_resetFlag, \XINT_resetFlags made public; the reset obeys \XINT_global, and resets all eight flags (the xintcore version listed four).

```
\def\CALX_RaiseFlag #1{\expandafter\xint_gobble_i\csname CALX_#1Flag_ON%
    \endcsname}%
```

29.7.2. \calxifflagraised, \calxresetflag, \calxresetflags

```
\def\calxifflagraised #1{%
    \ifcsname CALX_#1Flag_ON\endcsname
        \expandafter\xint_firstoftwo
    \else
        \expandafter\xint_secondoftwo
    \fi}%
```

29.7.3. \xintifNonFiniteSignalled

True if InvalidOperation, DivisionByZero or Overflow was signalled in the current group since the last corresponding flag reset. The flags describe conditions, not the result: a handler can return a finite value. Literal NaN/Infinity values alone do not raise flags.

```
\def\xintifNonFiniteSignalled{%
    \calxifflagraised{InvalidOperation}{\xint_firstoftwo}{%
      \calxifflagraised{DivisionByZero}{\xint_firstoftwo}{%
        \calxifflagraised{Overflow}{\xint_firstoftwo}{\xint_secondoftwo}%
      }%
    }%
}%
\def\calxresetflag #1%
    {\XINT_global\expandafter\let\csname CALX_#1Flag_ON\endcsname%
        \XINT_undefined}%
\def\calxresetflags {%
    \calxresetflag{Clamped}%
    \calxresetflag{DivisionByZero}%
    \calxresetflag{Inexact}%
    \calxresetflag{InvalidOperation}%
    \calxresetflag{Overflow}%
    \calxresetflag{Rounded}%
    \calxresetflag{Subnormal}%
    \calxresetflag{Underflow}%
}%
```

Default handlers. The xintcore table had _NaN and _SignedInfinity(#1) as placeholders; here they are the special value markers, catcode 12, produced by \detokenize. Underflow, which the xintcore table sent to _NaN, is the identity as in Python (Underflow is always accompanied by Inexact and Subnormal, and the result is the rounded one). DivisionByZero and Overflow both return signed infinity by default (round half up).

```
\catcode`. 11
\let\CALX_Clamped.handler\xint_firstofone
\edef\CALX_InvalidOperation.handler#1{\detokenize{I?}}%
\let\CALX_ConversionSyntax.handler  \CALX_InvalidOperation.handler
\let\CALX_DivisionImpossible.handler\CALX_InvalidOperation.handler
\let\CALX_DivisionUndefined.handler \CALX_InvalidOperation.handler
\let\CALX_InvalidContext.handler    \CALX_InvalidOperation.handler
\let\CALX_ControversialOperation.handler\CALX_InvalidOperation.handler
\edef\CALX_Overflow.handler#1{\detokenize{I}#1}%
\let\CALX_DivisionByZero.handler\CALX_Overflow.handler
\let\CALX_Inexact.handler\xint_firstofone
\let\CALX_Rounded.handler\xint_firstofone
\let\CALX_Subnormal.handler\xint_firstofone
\let\CALX_Underflow.handler\xint_firstofone
\catcode`. 12
```

Traps: none is set at load time. (The xintcore code set all eight at load time: \csname XINT_<name>_istrapped\endcsname.)

\calxsetup: the parsing loop is \bnumsetup's (bnumexpr.dtx), with the key dispatched through \csname CALX_setup_<key>\endcsname, one such macro per valid key, defined below. The value is handed over complete (not only its first token as in \bnumsetup) because .trap keys take a word. Each CALX_setup_<key> macro receives {<key as typed>}{<value>}.

```
\catcode`! 3
```

29.7.4. \calxsetup

```
\def\calxsetup #1{\CALX_parsekeys #1,=!,}%
\def\CALX_parsekeys #1=#2,%
  {%
    \ifx!#2\expandafter\CALX_parsedone\fi
    \ifcsname CALX_setup_\xint_zapspaces #1 \xint_gobble_i\endcsname
      \xint_dothis{\csname CALX_setup_\xint_zapspaces #1 \xint_gobble_i%
          \endcsname}%
    \fi
    \xint_orthat{\CALX_setup_badkey}{#1}{#2}%
    \CALX_parsekeys
  }%
\def\CALX_parsedone #1\CALX_parsekeys {}%
\catcode`! 12
\def\CALX_settrap #1#2#3{%
    \ifnum\xintstrcmp{\xint_zapspaces #3 \xint_gobble_i}{true}=\xint_c_
      \XINT_global\expandafter\let\csname CALX_#1_istrapped\endcsname\relax
      \CALX_setup_info{trap set for signal #1}%
    \else
    \ifnum\xintstrcmp{\xint_zapspaces #3 \xint_gobble_i}{false}=\xint_c_
      \XINT_global\expandafter\let\csname CALX_#1_istrapped\endcsname%
          \XINT_undefined
      \CALX_setup_info{trap removed for signal #1}%
    \else
      \CALX_setup_error{value \detokenize{#3} of key \detokenize{#2}
                        is neither true nor false}%
    \fi\fi
}%
\def\CALX_sethandler #1#2#3{%
    \XINT_global\expandafter\let\csname CALX_#1.handler\endcsname=#3%
    \CALX_setup_info{\string#3 assigned as handler of condition #1}%
}%
\def\CALX_setup_badkey #1#2{\CALX_setup_error{unknown key \detokenize{#1}}}%
\def\CALX_setup_info #1{%
    \ifxintverbose
      \xintMessage{calxcore}{Info}{\ifxintglobaldefs (global) \fi #1}%
    \fi
}%
\def\CALX_setup_error #1{%
    \xintMessage{calxcore}{Error}{\string\calxsetup\space #1}%
    \errhelp{See the comments of calxcore.sty on \string\calxsetup.}%
    \errmessage{calxcore: invalid input to \string\calxsetup}%
}%
\def\CALX_tmpa #1{%
    \expandafter\def\csname CALX_setup_#1.trap\endcsname{\CALX_settrap{#1}}%
}%
\CALX_tmpa{Clamped}\CALX_tmpa{DivisionByZero}\CALX_tmpa{Inexact}%
\CALX_tmpa{InvalidOperation}\CALX_tmpa{Overflow}\CALX_tmpa{Rounded}%
\CALX_tmpa{Subnormal}\CALX_tmpa{Underflow}%
\def\CALX_tmpa #1{%
    \expandafter\def\csname CALX_setup_#1.handler\endcsname{%
        \CALX_sethandler{#1}}%
```

```
}%
\CALX_tmpa{Clamped}\CALX_tmpa{DivisionByZero}\CALX_tmpa{Inexact}%
\CALX_tmpa{InvalidOperation}\CALX_tmpa{Overflow}\CALX_tmpa{Rounded}%
\CALX_tmpa{Subnormal}\CALX_tmpa{Underflow}%
\CALX_tmpa{ConversionSyntax}\CALX_tmpa{DivisionImpossible}%
\CALX_tmpa{DivisionUndefined}\CALX_tmpa{InvalidContext}%
\CALX_tmpa{ControversialOperation}%
\let\CALX_tmpa\XINT_undefined
```

29.8. Blocks of eight digits

29.8.1. \XINT_cuz_small

\XINT_cuz_small removes leading zeroes from the first eight digits. Expands following \romannumeral0. At least one digit is produced.

```
\def\XINT_cuz_small#1{%
\def\XINT_cuz_small ##1##2##3##4##5##6##7##8%
{%
    \expandafter#1\the\numexpr ##1##2##3##4##5##6##7##8\relax
}}\XINT_cuz_small{ }%
```

29.8.2. \XINT_zeroes_forviii

\romannumeral0\XINT_zeroes_forviii #1\R\R\R\R\R\R\R\R{10}0000001\W produces a string of k 0's such that k+length(#1) is smallest bigger multiple of eight.

```
\def\XINT_zeroes_forviii #1#2#3#4#5#6#7#8%
{%
    \xint_gob_til_R #8\XINT_zeroes_forviii_end\R\XINT_zeroes_forviii
}%
\def\XINT_zeroes_forviii_end#1{%
\def\XINT_zeroes_forviii_end\R\XINT_zeroes_forviii ##1##2##3##4##5##6##7##8##9\W
{%
    \expandafter#1\xint_gob_til_one ##2##3##4##5##6##7##8%
}}\XINT_zeroes_forviii_end{ }%
```

29.8.3. \XINT_rsepbyviii

This is used as

```
\the\numexpr1\XINT_rsepbyviii <8Ndigits>%
             \XINT_rsepbyviii_end_A 2345678%
             \XINT_rsepbyviii_end_B 2345678\relax UV%
```

and will produce

```
1<8digits>!1<8digits>\xint:1<8digits>!...
```

where the original digits are organized by eight, and the order inside successive pairs of blocks separated by \xint: has been reversed. Output ends either in 1<8d>!1<8d>\xint:1U\xint: (even) or 1<8d>!1<8d>\xint:1V!1<8d>\xint: (odd)

The U an V should be \numexpr1 stoppers (or will expand and be ended by !). This macro is currently (1.2..1.2l) exclusively used in combination with \XINT_sepandrev_andcount or \XINT_sepandrev.

```
\def\XINT_rsepbyviii #1#2#3#4#5#6#7#8%
{%
    \XINT_rsepbyviii_b {#1#2#3#4#5#6#7#8}%
}%
\def\XINT_rsepbyviii_b #1#2#3#4#5#6#7#8#9%
```

```
{%
    #2#3#4#5#6#7#8#9\expandafter!\the\numexpr
    1#1\expandafter\xint:\the\numexpr 1\XINT_rsepbyviii
}%
\def\XINT_rsepbyviii_end_B #1\relax #2#3{#2\xint:}%
\def\XINT_rsepbyviii_end_A #11#2\expandafter #3\relax #4#5{#5!1#2\xint:}%
```

29.8.4. \XINT_sepandrev_andcount

This is used typically as

```
   \romannumeral0\XINT_sepandrev_andcount <8Ndigits>%
                 \XINT_rsepbyviii_end_A 2345678%
                 \XINT_rsepbyviii_end_B 2345678\relax\xint_c_ii\xint_c_i
                 \R\xint:\xint_c_xii \R\xint:\xint_c_x  \R\xint:\xint_c_viii \R\xint:\xint_c_
↪ vi
                 \R\xint:\xint_c_iv  \R\xint:\xint_c_ii \R\xint:\xint_c_\W
```

and will produce

```
   <length>.1<8digits>!1<8digits>!1<8digits>!...
```

where the blocks have been globally reversed and <length> is the number of blocks.

```
\def\XINT_sepandrev_andcount
{%
    \expandafter\XINT_sepandrev_andcount_a\the\numexpr 1\XINT_rsepbyviii
}%
\def\XINT_sepandrev_andcount_a {\XINT_sepandrev_andcount_b 0!{}}%
\def\XINT_sepandrev_andcount_b #1!#2#3\xint:#4\xint:#5\xint:#6\xint:#7\xint:#8\xint:#9\xint:%
{%
    \xint_gob_til_R #9\XINT_sepandrev_andcount_end\R
    \expandafter\XINT_sepandrev_andcount_b \the\numexpr #1+\xint_c_i!%
    {#9!#8!#7!#6!#5!#4!#3!#2}%
}%
\def\XINT_sepandrev_andcount_end\R
    \expandafter\XINT_sepandrev_andcount_b\the\numexpr #1+\xint_c_i!#2#3#4\W
{\expandafter\XINT_sepandrev_andcount_done\the\numexpr #3+\xint_c_xiv*#1!#2}%
\def\XINT_sepandrev_andcount_done#1{%
\def\XINT_sepandrev_andcount_done##1!##21##3!{\expandafter#1\the\numexpr##1-##3\xint:}%
}\XINT_sepandrev_andcount_done{ }%
```

29.8.5. \XINT_rev_nounsep

This is used as

```
   \romannumeral0\XINT_rev_nounsep {}<blocks 1<8d>!>\R!\R!\R!\R!\R!\R!\R!\W
```

It reverses the blocks, keeping the 1's and ! separators. Used multiple times in the division algorithm. The inserted {} here is not optional.

```
\def\XINT_rev_nounsep #1#2!#3!#4!#5!#6!#7!#8!#9!%
{%
    \xint_gob_til_R #9\XINT_rev_nounsep_end\R
    \XINT_rev_nounsep {#9!#8!#7!#6!#5!#4!#3!#2!#1}%
}%
\def\XINT_rev_nounsep_end\R\XINT_rev_nounsep #1#2\W {\XINT_rev_nounsep_done #1}%
\def\XINT_rev_nounsep_done #11{ 1}%
```

29.8.6. \XINT_unrevbyviii

Used as \romannumeral0\XINT_unrevbyviii 1<8d>!....1<8d>! terminated by
1;!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W
The \romannumeral in unrevbyviii_a is for special effects (expand some token which was put as 1<token>! at the end of the original blocks). This mechanism is used by 1.2 subtraction (still true for 1.2l).

```
\def\XINT_unrevbyviii #11#2!1#3!1#4!1#5!1#6!1#7!1#8!1#9!%
{%
    \xint_gob_til_R #9\XINT_unrevbyviii_a\R
    \XINT_unrevbyviii {#9#8#7#6#5#4#3#2#1}%
}%
\def\XINT_unrevbyviii_a#1{%
\def\XINT_unrevbyviii_a\R\XINT_unrevbyviii ##1##2\W
    {\expandafter#1\romannumeral`&&@\xint_gob_til_sc ##1}%
}\XINT_unrevbyviii_a{ }%
```

29.8.7. Aliases for native callers

The native arithmetic section uses I and N with catcode 12. Its callers use CALX aliases of the common block utilities, whose implementations remain under the XINT names for the legacy packages.

```
\let\CALX_cuz_small\XINT_cuz_small
\let\CALX_zeroes_forviii\XINT_zeroes_forviii
\let\CALX_rsepbyviii\XINT_rsepbyviii
\let\CALX_rsepbyviii_end_A\XINT_rsepbyviii_end_A
\let\CALX_rsepbyviii_end_B\XINT_rsepbyviii_end_B
\let\CALX_sepandrev_andcount\XINT_sepandrev_andcount
\let\CALX_rev_nounsep\XINT_rev_nounsep
\let\CALX_unrevbyviii\XINT_unrevbyviii
```

29.9. Core arithmetic

29.9.1. \CALX_add_a

From this subsection of xintcore (\xintiiAdd) calx copies only the macros used by the native layer: \CALX_add_a, \CALX_add_b, \CALX_add_bi, \CALX_add_c, \CALX_add_d, \CALX_add_di, \CALX_add_e, \CALX_add_f, \CALX_add_fi, \CALX_add_g, \CALX_add_h, \CALX_add_hi, \CALX_add_i, \CALX_add_k, \CALX_add_ke, \CALX_add_kf, \CALX_add_l, \CALX_add_lf, \CALX_add_m, \CALX_add_n, \CALX_add_o. The other macros are not copied.

```
\def\CALX_add_a #1!#2!#3!#4!#5\W
                #6!#7!#8!#9!%
{%
    \CALX_add_b
        #1!#6!#2!#7!#3!#8!#4!#9!%
        #5\W
}%
\def\CALX_add_b #11#2#3!#4!%
{%
    \xint_gob_til_sc #2\CALX_add_bi ;%
    \expandafter\CALX_add_c\the\numexpr#1+1#2#3+#4-\xint_c_ii\xint:%
}%
\def\CALX_add_bi;\expandafter\CALX_add_c
    \the\numexpr#1+#2+#3-\xint_c_ii\xint:#4!#5!#6!#7!#8!#9!\W
```

```
{%
    \CALX_add_k #1#3!#5!#7!#9!%
}%
\def\CALX_add_c #1#2\xint:%
{%
    1#2\expandafter!\the\numexpr\CALX_add_d #1%
}%
\def\CALX_add_d #11#2#3!#4!%
{%
    \xint_gob_til_sc #2\CALX_add_di ;%
    \expandafter\CALX_add_e\the\numexpr#1+1#2#3+#4-\xint_c_ii\xint:%
}%
\def\CALX_add_di;\expandafter\CALX_add_e
    \the\numexpr#1+#2+#3-\xint_c_ii\xint:#4!#5!#6!#7!#8\W
{%
    \CALX_add_k #1#3!#5!#7!%
}%
\def\CALX_add_e #1#2\xint:%
{%
    1#2\expandafter!\the\numexpr\CALX_add_f #1%
}%
\def\CALX_add_f #11#2#3!#4!%
{%
    \xint_gob_til_sc #2\CALX_add_fi ;%
    \expandafter\CALX_add_g\the\numexpr#1+1#2#3+#4-\xint_c_ii\xint:%
}%
\def\CALX_add_fi;\expandafter\CALX_add_g
    \the\numexpr#1+#2+#3-\xint_c_ii\xint:#4!#5!#6\W
{%
    \CALX_add_k #1#3!#5!%
}%
\def\CALX_add_g #1#2\xint:%
{%
    1#2\expandafter!\the\numexpr\CALX_add_h #1%
}%
\def\CALX_add_h #11#2#3!#4!%
{%
    \xint_gob_til_sc #2\CALX_add_hi ;%
    \expandafter\CALX_add_i\the\numexpr#1+1#2#3+#4-\xint_c_ii\xint:%
}%
\def\CALX_add_hi;%
    \expandafter\CALX_add_i\the\numexpr#1+#2+#3-\xint_c_ii\xint:#4\W
{%
    \CALX_add_k #1#3!%
}%
\def\CALX_add_i #1#2\xint:%
{%
    1#2\expandafter!\the\numexpr\CALX_add_a #1%
}%
\def\CALX_add_k #1{\if #12\expandafter\CALX_add_ke\else\expandafter%
    \CALX_add_l \fi}%
\def\CALX_add_ke #11;#2\W {\CALX_add_kf #11;!}%
```

```
\def\CALX_add_kf 1{1\relax }%
\def\CALX_add_l 1#1{\xint_gob_til_sc #1\CALX_add_lf ;\CALX_add_m 1#1}%
\def\CALX_add_lf #1\W {1\relax 00000001!1;!}%
\def\CALX_add_m #1!{\expandafter\CALX_add_n\the\numexpr\xint_c_i+#1\xint:}%
\def\CALX_add_n #1#2\xint:{1#2\expandafter!\the\numexpr\CALX_add_o #1}%
\def\CALX_add_o #1{\if #12\expandafter\CALX_add_l\else\expandafter%
    \CALX_add_ke \fi}%
```

29.9.2. \CALX_mul_loop, \CALX_smallmul, \CALX_verysmallmul

From this subsection of xintcore (\xintiiMul) calx copies only the macros used by the native layer: \CALX_mul_loop, \CALX_mul_a, \CALX_mul_b, \CALX_mul_e, \CALX_minimulwc_a, \CALX_minimulwc_b, \CALX_minimulwc_c, \CALX_smallmul, \CALX_smallmul_a, \CALX_smallmul_e, \CALX_smallmul_f, \CALX_verysmallmul, \CALX_verysmallmul_e, \CALX_verysmallmul_f, \CALX_verysmallmul_a, \CALX_verysmallmul_bj, \CALX_verysmallmul_cj, \CALX_verysmallmul_bi, \CALX_minimul_a, \CALX_minimul_b, \CALX_minimul_c. The other macros are not copied.

```
\def\CALX_mul_loop #1\W #2\W 1#3!%
{%
    \xint_gob_til_sc #3\CALX_mul_e ;%
    \expandafter\CALX_mul_a\the\numexpr \CALX_smallmul 1#3!#2\W
    #1\W #2\W
}%
\def\CALX_mul_a #1\W #2\W
{%
    \expandafter\CALX_mul_b\the\numexpr
    \CALX_add_a \xint_c_ii #21;!1;!1;!\W #11;!1;!1;!\W\W
}%
\def\CALX_mul_b 1#1!{1#1\expandafter!\the\numexpr\CALX_mul_loop }%
\def\CALX_mul_e;#1\W 1#2\W #3\W {1\relax #2}%
\def\CALX_minimulwc_a 1#1\xint:#2\xint:#3!#4#5#6#7#8\xint:%
{%
    \expandafter\CALX_minimulwc_b
    \the\numexpr \xint_c_e_ix+#1+#3*#8\xint:
                        #3*#4#5#6#7+#2*#8\xint:
                              #2*#4#5#6#7\xint:%
}%
\def\CALX_minimulwc_b 1#1#2#3#4#5#6\xint:#7\xint:%
{%
    \expandafter\CALX_minimulwc_c
    \the\numexpr \xint_c_e_ix+#1#2#3#4#5+#7\xint:#6\xint:%
}%
\def\CALX_minimulwc_c 1#1#2#3#4#5#6\xint:#7\xint:#8\xint:%
{%
    1#6#7\expandafter!%
    \the\numexpr\expandafter\CALX_smallmul_a
    \the\numexpr \xint_c_e_viii+#1#2#3#4#5+#8\xint:%
}%
\def\CALX_smallmul 1#1#2#3#4#5!{\CALX_smallmul_a 100000000\xint:#1#2#3#4%
    \xint:#5!}%
\def\CALX_smallmul_a #1\xint:#2\xint:#3!1#4!%
{%
    \xint_gob_til_sc #4\CALX_smallmul_e;%
```

```
    \CALX_minimulwc_a #1\xint:#2\xint:#3!#4\xint:#2\xint:#3!%
}%
\def\CALX_smallmul_e;\CALX_minimulwc_a 1#1\xint:#2;#3!%
    {\xint_gob_til_eightzeroes #1\CALX_smallmul_f 000000001\relax #1!1;!}%
\def\CALX_smallmul_f 000000001\relax 00000000!1{1\relax}%
\def\CALX_verysmallmul #1\xint:#2!1#3!%
{%
    \xint_gob_til_sc #3\CALX_verysmallmul_e;%
    \expandafter\CALX_verysmallmul_a
    \the\numexpr #2*#3+#1\xint:#2!%
}%
\def\CALX_verysmallmul_e;\expandafter\CALX_verysmallmul_a\the\numexpr
    #1+#2#3\xint:#4!%
{\xint_gob_til_zero #2\CALX_verysmallmul_f 0\xint_c_e_viii+#2#3!1;!}%
\def\CALX_verysmallmul_f #1!1{1\relax}%
\def\CALX_verysmallmul_a #1#2\xint:%
{%
    \unless\ifnum #1#2<\xint_c_e_ix
    \expandafter\CALX_verysmallmul_bi\else
    \expandafter\CALX_verysmallmul_bj\fi
    \the\numexpr \xint_c_e_ix+#1#2\xint:%
}%
\def\CALX_verysmallmul_bj{\expandafter\CALX_verysmallmul_cj }%
\def\CALX_verysmallmul_cj 1#1#2\xint:%
    {1#2\expandafter!\the\numexpr\CALX_verysmallmul #1\xint:}%
\def\CALX_verysmallmul_bi\the\numexpr\xint_c_e_ix+#1#2#3\xint:%
    {1#3\expandafter!\the\numexpr\CALX_verysmallmul #1#2\xint:}%
\def\CALX_minimul_a #1\xint:#2!#3#4#5#6#7!%
{%
    \expandafter\CALX_minimul_b
    \the\numexpr \xint_c_e_viii+#2*#7\xint:#2*#3#4#5#6+#1*#7\xint:#1*#3#4#5#6%
        \xint:%
}%
\def\CALX_minimul_b 1#1#2#3#4#5\xint:#6\xint:%
{%
    \expandafter\CALX_minimul_c
    \the\numexpr \xint_c_e_ix+#1#2#3#4+#6\xint:#5\xint:%
}%
\def\CALX_minimul_c 1#1#2#3#4#5#6\xint:#7\xint:#8\xint:%
{%
    1#6#7\expandafter!\the\numexpr \xint_c_e_viii+#1#2#3#4#5+#8!%
}%
```

29.9.3. \CALX_smalldivx_a

From this subsection of xintcore (\xintiiDivision) calx copies only the macros used by the native layer: \CALX_smalldivx_a, \CALX_smalldivx_b, \CALX_smalldiv_c, \CALX_smalldiv_d, \CALX_smalldiv_end, \CALX_smalldiv_e, \CALX_smalldiv_f, \CALX_smalldiv_fz, \CALX_smalldiv_g, \CALX_smalldiv_h, \CALX_smalldiv_i, \CALX_smalldiv_j, \CALX_smalldiv_k. The other macros are not copied.

```
\def\CALX_smalldivx_a #1\xint:1#2!1#3!%
{%
    \expandafter\CALX_smalldivx_b
```

```
    \the\numexpr (#3+#1)/#2-\xint_c_i!#1\xint:#2!#3!%
}%
\def\CALX_smalldivx_b #1#2!%
{%
    \if0#1\else
          \xint_c_e_viii+#1#2\xint_afterfi{\expandafter!\the\numexpr}\fi
    \CALX_smalldiv_c #1#2!%
}%
\def\CALX_smalldiv_c #1!#2\xint:#3!#4!%
{%
    \expandafter\CALX_smalldiv_d\the\numexpr #4-#1*#3!#2\xint:#3!%
}%
\def\CALX_smalldiv_d #1!#2!1#3#4!%
{%
    \xint_gob_til_sc #3\CALX_smalldiv_end ;%
    \CALX_smalldiv_e #1!#2!1#3#4!%
}%
\def\CALX_smalldiv_end;\CALX_smalldiv_e #1!#2!1;!{1!;!#1!}%
\def\CALX_smalldiv_e #1!#2\xint:#3!%
{%
    \expandafter\CALX_smalldiv_f\the\numexpr
    \xint_c_xi_e_viii_mone+#1*\xint_c_e_viii/#3!#2\xint:#3!#1!%
}%
\def\CALX_smalldiv_f 1#1#2#3#4#5#6!#7\xint:#8!%
{%
     \xint_gob_til_zero #1\CALX_smalldiv_fz 0%
     \expandafter\CALX_smalldiv_g
     \the\numexpr\CALX_minimul_a #2#3#4#5\xint:#6!#8!#2#3#4#5#6!#7\xint:#8!%
}%
\def\CALX_smalldiv_fz 0%
    \expandafter\CALX_smalldiv_g\the\numexpr\CALX_minimul_a
    9999\xint:9999!#1!99999999!#2!0!1#3!%
{%
    \CALX_smalldiv_i \xint:#3!\xint_c_!#2!%
}%
\def\CALX_smalldiv_g 1#1!1#2!#3!#4!#5!#6!%
{%
    \expandafter\CALX_smalldiv_h\the\numexpr 1#6-#1\xint:#2!#5!#3!#4!%
}%
\def\CALX_smalldiv_h 1#1#2\xint:#3!#4!%
{%
    \expandafter\CALX_smalldiv_i\the\numexpr #4-#3+#1-\xint_c_i\xint:#2!%
}%
\def\CALX_smalldiv_i #1\xint:#2!#3!#4\xint:#5!%
{%
    \expandafter\CALX_smalldiv_j\the\numexpr (#1#2+#4)/#5-%
        \xint_c_i!#3!#1#2!#4\xint:#5!%
}%
\def\CALX_smalldiv_j #1!#2!%
{%
    \xint_c_e_viii+#1+#2\expandafter!\the\numexpr\CALX_smalldiv_k
    #1!%
```

```
}%
\def\CALX_smalldiv_k #1!#2!#3\xint:#4!%
{%
    \expandafter\CALX_smalldiv_d\the\numexpr #2-#1*#4!#3\xint:#4!%
}%
```

29.10. Native integer format

A calx integer in native format is the sequence of catcode 12 tokens
   N <sign> <n> . <block_1> <block_2> ... <block_n>
- N: type tag;
  - <sign>: + or -, always present; zero is N+0. (no blocks);
  - <n>: number of blocks, decimal, terminated by a dot;
  - the blocks are xint's blocks 1ddddddddd! (the 1 protects leading zeros for \numexpr), least significant block first;
  - the most significant block (the last one) is not 00000000.
  The value is <sign> sum_{k=1}^{n} d_k 10^{8(k-1)} with d_k the 8 digits of <block_k>. There is no terminator: macros append their own delimiters.
  All macros below are used as \romannumeral0\CALX_n...{...}{...} and deliver their result after a space token. Arguments are f-expanded first (as in xintcore).
  Coding rules followed here (second pass, efficiency):
  - block lists are copied as few times as possible: a macro parameter holding blocks appears ONCE in each replacement text; branch selection (\xint_dothis/\xint_orthat, \ifcase) only selects a macro token, the arguments follow once after the selection;
  - the header N<sign><n>. of a result is produced in front of the blocks by \expandafter\CALX_sp\expandafter N\expandafter<sign>\the\numexpr<n>. so that the blocks, once grabbed to remove the kernel terminator 1;!, are not grabbed again (\CALX_sp expands to the space which ends the \romannumeral0 expansion);
  - counting the blocks of a product skips them seven at a time.
  In this section N and I have catcode 12, so macro names here use neither letter in upper case.

```
\catcode`N=12 \catcode`I=12
```

\CALX_sp expands to a space token; it ends the \romannumeral0 expansion of the native macros.

```
\def\CALX_sp{ }%
```

\CALX_pten{k} = 10^k, 0 <= k <= 8, k a \numexpr expression: one \csname lookup of \CALX_pt<k> (k in decimal), cheaper than an \ifcase table. The other value tables and dispatches of calx are built in the same way.

```
\def\CALX_pten #1{\csname CALX_pt\the\numexpr#1\endcsname}%
\expandafter\def\csname CALX_pt0\endcsname{1}%
\expandafter\def\csname CALX_pt1\endcsname{10}%
\expandafter\def\csname CALX_pt2\endcsname{100}%
\expandafter\def\csname CALX_pt3\endcsname{1000}%
\expandafter\def\csname CALX_pt4\endcsname{10000}%
\expandafter\def\csname CALX_pt5\endcsname{100000}%
\expandafter\def\csname CALX_pt6\endcsname{1000000}%
\expandafter\def\csname CALX_pt7\endcsname{10000000}%
\expandafter\def\csname CALX_pt8\endcsname{100000000}%
```

Digit counts without \ifnum (each \ifnum scans its two numbers: three nested \ifnum on an 8-digit block cost about twice as much). The digits are followed by a sentinel list and the ninth token is the answer (xintfrac's \XINT_FL_fac_countdigits); \CALX_nd_b removes the leftover sentinel up to \xint: and outputs the answer, so the macros are f-expandable.

- \CALX_ndig{v}: number of digits of 0 <= v < 10^8 (1 for 0); v may have leading zeros (removed by \numexpr);
- \CALX_ndigs{v}: same, v explicit digits without leading zeros;
- \CALX_nlz{v}: number of leading zeros of an 8-digit block v > 0.

\CALX_nd_c <v><s_1>...<s_10>\xint: (v: 1 to 10 explicit digits, no leading zero; s_k: one token or a braced group) outputs s_k for k = 11 - (number of digits of v), the eleventh token: a value per digit count (sentinel list of the caller).

```
\def\CALX_ndig #1%
   {\expandafter\CALX_nd_a\the\numexpr#1\relax 87654321\xint:}%
\def\CALX_ndigs #1{\CALX_nd_a #187654321\xint:}%
\def\CALX_nlz #1%
   {\expandafter\CALX_nd_a\the\numexpr#1\relax 01234567\xint:}%
\def\CALX_nd_a #1#2#3#4#5#6#7#8#9{\CALX_nd_b #9}%
\def\CALX_nd_b #1#2\xint:{#1}%
\def\CALX_nd_c #1#2#3#4#5#6#7#8#9{\CALX_nd_d}%
\def\CALX_nd_d #1#2#3\xint:{#2}%
```

29.10.1. \CALX_nfromraw

Syntax (use with \romannumeral0; R f-expanded):
\CALX_nfromraw{R} -> <space>N<s><n>.<blocks>
Conversion from the raw integer format of xintcore (optional minus sign, no leading zeros, zero is 0) to native. The shared \CALX_sepandrev_andcount, as called by \XINT_add_pp_a, delivers <n>\xint:<blocks LS-first>, and \CALX_nfromraw_c inserts the header without touching the blocks.

```
\def\CALX_nfromraw #1{\expandafter\CALX_nfromraw_a\romannumeral`&&@#1\xint:}%
\def\CALX_nfromraw_a #1{%
    \xint_UDzerominusfork
       #1-{\CALX_nfromraw_z}%
       0#1{\CALX_nfromraw_b -}%
        0-{\CALX_nfromraw_b +#1}%
    \krof
}%
\def\CALX_nfromraw_z #1\xint:{ N+0.}%
\def\CALX_nfromraw_b #1#2\xint:{%
    \expandafter\CALX_nfromraw_c\expandafter#1%
    \romannumeral0\expandafter\CALX_sepandrev_andcount
    \romannumeral0\CALX_zeroes_forviii #2\R\R\R\R\R\R\R\R{10}0000001\W
    #2\CALX_rsepbyviii_end_A 2345678%
      \CALX_rsepbyviii_end_B 2345678\relax\xint_c_ii\xint_c_i
          \R\xint:\xint_c_xii \R\xint:\xint_c_x  \R\xint:\xint_c_viii \R%
              \xint:\xint_c_vi
          \R\xint:\xint_c_iv  \R\xint:\xint_c_ii \R\xint:\xint_c_\W
}%
\def\CALX_nfromraw_c #1#2\xint:{ N#1#2.}%
```

29.10.2. \CALX_ntoraw

Syntax (use with \romannumeral0; X f-expanded native integer):
\CALX_ntoraw{X} -> <raw integer> (xintcore format: -digits, digits, 0)
Conversion to raw: the shared \CALX_unrevbyviii then \CALX_cuz_small, as in \XINT_add_out. The header is examined without grabbing the blocks.

```
\def\CALX_ntoraw #1{\expandafter\CALX_ntoraw_a\romannumeral`&&@#1\xint:}%
```

```
\def\CALX_ntoraw_a N#1#2.{%
    \ifnum#2=\xint_c_ \xint_dothis\CALX_ntoraw_z\fi
    \if-#1\xint_dothis\CALX_ntoraw_m\fi
    \xint_orthat\CALX_ntoraw_b
}%
\def\CALX_ntoraw_z #1\xint:{ 0}%
\def\CALX_ntoraw_m {\expandafter-\romannumeral0\CALX_ntoraw_b}%
\def\CALX_ntoraw_b #1\xint:{%
    \expandafter\CALX_cuz_small\romannumeral0\CALX_unrevbyviii {}#11;!%
    1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W
}%
```

29.10.3. \CALX_nskip

Skipping blocks. \CALX_nskip <cont><k>.<blocks> removes k blocks (k a \numexpr expression, k>=0) and leaves <cont> in front of what follows. The whole skip is done inside \romannumeral\csname, <cont> waits behind an \expandafter. \CALX_nskip_a takes the first two tokens of k: if the second one is the dot (k <= 9), \xint_gob_til_dot leaves \CALX_nskip_s, which completes the name \CALX_nskip<k>; this macro removes the k blocks and expands to \xint_c_ (ends \romannumeral). Otherwise \CALX_nskip_g removes eight blocks and loops.

```
\def\CALX_nskip #1#2.{\expandafter#1\romannumeral
                      \csname\expandafter\CALX_nskip_a\the\numexpr#2.}%
\def\CALX_nskip_a #1#2%
   {\xint_gob_til_dot #2\CALX_nskip_s.\CALX_nskip_g #1#2}%
\def\CALX_nskip_s.\CALX_nskip_g #1.{CALX_nskip#1\endcsname}%
\def\CALX_nskip_g #1.#2!#3!#4!#5!#6!#7!#8!#9!%
    {\expandafter\CALX_nskip_a\the\numexpr#1-\xint_c_viii.}%
\expandafter\let\csname CALX_nskip0\endcsname\xint_c_
\expandafter\def\csname CALX_nskip1\endcsname #1!{\xint_c_}%
\expandafter\def\csname CALX_nskip2\endcsname #1!#2!{\xint_c_}%
\expandafter\def\csname CALX_nskip3\endcsname #1!#2!#3!{\xint_c_}%
\expandafter\def\csname CALX_nskip4\endcsname #1!#2!#3!#4!{\xint_c_}%
\expandafter\def\csname CALX_nskip5\endcsname #1!#2!#3!#4!#5!{\xint_c_}%
\expandafter\def\csname CALX_nskip6\endcsname #1!#2!#3!#4!#5!#6!{\xint_c_}%
\expandafter\def\csname CALX_nskip7\endcsname #1!#2!#3!#4!#5!#6!#7!%
   {\xint_c_}%
\expandafter\def\csname CALX_nskip8\endcsname #1!#2!#3!#4!#5!#6!#7!#8!%
   {\xint_c_}%
\expandafter\def\csname CALX_nskip9\endcsname #1!#2!#3!#4!#5!#6!#7!#8!#9!%
   {\xint_c_}%
```

29.11. Sign and zero tests of values

29.11.1. \CALX_ifnz, \CALX_ifsgn

Syntax (use with \romannumeral0; X f-expanded):

```
  \CALX_ifnz{X}{A}{B} -> <space>A (X <> 0) or <space>B
    \CALX_ifnz_a <v>\xint:<y><n>        -> <y> (v <> 0) or <n>
    \CALX_ifsgn{X}{A}{B}{C}             -> <space>A, B or C (X <0, =0, >0)
    \CALX_ifsgn_a <v>\xint:{A}{B}{C}    -> the same
```

The values are those of the three parsers, after f-expansion: native N<s><n>., M<s><n>.<e>., F<s><D>.<e>. (blocks after the header), I+, I-, I?, or raw A, A[N], A/B[N] (A a raw integer without leading zeros, B > 0). The low level forms take the value v already expanded, in front of the \xint:

that ends it; \CALX_ifnz_a also takes the two tokens <y> and <n> which select a branch (\CALX_ifnz uses \xint_stop_atfirstoftwo, \xint_stop_atsecondoftwo; loops use \xint_firstoftwo, \xint_secondoftwo, which add no space).

Only the first tokens are read, no xintfrac parsing: 0 is zero, - is negative, another digit is positive; a token of character code above `9 is a tag: I+ and I- are infinities, I? is non-zero for \CALX_ifnz and signals InvalidOperation for \CALX_ifsgn, which then takes the zero branch; after F, M, N comes the sign token, then the count, whose first character is 0 only for zero.

```
\def\CALX_ifnz #1{\expandafter\CALX_ifnz_a\romannumeral`&&@#1\xint:%
    \xint_stop_atfirstoftwo\xint_stop_atsecondoftwo}%
\def\CALX_ifnz_a #1%
{%
    \if0#1\xint_dothis\CALX_ifnz_z\fi
    \ifnum`#1>57 \xint_dothis\CALX_ifnz_t\fi
    \xint_orthat\CALX_ifnz_n #1%
}%
\def\CALX_ifnz_t #1#2%
   {\if I#1\expandafter\CALX_ifnz_n\else\expandafter\CALX_ifnz_c\fi}%
\def\CALX_ifnz_c #1%
   {\if0#1\expandafter\CALX_ifnz_z\else\expandafter\CALX_ifnz_n\fi}%
\def\CALX_ifnz_z #1\xint:#2#3{#3}%
\def\CALX_ifnz_n #1\xint:#2#3{#2}%
\def\CALX_ifsgn #1{\expandafter\CALX_ifsgn_a\romannumeral`&&@#1\xint:}%
\def\CALX_ifsgn_a #1%
{%
    \if0#1\xint_dothis\CALX_ifsgn_z\fi
    \if-#1\xint_dothis\CALX_ifsgn_m\fi
    \ifnum`#1>57 \xint_dothis\CALX_ifsgn_t\fi
    \xint_orthat\CALX_ifsgn_p #1%
}%
\def\CALX_ifsgn_t #1#2%
   {\if I#1\expandafter\CALX_ifsgn_i\else\expandafter\CALX_ifsgn_c\fi #2}%
\def\CALX_ifsgn_c #1#2%
{%
    \if0#2\xint_dothis\CALX_ifsgn_z\fi
    \if-#1\xint_dothis\CALX_ifsgn_m\fi
    \xint_orthat\CALX_ifsgn_p
}%
\def\CALX_ifsgn_i #1%
{%
    \if-#1\xint_dothis\CALX_ifsgn_m\fi
    \if+#1\xint_dothis\CALX_ifsgn_p\fi
    \xint_orthat\CALX_ifsgn_q
}%
\def\CALX_ifsgn_q #1\xint:%
   {\expandafter\CALX_ifsgn_z\romannumeral`&&@%
    \CALX_signalcondition{InvalidOperation}{sign of I?}{}{}\xint:}%
\def\CALX_ifsgn_z #1\xint:{\xint_stop_atsecondofthree}%
\def\CALX_ifsgn_m #1\xint:{\xint_stop_atfirstofthree}%
\def\CALX_ifsgn_p #1\xint:{\xint_stop_atthirdofthree}%
```

29.12. Native addition and subtraction

29.12.1. \CALX_nadd

Syntax (use with \romannumeral0; X, Y f-expanded, native integers or I+, I-, I?):
  \CALX_nadd{X}{Y} -> <space>Z Z = X + Y
    \CALX_nsub{X}{Y} -> <space>Z   Z = X - Y
Z: native integer, or I+, I-, I? (special operands, rules of the signals section).
  Addition and subtraction: dispatch.
  \CALX_nadd{X}{Y}: \CALX_nadd_x f-expands Y and leaves
   Y\xint:X1;!1;!1;!1;!\W\xint:
so that X is already followed by the kernel terminator. With P = Y (count n, sign s) and Q = X (count m, sign t), \CALX_nadd_y grabs only P's list and Q's header N<t><m>. and runs a chain of tests. A test that fires puts a "jumper" in its branch: a macro taking a few data tokens, delimited by the \fi of that branch, which swallows the rest of \CALX_nadd_y's body (P's list included) and starts with the \fi it removed. Without jumper, the common case runs inline: same signs and n<=m, kernel started with P's list, Q's list left where it is.

n=0: Q	\CALX_nadd_retx
m=0: P	\CALX_nadd_rety
s<>t, n<m: |Q|-|P|, sign t, Q in place	\CALX_ndiff_q
s<>t, n>m: |P|-|Q|, sign s	\CALX_ndiff_p
s<>t, n=m: P-Q, possibly negative	\CALX_ndiff_e
s=t, n>m: Q shorter	\CALX_nsum_q
s=t, n<=m: inline	

The count of the result (m, or n) is written BEFORE the kernel:
    \expandafter\CALX_naddm\expandafter<sign>\the\numexpr<count>
      \expandafter.\the\numexpr\CALX_nadd_a\xint_c_ii<S>1;!1;!1;!1;!\W
      <L>1;!1;!1;!1;!\W\xint:
Lists grabbed: P's once more in the jumper cases, Q's once in \CALX_ndiff_p, _e and \CALX_nsum_q, the result once by the final macro (its header must precede it).
  Addition kernel: xintcore's \XINT_add_a ... \XINT_add_o (xintcore l.598-668, code-only numbering), copied as \CALX_nadd_a ... \CALX_nadd_o.
  Contract: first list the SHORTER (or equal) one, both lists followed by 1;!1;!1;!1;!\W, initial state 2; output: the LS-first sum blocks followed by 1;!, then the tokens which followed the second list's \W. The only place where the result gets one more block than the longer operand is \X
↪ INT_add_lf, which appends block 00000001; its copy \CALX_nadd_lf also emits +\xint_c_i after the 1;!, before the final \xint:. \CALX_naddm grabs <count>. the list and that optional increment.
  Subtraction kernel: xintcore's \XINT_sub_a ... \XINT_sub_i with \XINT_sub_bi, _di, _fi, _hi (xintcore l.786-851, code-only numbering), copied as \CALX_nsub_*. Contract: called as
    \CALX_nsub_a\xint_c_i <S>1;!1;!1;!1;!\W <L>1;!1;!1;!1;!\W
with S not longer than L, it computes L-S; each step produces a block and a state digit, 1 for no borrow, 0 for borrow; when S is exhausted it calls
    \CALX_nsub_k <state><remaining L blocks>1;!1;!1;!1;!\W
xintcore's \XINT_sub_k and what follows are tied to its decimal output; the end of the loop (\CALX
↪ _nsub_k and the following) is calx's own. It ends the list with 1;! followed by one digit, before the final \xint:.
  Unequal lengths (n_L > n_S): remaining L blocks are not empty and |L| > |S|, so a final borrow is absorbed by L:
  - no borrow: copy the remaining blocks (\CALX_nsub_kc), digit 0;
  - borrow: 00000000 blocks become 99999999 and the borrow goes on; the first non zero block d becomes d-1, then copy (\CALX_nsub_kb). If d is the top block (next is 1;!) and d=1, the block is dropped (\CALX_nsub_kt) and the digit is 1: the blocks produced by the loop below it may be zero

(e.g. 10^16-(10^16-1)). This happens only when n_L=n_S+1, the top block of L is 00000001 and absorbs a borrow.

\CALX_nsubu: digit 0, the result is final; digit 1, \CALX_nsubrd takes the n_L-1 blocks again with \CALX_ntakex and normalizes (\CALX_nnorm).

Equal lengths: remaining L blocks are empty; \CALX_nsub_ke ends the list and leaves the state as digit. State 0 means P<Q and the blocks hold 10^(8n)+P-Q: \CALX_nsubng<sign> complements them (\CALX_nnega), flips the sign and goes to \CALX_nsubf. \CALX_nsubf examines the top block (\CALX_ntop_c): if it is not zero, the count is n and the result is final; otherwise \CALX_nsubr strips zero top blocks (\CALX_nnorm).

\CALX_ntakex <n>. takes exactly n blocks 1<8 digits>! from the input stream and leaves them followed by 1;! (inside \numexpr chains): the list then needs no delimiter, so the final result N<s><n>.<list> can be left as is in the common case and taken again only in the rare ones.

```
\def\CALX_nadd #1{\expandafter\CALX_nadd_x\romannumeral`&&@#1\xint:}%
\def\CALX_nadd_x #1#2\xint:#3{\expandafter\CALX_nsp_t\expandafter\CALX_nadd_y
    \expandafter\CALX_nadd_sp\expandafter#1\romannumeral`&&@#3\xint:#1#2%
        1;!1;!1;!1;!\W\xint:}%
```

Special values (SPEC_native D9). \CALX_nsp_t <fast><special><tX><tY> (tags of X and Y, the stream continues with Y without its tag, then \xint:X<tail>\xint:, see below): both tags N: <fast><tY>, else <special><tX><tY>. For \CALX_nmul <tail> is empty.

\CALX_nadd_sp <tX><tY><sY>...\xint:<tX><sX>...\xint: (the tail of X is swallowed with it; s = second token: sign, or + - ? after I): NaN in an operand: I?; Inf + finite: that Inf; Inf + Inf: that Inf if same sign, else InvalidOperation (default I?).

\CALX_nsub_sp flips sY (I? unchanged) and goes to \CALX_nadd_sp.

```
\def\CALX_nsp_t #1#2#3#4%
   {\if\if N#3#4\else ?\fi N\xint_dothis{#1#4}\fi\xint_orthat{#2#3#4}}%
\def\CALX_nadd_sp #1#2#3#4\xint:#5#6#7\xint:%
{%
    \if ?#3\xint_dothis{ I?}\fi
    \if ?#6\xint_dothis{ I?}\fi
    \if N#2\xint_dothis{ I#6}\fi
    \if N#5\xint_dothis{ I#3}\fi
    \if #3#6\xint_dothis{ I#3}\fi
    \xint_orthat{\CALX_nsig{InvalidOperation}{inf - inf}{}}%
}%
\def\CALX_nadd_y N#1#2.#3\xint:N#4#5.%
{%
    \ifnum#2=\xint_c_ \CALX_nadd_retx#4#5.\fi
    \ifnum#5=\xint_c_ \CALX_nadd_rety#1#2.\fi
    \if#1#4\else
      \ifnum#2<#5 \CALX_ndiff_q#4#5.\fi
      \ifnum#2>#5 \CALX_ndiff_p#1#2.\fi
      \CALX_ndiff_e#1#2.%
    \fi
    \ifnum#2>#5 \CALX_nsum_q#1#2.\fi
    \expandafter\CALX_naddm\expandafter#1\the\numexpr#5\expandafter.%
    \the\numexpr\CALX_nadd_a\xint_c_ii#31;!1;!1;!1;!\W
}%
\def\CALX_nadd_retx#1#2.\fi#3\W#41;!1;!1;!1;!\W\xint:{\fi\CALX_sp N#1#2.#4}%
\def\CALX_nadd_rety#1#2.\fi#3\xint_c_ii#41;!1;!1;!1;!\W
    1;!1;!1;!1;!\W\xint:{\fi\CALX_sp N#1#2.#4}%
\def\CALX_nsum_q#1#2.\fi#3\xint_c_ii#41;!1;!1;!1;!\W
    #51;!1;!1;!1;!\W\xint:%
```

```
{%
    \fi\expandafter\CALX_naddm\expandafter#1\the\numexpr#2\expandafter.%
    \the\numexpr\CALX_nadd_a\xint_c_ii#51;!1;!1;!1;!\W#41;!1;!1;!1;!\W\xint:
}%
\def\CALX_naddm #1#2.#31;!#4\xint:%
    {\expandafter\CALX_sp\expandafter N\expandafter#1\the\numexpr#2#4.#3}%
\def\CALX_ndiff_q#1#2.\fi#3\xint_c_ii#41;!1;!1;!1;!\W
{%
    \fi\fi\expandafter\CALX_nsubu\expandafter#1\the\numexpr#2\expandafter.%
    \the\numexpr\CALX_nsub_a\xint_c_i#41;!1;!1;!1;!\W
}%
\def\CALX_ndiff_p#1#2.\fi#3\xint_c_ii#41;!1;!1;!1;!\W
    #51;!1;!1;!1;!\W\xint:%
{%
    \fi\fi\expandafter\CALX_nsubu\expandafter#1\the\numexpr#2\expandafter.%
    \the\numexpr\CALX_nsub_a\xint_c_i#51;!1;!1;!1;!\W#41;!1;!1;!1;!\W\xint:
}%
\def\CALX_ndiff_e#1#2.\fi#3\xint_c_ii#41;!1;!1;!1;!\W
    #51;!1;!1;!1;!\W\xint:%
{%
    \fi\expandafter\CALX_nsube\expandafter#1\the\numexpr#2\expandafter.%
    \the\numexpr\CALX_nsub_a\xint_c_i#51;!1;!1;!1;!\W#41;!1;!1;!1;!\W\xint:
}%
\def\CALX_nsubu #1#2.#31;!#4\xint:%
{%
    \ifnum#4=\xint_c_ \expandafter\CALX_sp\else\expandafter\CALX_nsubrd\fi
    N#1#2.#3%
}%
\def\CALX_nsube #1#2.#31;!#4\xint:%
{%
    \if0#4\csname CALX_nsubng#1\expandafter\endcsname
    \else\expandafter\CALX_nsubf\fi #1#2.#31;!%
}%
\expandafter\def\csname CALX_nsubng+\endcsname #1#2.#31;!%
{%
    \expandafter\CALX_nsubf\expandafter-\the\numexpr#2\expandafter.%
    \the\numexpr\CALX_nnega#31;!%
}%
\expandafter\def\csname CALX_nsubng-\endcsname #1#2.#31;!%
{%
    \expandafter\CALX_nsubf\expandafter+\the\numexpr#2\expandafter.%
    \the\numexpr\CALX_nnega#31;!%
}%
\def\CALX_nsubf #1#2.#31;!%
{%
    \ifnum\CALX_nskip\CALX_ntop_c #2-\xint_c_i.#3\xint:=\xint_c_
      \expandafter\CALX_nsubr\else\expandafter\CALX_sp\fi
    N#1#2.#3%
}%
\def\CALX_nsubr N#1#2.%
    {\expandafter\CALX_nnorm\expandafter#1\the\numexpr\CALX_ntakex#2.}%
\def\CALX_nsubrd N#1#2.%
```

```
{%
    \expandafter\CALX_nnorm\expandafter#1\the\numexpr
    \expandafter\CALX_ntakex\the\numexpr#2-\xint_c_i.%
}%
\def\CALX_ntakex #1.%
{%
    \ifnum#1=\xint_c_ \expandafter\CALX_ntakex_e\else
                      \expandafter\CALX_ntakex_b\fi #1.%
}%
\def\CALX_ntakex_e #1.{1\relax;!}%
\def\CALX_ntakex_b #1.1#2!%
{%
    1#2\expandafter!\the\numexpr
    \expandafter\CALX_ntakex\the\numexpr#1-\xint_c_i.%
}%
\def\CALX_ntop_c 1#1!#2\xint:{#1}%
\def\CALX_nsub_k #1#2#3{%
    \xint_gob_til_sc #3\CALX_nsub_ke;%
    \xint_gob_til_zero #1\CALX_nsub_kz0\CALX_nsub_kc #2#3%
}%
\def\CALX_nsub_ke;\xint_gob_til_zero#1#2\W{1\relax;!#1}%
\def\CALX_nsub_kz0\CALX_nsub_kc{\CALX_nsub_kb}%
\def\CALX_nsub_kc #11;!1;!1;!1;!\W{#11;!0}%
\def\CALX_nsub_kb 1#1!#2#3{%
    \xint_gob_til_sc #3\CALX_nsub_kt;%
    \ifnum#1=\xint_c_ \expandafter\CALX_nsub_kbz\else
                      \expandafter\CALX_nsub_kbn\fi #1!#2#3%
}%
\def\CALX_nsub_kbz #1!{199999999\expandafter!\the\numexpr\CALX_nsub_kb}%
\def\CALX_nsub_kbn #1!{1#1-\xint_c_i\expandafter!\the\numexpr\CALX_nsub_kc}%
\def\CALX_nsub_kt;\ifnum#1=#2\W{%
    \ifnum#1=\xint_c_i \expandafter\CALX_nsub_kto\else
                       \expandafter\CALX_nsub_ktn\fi #1!%
}%
\def\CALX_nsub_kto #1!{1\relax;!1}%
\def\CALX_nsub_ktn #1!{1#1-\xint_c_i\relax!1;!0}%
```

29.12.2. \CALX_nsub

\CALX_nsub{X}{Y} = X-Y: flips the sign of Y and goes to the addition (N-0. may arise transiently but is caught by the zero tests of \CALX_nadd_y before any use of the sign).

```
\def\CALX_nsub #1{\expandafter\CALX_nsub_x\romannumeral`&&@#1\xint:}%
\def\CALX_nsub_x #1#2\xint:#3{\expandafter\CALX_nsp_t\expandafter\CALX_nsub_y
    \expandafter\CALX_nsub_sp\expandafter#1\romannumeral`&&@#3\xint:#1#2%
        1;!1;!1;!1;!\W\xint:}%
\def\CALX_nsub_sp #1#2#3%
{%
    \if +#3\xint_dothis{\CALX_nadd_sp#1#2-}\fi
    \if -#3\xint_dothis{\CALX_nadd_sp#1#2+}\fi
    \xint_orthat{\CALX_nadd_sp#1#2#3}%
}%
\def\CALX_nsub_y N#1%
    {\if#1-\expandafter\CALX_nsub_yp\else\expandafter\CALX_nsub_ym\fi}%
```

```
\def\CALX_nsub_yp{\CALX_nadd_y N+}%
\def\CALX_nsub_ym{\CALX_nadd_y N-}%
```

29.12.3. \CALX_nnega, \CALX_nnegb, \CALX_nnorm, \CALX_nlast, \CALX_ntake

\CALX_nnega: 10^(8n)-R on LS-first blocks terminated by 1;! (R not zero): low zero blocks stay, the first non zero block d becomes 10^8-d (written 2*10^8-d to keep the protecting 1), the next ones 99999999-d (written 199999999-d).

```
\def\CALX_nnega 1#1!{%
    \ifnum#1=\xint_c_ 100000000\expandafter!\the\numexpr
                      \expandafter\CALX_nnega
    \else \xint_c_ii*\xint_c_e_viii-#1\expandafter!\the\numexpr
          \expandafter\CALX_nnegb
    \fi
}%
\def\CALX_nnegb 1#1#2!{\xint_gob_til_sc #1\CALX_nnegz;%
    199999999-#1#2\expandafter!\the\numexpr\CALX_nnegb}%
\def\CALX_nnegz;#1\CALX_nnegb{1\relax;!}%
```

Syntax:
  \CALX_nnorm <s><blocks>1;! -> <space>N<s><n>.<blocks'>
(blocks LS first, zero top blocks allowed; blocks' without them, N+0. if all are zero.)
\CALX_nnorm <sign> <blocks>1;!: strips the zero top blocks and counts, in two linear passes: \CALX_nlast finds the index of the last non zero block (arguments are two numbers <index>.<last>.), then \CALX_ntake emits that many blocks inside a \numexpr chain (xintcore's 1<block>\expandafter!\the\numexpr technique) and gobbles the rest up to \xint:. Returns N+0. if all blocks are zero.

```
\def\CALX_nnorm #1#21;!%
   {\expandafter\CALX_nnorm_b\romannumeral0\CALX_nlast 0.0.#21;!.#1#2\xint:}%
\def\CALX_nlast #1.#2.1#3#4!{%
    \xint_gob_til_sc #3\CALX_nlast_e;%
    \ifnum#3#4=\xint_c_
       \expandafter\CALX_nlast_z
    \else
       \expandafter\CALX_nlast_nz
    \fi #1.#2.%
}%
\def\CALX_nlast_z #1.#2.{\expandafter\CALX_nlast\the\numexpr#1+\xint_c_i.#2.}%
\def\CALX_nlast_nz #1.#2.%
   {\expandafter\CALX_nlast\the\numexpr#1+\xint_c_i\expandafter.\the%
       \numexpr#1+\xint_c_i.}%
\def\CALX_nlast_e;#1\fi #2.#3.{ #3}%
\def\CALX_nnorm_b #1.{\ifnum#1=\xint_c_ \CALX_nnorm_z\fi\CALX_nnorm_c #1.}%
\def\CALX_nnorm_z\fi#1\xint:{\fi\CALX_sp N+0.}%
\def\CALX_nnorm_c #1.#2#3\xint:%
   {\expandafter\CALX_nnorm_d\the\numexpr\CALX_ntake #1.#31;!\xint:#2#1.}%
\def\CALX_nnorm_d #11;!#2#3.{ N#2#3.#1}%
\def\CALX_ntake #1.1#2!{%
    \ifnum#1=\xint_c_ \expandafter\CALX_ntake_e\fi
    1#2\expandafter!\the\numexpr
    \expandafter\CALX_ntake\the\numexpr#1-\xint_c_i.%
}%
\def\CALX_ntake_e #1\xint:{1\relax;!}%
```

29.13. Native multiplication

29.13.1. \CALX_nmul

Syntax (use with \romannumeral0; X, Y f-expanded, native integers or I+, I-, I?):
\CALX_nmul{X}{Y} -> <space>Z Z = X Y (native integer, or I)
Multiplication. \CALX_nmul{X}{Y}. The kernels are used as they are, from the renamed copy of xintcore (they are self contained, the addition they use is the unmodified \CALX_add_a): \CALX_mul_loop, \CALX_smallmul, \CALX_verysmallmul. Only the dispatch (xintcore's \XINT_mul_checklengths, \XINT_mul_smallbyfirst/second) is rewritten, with the stored block counts, and the output.
Contracts (from the code, xintcore l.1013-1097 code-only):
- \the\numexpr\CALX_mul_loop 100000000!1;!\W <X>1;!\W <Y>1;! computes X*Y, iterating over the blocks of Y (the initial 100000000!1;! is the zero accumulator); output: product blocks LS-first then 1;!, then the tokens which followed Y's 1;!;
- \the\numexpr\CALX_smallmul 1<y>!<X>1;! for a single block y, and \the\numexpr\CALX_verysmallmul 0\xint:<y>!<X>1;! for y<22 (xintcore's threshold): same output form.
These kernels never produce a zero top block (the final carry block is dropped when zero, \XINT_smallmul_f, \XINT_verysmallmul_f). A product of n and m blocks has n+m-1 or n+m blocks: \CALX_nmulo<sign> receives n+m-1 (computed before the kernel), skips that many blocks (\CALX_nskip) and checks whether one more follows.
Dispatch as for \CALX_nadd: \CALX_nmul_x writes X followed by 1;!\xint:, \CALX_nmul_y grabs P's list and Q's header; jumpers \CALX_nmul_zero, \CALX_nmul_sp (P one block: Q stays in place), \CALX_nmul_sq (Q one block), \CALX_nmul_xq (iteration over P: Q's list grabbed); inline default: iteration over Q, left in place. A one-block factor y uses \CALX_verysmallmul if y<22, else \CALX_smallmul (y=1 included).
Choice of the iterated operand: the rule of xintcore's \XINT_mul_checklengths, transcribed with X=Q, Y=P by default (A=P, B=Q in xintcore): exchange if m<n and (m-1)(n-m)<383, or if m>=n and (n-1)(m-n)>383.
\CALX_psign <s1><s2><macro> gives <macro>+ or <macro>- (sign of a product) by expansion (used by the division).

```
\def\CALX_psign #1#2#3%
    {\if#1#2\expandafter#3\expandafter+\else\expandafter#3\expandafter-\fi}%
\def\CALX_nmul #1{\expandafter\CALX_nmul_x\romannumeral`&&@#1\xint:}%
\def\CALX_nmul_x #1#2\xint:#3{\expandafter\CALX_nsp_t\expandafter\CALX_nmul_y
    \expandafter\CALX_nmul_spv\expandafter#1\romannumeral`&&@#3\xint:#1#2%
        1;!\xint:}%
```

\CALX_nmul_spv (same input as \CALX_nadd_sp): NaN: I?; Inf times 0: InvalidOperation (I?); otherwise Inf with the product of the signs.
\CALX_nsp_z <rest of a native after its tag and sign>: its block count.
\CALX_nsp_ps <s1><s2>: I+ or I- (product of the signs, with the space).

```
\def\CALX_nsp_z #1.#2\xint:{#1}%
\def\CALX_nsp_ps #1#2{\if#1#2\xint_dothis{ I+}\fi\xint_orthat{ I-}}%
\def\CALX_nmul_spv #1#2#3#4\xint:#5#6#7\xint:%
{%
    \if ?#3\xint_dothis{ I?}\fi
    \if ?#6\xint_dothis{ I?}\fi
    \ifnum\if N#2\CALX_nsp_z#4\xint:\else\if N#5\CALX_nsp_z#7\xint:\else
      \xint_c_i\fi\fi=\xint_c_
      \xint_dothis{\CALX_nsig{InvalidOperation}{inf * 0}{}}\fi
    \xint_orthat{\CALX_nsp_ps#3#6}%
}%
\def\CALX_nmul_y N#1#2.#3\xint:N#4#5.%
```

```
{%
    \ifnum#2=\xint_c_ \CALX_nmul_zero\fi
    \ifnum#5=\xint_c_ \CALX_nmul_zero\fi
    \ifnum#2=\xint_c_i \CALX_nmul_sp#1#4#2.#5.\fi
    \ifnum#5=\xint_c_i \CALX_nmul_sq#1#4#2.#5.\fi
    \ifnum\ifnum#5<#2 \numexpr383-(#5-\xint_c_i)*(#2-#5)\relax
          \else\numexpr(#2-\xint_c_i)*(#5-#2)-383\relax\fi>\xint_c_
    \else\CALX_nmul_xq#1#4#2.#5.\fi
    \csname CALX_nmulo\if#1#4+\else-\fi\expandafter\endcsname
    \the\numexpr#2+#5-\xint_c_i\expandafter.%
    \the\numexpr\CALX_mul_loop 100000000!1;!\W #31;!\W
}%
\def\CALX_nmul_zero\fi#1\W#2\W#31;!\xint:{\fi\CALX_sp N+0.}%
\def\CALX_nmul_sp#1#2#3.#4.\fi#5\W1#6!1;!\W
{%
    \fi\csname CALX_nmulo\if#1#2+\else-\fi\expandafter\endcsname
    \the\numexpr#4\expandafter.\the\numexpr
    \ifnum#6<\xint_c_xxii \expandafter\CALX_nmul_vs\else
                          \expandafter\CALX_nmul_s\fi #6!%
}%
\def\CALX_nmul_sq#1#2#3.#4.\fi#5\W#6\W1#7!1;!\xint:%
{%
    \fi\csname CALX_nmulo\if#1#2+\else-\fi\expandafter\endcsname
    \the\numexpr#3\expandafter.\the\numexpr
    \ifnum#7<\xint_c_xxii \expandafter\CALX_nmul_vs\else
                          \expandafter\CALX_nmul_s\fi #7!#6\xint:
}%
\def\CALX_nmul_vs #1!{\CALX_verysmallmul 0\xint:#1!}%
\def\CALX_nmul_s #1!{\CALX_smallmul 1#1!}%
\def\CALX_nmul_xq#1#2#3.#4.\fi#5\W#6\W#71;!\xint:%
{%
    \fi\csname CALX_nmulo\if#1#2+\else-\fi\expandafter\endcsname
    \the\numexpr#3+#4-\xint_c_i\expandafter.%
    \the\numexpr\CALX_mul_loop 100000000!1;!\W#71;!\W#6\xint:
}%
\expandafter\def\csname CALX_nmulo+\endcsname #1.#21;!\xint:%
{%
    \expandafter\CALX_sp\expandafter N\expandafter+\the\numexpr
    #1+\CALX_nskip\CALX_nmul_c #1.#21;!\xint:.#2%
}%
\expandafter\def\csname CALX_nmulo-\endcsname #1.#21;!\xint:%
{%
    \expandafter\CALX_sp\expandafter N\expandafter-\the\numexpr
    #1+\CALX_nskip\CALX_nmul_c #1.#21;!\xint:.#2%
}%
\def\CALX_nmul_c 1#1#2!#3\xint:{\xint_gob_til_sc #1\CALX_nmul_ce;\xint_c_i}%
\def\CALX_nmul_ce;\xint_c_i{\xint_c_}%
\def\CALX_nadd_a #1!#2!#3!#4!#5\W
                 #6!#7!#8!#9!%
{%
    \CALX_nadd_b
        #1!#6!#2!#7!#3!#8!#4!#9!%
```

```
        #5\W
}%
\def\CALX_nadd_b #11#2#3!#4!%
{%
    \xint_gob_til_sc #2\CALX_nadd_bi ;%
    \expandafter\CALX_nadd_c\the\numexpr#1+1#2#3+#4-\xint_c_ii\xint:%
}%
\def\CALX_nadd_bi;\expandafter\CALX_nadd_c
    \the\numexpr#1+#2+#3-\xint_c_ii\xint:#4!#5!#6!#7!#8!#9!\W
{%
    \CALX_nadd_k #1#3!#5!#7!#9!%
}%
\def\CALX_nadd_c #1#2\xint:%
{%
    1#2\expandafter!\the\numexpr\CALX_nadd_d #1%
}%
\def\CALX_nadd_d #11#2#3!#4!%
{%
    \xint_gob_til_sc #2\CALX_nadd_di ;%
    \expandafter\CALX_nadd_e\the\numexpr#1+1#2#3+#4-\xint_c_ii\xint:%
}%
\def\CALX_nadd_di;\expandafter\CALX_nadd_e
    \the\numexpr#1+#2+#3-\xint_c_ii\xint:#4!#5!#6!#7!#8\W
{%
    \CALX_nadd_k #1#3!#5!#7!%
}%
\def\CALX_nadd_e #1#2\xint:%
{%
    1#2\expandafter!\the\numexpr\CALX_nadd_f #1%
}%
\def\CALX_nadd_f #11#2#3!#4!%
{%
    \xint_gob_til_sc #2\CALX_nadd_fi ;%
    \expandafter\CALX_nadd_g\the\numexpr#1+1#2#3+#4-\xint_c_ii\xint:%
}%
\def\CALX_nadd_fi;\expandafter\CALX_nadd_g
    \the\numexpr#1+#2+#3-\xint_c_ii\xint:#4!#5!#6\W
{%
    \CALX_nadd_k #1#3!#5!%
}%
\def\CALX_nadd_g #1#2\xint:%
{%
    1#2\expandafter!\the\numexpr\CALX_nadd_h #1%
}%
\def\CALX_nadd_h #11#2#3!#4!%
{%
    \xint_gob_til_sc #2\CALX_nadd_hi ;%
    \expandafter\CALX_nadd_i\the\numexpr#1+1#2#3+#4-\xint_c_ii\xint:%
}%
\def\CALX_nadd_hi;%
    \expandafter\CALX_nadd_i\the\numexpr#1+#2+#3-\xint_c_ii\xint:#4\W
{%
```

```
    \CALX_nadd_k #1#3!%
}%
\def\CALX_nadd_i #1#2\xint:%
{%
    1#2\expandafter!\the\numexpr\CALX_nadd_a #1%
}%
\def\CALX_nadd_k #1{\if #12\expandafter\CALX_nadd_ke\else\expandafter%
    \CALX_nadd_l \fi}%
\def\CALX_nadd_ke #11;#2\W {\CALX_nadd_kf #11;!}%
\def\CALX_nadd_kf 1{1\relax }%
\def\CALX_nadd_l 1#1{\xint_gob_til_sc #1\CALX_nadd_lf ;\CALX_nadd_m 1#1}%
\def\CALX_nadd_lf #1\W {1\relax 00000001!1;!+\xint_c_i}% calx: count +1
\def\CALX_nadd_m #1!{\expandafter\CALX_nadd_n\the\numexpr\xint_c_i+#1\xint:}%
\def\CALX_nadd_n #1#2\xint:{1#2\expandafter!\the\numexpr\CALX_nadd_o #1}%
\def\CALX_nadd_o #1{\if #12\expandafter\CALX_nadd_l\else\expandafter%
    \CALX_nadd_ke \fi}%
\def\CALX_nsub_a #1!#2!#3!#4!#5\W #6!#7!#8!#9!%
{%
    \CALX_nsub_b
    #1!#6!#2!#7!#3!#8!#4!#9!%
    #5\W
}%
\def\CALX_nsub_b #1#2#3#4!#5!%
{%
    \xint_gob_til_sc #3\CALX_nsub_bi ;%
    \expandafter\CALX_nsub_c\the\numexpr#1+1#5-#3#4-\xint_c_i\xint:%
}%
\def\CALX_nsub_c 1#1#2\xint:%
{%
    1#2\expandafter!\the\numexpr\CALX_nsub_d #1%
}%
\def\CALX_nsub_d #1#2#3#4!#5!%
{%
    \xint_gob_til_sc #3\CALX_nsub_di ;%
    \expandafter\CALX_nsub_e\the\numexpr#1+1#5-#3#4-\xint_c_i\xint:
}%
\def\CALX_nsub_e 1#1#2\xint:%
{%
    1#2\expandafter!\the\numexpr\CALX_nsub_f #1%
}%
\def\CALX_nsub_f #1#2#3#4!#5!%
{%
    \xint_gob_til_sc #3\CALX_nsub_fi ;%
    \expandafter\CALX_nsub_g\the\numexpr#1+1#5-#3#4-\xint_c_i\xint:
}%
\def\CALX_nsub_g 1#1#2\xint:%
{%
    1#2\expandafter!\the\numexpr\CALX_nsub_h #1%
}%
\def\CALX_nsub_h #1#2#3#4!#5!%
{%
    \xint_gob_til_sc #3\CALX_nsub_hi ;%
```

```
    \expandafter\CALX_nsub_i\the\numexpr#1+1#5-#3#4-\xint_c_i\xint:
}%
\def\CALX_nsub_i 1#1#2\xint:%
{%
    1#2\expandafter!\the\numexpr\CALX_nsub_a #1%
}%
\def\CALX_nsub_bi;%
    \expandafter\CALX_nsub_c\the\numexpr#1+1#2-#3\xint:
    #4!#5!#6!#7!#8!#9!\W
{%
    \CALX_nsub_k #1#2!#5!#7!#9!%
}%
\def\CALX_nsub_di;%
    \expandafter\CALX_nsub_e\the\numexpr#1+1#2-#3\xint:
    #4!#5!#6!#7!#8\W
{%
    \CALX_nsub_k #1#2!#5!#7!%
}%
\def\CALX_nsub_fi;%
    \expandafter\CALX_nsub_g\the\numexpr#1+1#2-#3\xint:
    #4!#5!#6\W
{%
    \CALX_nsub_k #1#2!#5!%
}%
\def\CALX_nsub_hi;%
    \expandafter\CALX_nsub_i\the\numexpr#1+1#2-#3\xint:
    #4\W
{%
    \CALX_nsub_k #1#2!%
}%
```

29.14. Native division

29.14.1. \CALX_nudiv

```
Division (calx). Unsigned core:
   \CALX_nudiv <cont>{na}{A}{nb}{B}
with A, B non zero, given by their block counts and LS-first blocks. It expands to
   <cont>{nq}{Q}{s}{R'}
where nq, Q (count, LS-first blocks) is the quotient floor(A/B), and R' the LS-first blocks (non
normalized: zero top blocks possible) terminated by 1;! of 10^s times the remainder R = A - Q*B.
Use \CALX_nudiv_rem to get R in native format. The tokens following the last argument are left
untouched after <cont>{nq}{Q}{s}{R'}.
  Paths:
  - na < nb: Q = 0, R' = A, s = 0.
  - nb = 1, na = 1: \numexpr.
  - nb = 1, divisor d <= 45249: dedicated kernel (see below).
  - nb = 1, larger divisor: A reversed (\CALX_rev_nounsep), \CALX_smalldivx_a (from xintcore)
(MS-first), quotient reversed.
  - nb = 2, 3, 4: dedicated loops without normalization (see below), s = 0.
  - nb >= 5: Knuth's algorithm D in base 10^8, described now.
  Algorithm D (general path).
```

Normalisation: s = number of leading zeros of the top block of B (as an 8 digit block); A'' = 10^s A with exactly na+1 blocks (a zero block is appended before multiplying), B' = 10^s B (nb blocks, top block v1 >= 10^7). \CALX_smallmul (from xintcore) does both products, LS-first. The quotient is the same, the remainder is multiplied by 10^s.

Data: the window W (nb+1 blocks w_0..w_nb, W < 10^8 B') and B' (b_0 .. b_{nb-1}) are kept INTERLEAVED in one list

L = 1w_0!1b_0!1w_1!1b_1!...1w_{nb-1}!1b_{nb-1}!1w_nb!1;;;;;!

(1;;;;;! = marker Z). The remaining blocks of A'' are kept MS-first in the input stream, after L, terminated by 1;!.

The multiply-subtract pass \CALX_ndms_a reads the pairs (w_k,b_k) and emits b_k then r_k, where r = W - q*B' (one borrow chain); its output is

1b_0!1r_0!1b_1!1r_1!...1b_{nb-1}!1r_{nb-1}!1;;;;;!{fin}

with fin = 0 if r >= 0 and fin = -1 if r < 0 (then the blocks hold r + 10^(8nb)). Prefixing the next block of A'' gives exactly the next window in the layout of L: pairs (new,b_0), (r_0,b_1), ..., top (r_{nb-1},Z).

Estimation of q (with u0, u1, u2 the three top blocks of W, v1, v2 those of B', U = u0 b^2+u1 b+u2, V = v1 b+v2, b = 10^8): q* = min(floor(U/V), b-1) satisfies q <= q* <= q+1. Indeed W/B' < (U+1)/V gives q <= floor(U/V), and W/B' >= U/(V+1) gives q >= floor(U/(V+1)) >= floor(U/V)-1 as U/V - U/(V+1) = U/(V(V+1)) < 1 (U < b V, V >= b). This holds for any v1 > 0; q* = q+1 needs the lower blocks to matter (probability about 1/v1). q* is obtained from

q0 = round(u0*10^9/Vp) + round(u1d/Vp),

where Vp = 10 v1 + first digit of v2 (9 digits) and u1d = 10 u1 + first digit of u2 (eTeX's a*b/c has a 64 bit intermediate product), clamped to [0, b-1]; then T = U - q0 V is computed exactly as T = a b + c with 0 <= c < b, by splitting q0 and v1, v2 in 4 digit halves, and corrected:

while T < 0: q0 -= 1, T += V;

while T >= V and q0 < b-1: q0 += 1, T -= V.

(|q0 - U/V| < 2.1, so at most three corrections.)

If fin < 0 after the pass, B' is added back (\CALX_ndab) and q decreases.

The quotient blocks are left behind in a \numexpr chain (xintcore's technique, value 10^8+q emitted by the enclosing \numexpr), MS-first, and reversed once at the end.

```
\def\CALX_nudiv #1#2#3#4{%
    \ifnum#2<#4 \xint_dothis\CALX_nudiv_lt\fi
    \xint_orthat{\CALX_nudiv_sel #4.}#1{#2}{#3}{#4}%
}%
\def\CALX_nudiv_sel #1.{%
    \ifcase#1
    \or  \expandafter\CALX_nudiv_one
    \or  \expandafter\CALX_nudiv_two
    \or  \expandafter\CALX_nudiv_three
    \or  \expandafter\CALX_nudiv_four
    \else\expandafter\CALX_nudiv_gen
    \fi
}%
\def\CALX_nudiv_lt #1#2#3#4#5{#1{0}{}{0}{#31;!}}%
```

One block divisor.

```
\def\CALX_nudiv_one #1#2#3#4#5{\CALX_nudiv_oa #1{#2}#5{#3}}%
\def\CALX_nudiv_oa #1#21#3!{%
    \ifnum#2=\xint_c_i \xint_dothis\CALX_nudiv_oo\fi
    \ifnum#3<45250 \xint_dothis\CALX_nudiv_otd\fi
    \xint_orthat\CALX_nudiv_ob #1{#2}{#3}%
}%
\def\CALX_nudiv_oo #1#2#3#4{\CALX_nudiv_oo_a #1{#3}#4}%
```

```
\def\CALX_nudiv_oo_a #1#21#3!{%
    \expandafter\CALX_nudiv_oo_b
    \the\numexpr(#3+#2/\xint_c_ii)/#2-\xint_c_i\xint:#3\xint:#2\xint:#1%
}%
\def\CALX_nudiv_oo_b #1\xint:#2\xint:#3\xint:{%
    \expandafter\CALX_nudiv_oo_c
    \the\numexpr\xint_c_e_viii+#1\expandafter\xint:%
    \the\numexpr\xint_c_e_viii+#2-#1*#3\xint:%
}%
\def\CALX_nudiv_oo_c 1#1\xint:#2\xint:#3{%
    \ifnum#1=\xint_c_ \xint_dothis{#3{0}{}}\fi
    \xint_orthat{#3{1}{1#1!}}{0}{#2!1;!}%
}%
\def\CALX_nudiv_ob #1#2#3#4{%
    \expandafter\CALX_nudiv_oc
    \romannumeral0\CALX_rev_nounsep {}#4\R!\R!\R!\R!\R!\R!\R!\W%
        \xint:{#3}{#2}#1%
}%
\def\CALX_nudiv_oc 1#1!#2\xint:#3#4{%
    \expandafter\CALX_nudiv_od
    \the\numexpr\expandafter\CALX_smalldivx_a\the\numexpr#3/\xint_c_ii%
        \xint:1#3!1#1!#21;!%
    {\ifnum#1<#3 -\xint_c_i\fi+#4}%
}%
\def\CALX_nudiv_od #11!;!#2!#3{%
    \expandafter\CALX_nudiv_oe
    \romannumeral0\CALX_rev_nounsep {}#1\R!\R!\R!\R!\R!\R!\R!\W%
        \xint:{#3}{#2}%
}%
\def\CALX_nudiv_oe #1\xint:#2#3{%
    \expandafter\CALX_nudiv_of
    \the\numexpr\xint_c_e_viii+#3\expandafter\xint:\the\numexpr#2\xint:{#1}%
}%
\def\CALX_nudiv_of #1\xint:#2\xint:#3#4{#4{#2}{#3}{0}{#1!1;!}}%
```

General path. \CALX_nudiv_gs computes s from the top block of B.

```
\def\CALX_nudiv_gen #1#2#3#4#5{%
    \expandafter\CALX_nudiv_ga\the\numexpr
    \CALX_nskip\CALX_nudiv_gs #4-\xint_c_i.#5\xint:\xint:{#5}{#2}{#3}{#4}#1%
}%
\def\CALX_nudiv_gs 1#1!\xint:{\CALX_nlz{#1}}%
\def\CALX_nudiv_pow #1{\csname CALX_nudiv_pow\the\numexpr#1\endcsname}%
\expandafter\def\csname CALX_nudiv_pow0\endcsname{100000001}%
\expandafter\def\csname CALX_nudiv_pow1\endcsname{100000010}%
\expandafter\def\csname CALX_nudiv_pow2\endcsname{100000100}%
\expandafter\def\csname CALX_nudiv_pow3\endcsname{100001000}%
\expandafter\def\csname CALX_nudiv_pow4\endcsname{100010000}%
\expandafter\def\csname CALX_nudiv_pow5\endcsname{100100000}%
\expandafter\def\csname CALX_nudiv_pow6\endcsname{101000000}%
\expandafter\def\csname CALX_nudiv_pow7\endcsname{110000000}%
```

#1=s, #2=B, #3=na, #4=A, #5=nb, #6=cont. Produces
\CALX_nudiv_gc <A''>1;!\xint:<B'>1;!\xint:{na}{nb}{s}<cont>

```
\def\CALX_nudiv_ga #1\xint:{%
```

```
    \ifnum#1=\xint_c_ \xint_dothis\CALX_nudiv_gz\fi
    \xint_orthat\CALX_nudiv_gm {#1}%
}%
\def\CALX_nudiv_gz #1#2#3#4#5{\CALX_nudiv_gc #4100000000!1;!\xint:#21;!%
    \xint:{#3}{#5}{0}}%
\def\CALX_nudiv_gm #1#2{%
    \expandafter\CALX_nudiv_gb\the\numexpr
    \expandafter\CALX_smallmul\the\numexpr\CALX_nudiv_pow{#1}!#21;!\xint:%
    {#1}%
}%
\def\CALX_nudiv_gb #11;!\xint:#2#3#4#5{%
    \expandafter\CALX_nudiv_gc\the\numexpr
    \expandafter\CALX_smallmul\the\numexpr\CALX_nudiv_pow{#2}!#4100000000!1;!%
    \xint:#11;!\xint:{#3}{#5}{#2}%
}%
```

#1=A'' (na+1 blocks), #2=B', #3=na, #4=nb, #5=s, then <cont>. The constants C = {k}{v1h}{v1l}{v2h}{v2l}{Vp}, k = 2nb-4, are obtained from the two top blocks of a copy of B'.

```
\def\CALX_nudiv_gc #11;!\xint:#21;!\xint:#3#4#5{%
    \CALX_nskip\CALX_nudiv_gv #4-\xint_c_ii.#21;!%
        \xint:{#4}{#3-#4}{#5}{#1}{#2}%
}%
\def\CALX_nudiv_gv 1#1#2#3#4#5!1#6#7#8#9{\CALX_nudiv_gu %
    {#1#2#3#4}{#5}{#6#7#8#9}{#1}}%
```

#1=v2h, #2=v2l, #3=v1h, #4=first digit of v2, #5=v1l, #6=nb, #7=m (number of low blocks of A'' to reverse, expression), #8=s, #9=A'', then {B'}.
C = {k}{v1h}{v1l}{v2h}{v2l}{Vp} with k an expression (only used through \numexpr).

```
\def\CALX_nudiv_gu #1#2#3#4#5!1;!\xint:#6#7#8#9{%
    \expandafter\CALX_nrevk\the\numexpr#7.\CALX_nudiv_gw{}#91;!\xint:%
    {{\xint_c_ii*#6-\xint_c_iv}{#3}{#5}{#1}{#2}{#3#5#4}}{#7+\xint_c_i}{#8}%
}%
```

29.14.2. \CALX_nrevk

\CALX_nrevk <k>.<cont>{acc}<blocks>: reverses k blocks onto acc, then <cont>{acc'}<remaining blocks>. Six blocks per step.

```
\def\CALX_nrevk #1.{%
    \ifnum#1>\xint_c_v\expandafter\CALX_nrevk_g\else\expandafter\CALX_nrevk_b%
        \fi #1.%
}%
\def\CALX_nrevk_g #1.#2#3#4!#5!#6!#7!#8!#9!%
    {\expandafter\CALX_nrevk\the\numexpr#1-%
        \xint_c_vi.#2{#9!#8!#7!#6!#5!#4!#3}}%
\def\CALX_nrevk_b #1.{%
    \ifcase#1
         \expandafter\CALX_nrevk_z
    \or  \expandafter\CALX_nrevk_i
    \or  \expandafter\CALX_nrevk_ii
    \or  \expandafter\CALX_nrevk_iii
    \or  \expandafter\CALX_nrevk_iv
    \else\expandafter\CALX_nrevk_v
    \fi
}%
```

```
\def\CALX_nrevk_z   #1{#1}%
\def\CALX_nrevk_i   #1#2#3!{#1{#3!#2}}%
\def\CALX_nrevk_ii  #1#2#3!#4!{#1{#4!#3!#2}}%
\def\CALX_nrevk_iii #1#2#3!#4!#5!{#1{#5!#4!#3!#2}}%
\def\CALX_nrevk_iv  #1#2#3!#4!#5!#6!{#1{#6!#5!#4!#3!#2}}%
\def\CALX_nrevk_v   #1#2#3!#4!#5!#6!#7!{#1{#7!#6!#5!#4!#3!#2}}%
```

#1=R (MS-first low blocks of A''), #2=window (LS-first), #3=C, #4=steps, #5=s, #6=B'. The interleaving \CALX_nilv takes one block of each list per step (it regrabs the rest of the window).

```
\def\CALX_nudiv_gw #1#21;!\xint:#3#4#5#6{%
    \expandafter\CALX_nudiv_gl\the\numexpr\CALX_nilv #21;!\xint:#61;;;;;!%
    {#3}{#11;!}{#4}{#5}%
}%
```

29.14.3. \CALX_nilv

```
\def\CALX_nilv 1#1!#2\xint:1#3#4!{%
    \xint_gob_til_sc #3\CALX_nilv_e;%
    1#1\expandafter!\the\numexpr1#3#4\expandafter!\the\numexpr\CALX_nilv #2%
        \xint:%
}%
\def\CALX_nilv_e;1#1\expandafter!\the\numexpr1#2\xint:{1#1\relax!1;;;;;!}%
\def\CALX_nudiv_gl #1;;;;;!#2#3{%
    \expandafter\CALX_nudiv_x\the\numexpr\CALX_nds {#2}#1;;;;;!#3%
}%
```

The main loop. Input stream: \CALX_nds {C}<L>;;;;;!<R>1;!... (here <L> lacks its final ;;;;;! which is written explicitly).

\CALX_nds_t receives the top of a copy of L after k = 2nb-4 blocks are skipped.

```
\def\CALX_ndc_k  #1#2#3#4#5#6{#1}%
\def\CALX_ndc_ah #1#2#3#4#5#6{#2}%
\def\CALX_ndc_al #1#2#3#4#5#6{#3}%
\def\CALX_ndc_bh #1#2#3#4#5#6{#4}%
\def\CALX_ndc_bl #1#2#3#4#5#6{#5}%
\def\CALX_ndc_a  #1#2#3#4#5#6{#2#3}%
\def\CALX_ndc_b  #1#2#3#4#5#6{#4#5}%
\def\CALX_ndc_vp #1#2#3#4#5#6{#6}%
```

29.14.4. \CALX_nds

```
\def\CALX_nds #1#2;;;;;!{%
    \CALX_nskip\CALX_nds_t\CALX_ndc_k#1.#2;;;;;!\xint:{#1}#2;;;;;!%
}%
\def\CALX_nds_t 1#1#2!1#3!1#4!1#5!1#6!1;;;;;!\xint:#7{%
    \expandafter\CALX_nds_r
    \the\numexpr#6*\xint_c_e_ix/\CALX_ndc_vp#7+#4#1/\CALX_ndc_vp#7\xint:%
    {#6}{#4}{#1#2}{#7}%
}%
\def\CALX_nds_r #1\xint:{%
    \ifnum#1>\xint_c_e_viii_mone \CALX_nds_rm\fi
    \expandafter\CALX_nds_s\the\numexpr\xint_c_e_viii+#1\xint:%
}%
```

#1#2#3#4=qh (4 digits), #5=ql, #6=u0, #7=u1, #8=u2, #9=C.

```
  Splitting M1 = qh v1l + ql v1h and M2 = qh v2l + ql v2h as M = 10^4 Mh + Ml, with Mh = floor(M/10^4)
= (M+5000)/10^4-1:
   q0 v1 = (qh v1h + M1h) b + 10^4 M1l + ql v1l, idem for v2;
   T = U - q0 (v1 b + v2) = Ax b + Bx with
   Bx = u2 - 10^4 M2l - ql v2l                        in (-2b, b),
   Ax = (u0 - qh v1h - M1h) b + u1 - 10^4 M1l - ql v1l - qh v2h - M2h.
3b+Bx is in (b, 4b) and has nine digits e<c>, so that T = a b + c with a = Ax+e-3, 0 <= c < b.
```

```
\def\CALX_nds_rm\fi#1\xint:{\fi\CALX_nds_s 199999999\xint:}%
\def\CALX_nds_s 1#1#2#3#4#5\xint:#6#7#8#9{%
    \expandafter\CALX_nds_u\the\numexpr
    (#1#2#3#4*\CALX_ndc_al#9+#5*\CALX_ndc_ah#9+5000)/\xint_c_e_iv-\xint_c_i
    \expandafter\xint:\the\numexpr
    (#1#2#3#4*\CALX_ndc_bl#9+#5*\CALX_ndc_bh#9+5000)/\xint_c_e_iv-\xint_c_i
    \xint:{#1#2#3#4}{#5}{#6}{#7}{#8}{#9}%
}%
\def\CALX_nds_u #1\xint:#2\xint:#3#4#5#6#7#8{%
    \expandafter\CALX_nds_w\the\numexpr
    \xint_c_iii*\xint_c_e_viii+#7-#4*\CALX_ndc_bl#8%
    -(#3*\CALX_ndc_bl#8+#4*\CALX_ndc_bh#8-\xint_c_e_iv*#2)*\xint_c_e_iv
    \expandafter\xint:\the\numexpr
    (#5-#3*\CALX_ndc_ah#8-#1)*\xint_c_e_viii+#6%
    -(#3*\CALX_ndc_al#8+#4*\CALX_ndc_ah#8-\xint_c_e_iv*#1)*\xint_c_e_iv
    -#4*\CALX_ndc_al#8-#3*\CALX_ndc_bh#8-#2%
    \xint:{#3#4}{#8}%
}%
\def\CALX_nds_w #1#2\xint:#3\xint:{%
    \expandafter\CALX_nds_c\the\numexpr#3+#1-%
        \xint_c_iii.{#2}%
}%
```

```
Correction loop:  #1=a, #2=c (T = a b + c, 0 <= c < b), #3=q, #4=C. T < 0 iff a < 0.  T >= V iff 2(a-v1)
+ (0 if c >= v2 else -1) >= 0.
```

```
\def\CALX_nds_c #1.#2#3#4{%
    \ifnum#1<\xint_c_ \xint_dothis\CALX_nds_cd\fi
    \ifnum\numexpr\xint_c_ii*(#1-\CALX_ndc_a#4)\ifnum#2<\CALX_ndc_b#4 -%
        \xint_c_i\fi
        <\xint_c_ \xint_dothis\CALX_nds_g\fi
    \ifnum#3=\xint_c_e_viii_mone \xint_dothis\CALX_nds_g\fi
    \xint_orthat\CALX_nds_ci {#1}{#2}{#3}{#4}%
}%
\def\CALX_nds_cd #1#2#3#4{%
    \expandafter\CALX_nds_ce\the\numexpr\xint_c_e_viii+#2+\CALX_ndc_b#4\xint:%
    {#1+\CALX_ndc_a#4}{#3-\xint_c_i}{#4}%
}%
\def\CALX_nds_ce #1#2\xint:#3#4{%
    \expandafter\CALX_nds_cf\the\numexpr#3+#1-\xint_c_i\expandafter\xint:%
    \the\numexpr#4\xint:{#2}%
}%
\def\CALX_nds_cf #1\xint:#2\xint:#3{\CALX_nds_c #1.{#3}{#2}}%
\def\CALX_nds_ci #1#2#3#4{%
    \expandafter\CALX_nds_ce\the\numexpr
    \xint_c_ii*\xint_c_e_viii+#2-\CALX_ndc_b#4\xint:%
    {#1-\CALX_ndc_a#4-\xint_c_i}{#3+\xint_c_i}{#4}%
```

```
}%
```

```
\CALX_nds_g #1=a, #2=c, #3=q, #4=C; then <L>;;;;;!<R>1;!.
  The multiply-subtract pass runs on L.
```

```
\def\CALX_nds_g #1#2#3#4#5;;;;;!{%
    \expandafter\CALX_nds_h\the\numexpr\xint_c_e_viii+#3\xint:{#4}#5;;;;;!%
}%
\def\CALX_nds_h 1#1#2#3#4#5\xint:#6#7;;;;;!{%
    \expandafter\CALX_nds_p\the\numexpr
    \CALX_ndms_a 0\xint:#1#2#3#4\xint:#5!#7;;;;;!{#1#2#3#4#5}{#6}%
}%
\def\CALX_nds_p #1;;;;;!#2{%
    \ifnum\numexpr#2<\xint_c_ \xint_dothis\CALX_nds_pa\fi
    \xint_orthat\CALX_nds_pb #1;;;;;!%
}%
\def\CALX_nds_pa #1;;;;;!{%
    \expandafter\CALX_nds_pc\the\numexpr\CALX_ndab 0\xint:#1;;;;;!%
}%
\def\CALX_nds_pc #1;;;;;!#2{\CALX_nds_pb #1;;;;;!{#2-\xint_c_i}}%
```

```
#1=new list without ;;;;;!, #2=q (expression), #3=C, then the first block of R. Emission of 10^8+q
into the enclosing \numexpr.
```

```
\def\CALX_nds_pb #1;;;;;!#2#31#4#5!{%
    \xint_gob_til_sc #4\CALX_nds_end;%
    \xint_c_e_viii+#2\expandafter!\the\numexpr\CALX_nds {#3}1#4#5!#1;;;;;!%
}%
\def\CALX_nds_end;\xint_c_e_viii+#1\expandafter!\the\numexpr\CALX_nds %
    #21#3!#4;;;;;!%
    {\xint_c_e_viii+#1\relax!\xint:{#4}}%
```

```
Multiply-subtract pass.  State <cexpr>\xint:<qh>\xint:<ql>! where cexpr is an expression for the
borrow c (0 <= c < 10^8), then the pairs 1<w>!1<bh4><bl4>!.  With M = qh bl + ql bh = 10^4 Mh + Ml
(10^9+M has ten digits:  1, Mh on five, Ml on four):
   w - c - q b = X - (qh bh + Mh) 10^8,  X = w - c - ql bl - 10^4 Ml,
X in (-3 10^8, 10^8); 4 10^8 + X has nine digits e<r>, r is the block, and the new borrow is qh bh +
Mh + 4 - e.  At the marker Z, the expression w_nb - c (0 or -1) is left in braces.
```

```
\def\CALX_ndms_a #1\xint:#2\xint:#3!1#4!1#5#6#7#8#9!{%
    \xint_gob_til_sc #5\CALX_ndms_e;%
    1#5#6#7#8#9\expandafter!\the\numexpr\expandafter\CALX_ndms_b
    \the\numexpr\xint_c_e_ix+#2*#9+#3*#5#6#7#8\xint:%
    #4-(#1)\xint:#3*#9\xint:#2*#5#6#7#8\xint:#2\xint:#3!%
}%
\def\CALX_ndms_e;#1\xint:#2\xint:#3\xint:#4\xint:#5\xint:#6!{1%
    \relax;;;;;!{#2}}%
\def\CALX_ndms_b 1#1#2#3#4#5#6#7#8#9\xint:{\CALX_ndms_c #1#2#3#4#5%
    \xint:#6#7#8#9\xint:}%
\def\CALX_ndms_c #1\xint:#2\xint:#3\xint:#4\xint:#5\xint:{%
    \expandafter\CALX_ndms_d
    \the\numexpr\xint_c_iv*\xint_c_e_viii+#3-#4-#2*\xint_c_e_iv\xint:#5+#1%
        \xint:%
}%
\def\CALX_ndms_d #1#2\xint:#3\xint:%
    {1#2\expandafter!\the\numexpr\CALX_ndms_a #3+\xint_c_iv-#1\xint:}%
```

29.14.5. \CALX_ndab

Add back: on 1b_0!1r_0!...1b_{nb-1}!1r_{nb-1}!1;;;;;!, r += B' (final carry dropped), b's copied.

```
\def\CALX_ndab #1\xint:1#2#3!{\xint_gob_til_sc #2\CALX_ndab_e;\CALX_ndab_c #1%
    \xint:#2#3!}%
\def\CALX_ndab_e;\CALX_ndab_c #1\xint:#2!{1\relax;;;;;!}%
\def\CALX_ndab_c #1\xint:#2!1#3!{%
    1#2\expandafter!\the\numexpr\expandafter\CALX_ndab_d
    \the\numexpr\xint_c_e_viii+#3+#2+#1\xint:%
}%
\def\CALX_ndab_d #1#2\xint:{1#2\expandafter!\the\numexpr\CALX_ndab #1-%
    \xint_c_i\xint:}%
```

End: <Q MS-first blocks>\xint:{<last list>}{steps}{s}<cont>. The top quotient block may be zero (only the top one).

```
\def\CALX_nudiv_x 1#1!#2\xint:#3#4{%
    \ifcase\numexpr\ifnum#1=\xint_c_
                      \ifnum\numexpr#4=\xint_c_i \xint_c_ii\else\xint_c_i\fi
                    \else\xint_c_\fi\relax
         \expandafter\CALX_nudiv_x_n
    \or  \expandafter\CALX_nudiv_x_z
    \else\expandafter\CALX_nudiv_x_zero
    \fi {#1}{#2}{#3}{#4}%
}%
\def\CALX_nudiv_x_zero #1#2#3#4{\CALX_nudiv_y \xint:{0}{#3}}%
\def\CALX_nudiv_x_z #1#2#3#4{%
    \expandafter\CALX_nudiv_y
    \romannumeral0\CALX_rev_nounsep {}#2\R!\R!\R!\R!\R!\R!\R!\W\xint:{#4-%
        \xint_c_i}{#3}%
}%
\def\CALX_nudiv_x_n #1#2#3#4{%
    \expandafter\CALX_nudiv_y
    \romannumeral0\CALX_rev_nounsep {}1#1!#2\R!\R!\R!\R!\R!\R!\R!\W%
        \xint:{#4}{#3}%
}%
\def\CALX_nudiv_y #1\xint:#2#3{%
    \expandafter\CALX_nudiv_z\the\numexpr#2\expandafter.%
    \the\numexpr\CALX_ndrx #3;;;;;!1;!.{#1}%
}%
\def\CALX_nudiv_z #1.#2.#3#4#5{#5{#1}{#3}{#4}{#2}}%
```

29.14.6. \CALX_ndrx

```
\def\CALX_ndrx 1#1#2!1#3!{%
    \xint_gob_til_sc #1\CALX_ndrx_e;1#3\expandafter!\the\numexpr\CALX_ndrx
}%
\def\CALX_ndrx_e;1#1\CALX_ndrx{1\relax;!}%
```

\CALX_nudiv_rem {s}{R'}: native positive R (or N+0.). For s>0, R = 10^(8-s) R' / 10^8: \CALX_smallmul (from xintcore) then the low block (zero) is dropped. Then \CALX_nnorm counts and strips zero top blocks.

```
\def\CALX_nudiv_rem #1#2{%
    \ifnum#1=\xint_c_ \xint_dothis{\CALX_nnorm +}\fi
```

```
    \xint_orthat{\expandafter\CALX_nudiv_rem_a\the\numexpr
     \expandafter\CALX_smallmul\the\numexpr\CALX_nudiv_pow{%
         \xint_c_viii-#1}!}#2%
}%
\def\CALX_nudiv_rem_a 1#1!{\CALX_nnorm +}%
```

Two-block divisor (nb = 2). No normalization: A'' = A with a zero block appended (na+1 blocks), reversed once (MS-first); s = 0. With s the number of leading zeros of v1 (here s only scales the estimate), M = 10^(s+1) and Vp = v1 M + floor(v2/10^(7-s)) (nine digits), the first guess is
q0 = round((u0 M) 10^8/Vp) + round(u1 M/Vp) + round(round(u2 M/10^8)/Vp).
The window is U = u0 b^2 + u1 b + u2 exactly and the divisor V = v1 b + v2 exactly, so q = floor(U/V) is the quotient block (no add-back) and T = U - qV = a b + c is the next (u0, u1).
Loop: \CALX_ndtw {Vp}{M}{v1h}{v1l}{v2h}{v2l}{u0}{u1}1<u2>!...1;! with u1 (and u2) on eight digits. qh and ql by \numexpr, then
M1 = qh v1l + ql v1h, M2 = qh v2l + ql v2h, Mh = (M+5000)/10^4-1,
Bx = 3b + u2 - ql v2l - 10^4 (M2 - 10^4 M2h),
Ax = (u0 - qh v1h - M1h) b + u1 - 10^4 (M1 - 10^4 M1h) - ql v1l
- qh v2h - M2h,
T = Ax b + Bx, 3b+Bx has nine digits e<c>, a = Ax+e-3, then the correction loop \CALX_ndtw_l (same tests as \CALX_nds_c).
Quotient blocks are emitted MS-first into the enclosing \numexpr chain, which ends with 1;!.

```
\def\CALX_nudiv_two #1#2#3#4#5{\CALX_nudiv_tw #5{#2}{#3}#1}%
\def\CALX_nudiv_tw 1#1#2#3#4#5!1#6#7#8#9{\CALX_nudiv_tu %
    {#1#2#3#4}{#5}{#6#7#8#9}}%
\def\CALX_nudiv_tu #1#2#3#4!{%
    \expandafter\CALX_nudiv_tv
    \expanded{\CALX_nudiv_mp{\CALX_nudiv_gs 1#3#4!\xint:}}{#1}{#2}{#3}{#4}%
}%
```

\CALX_nudiv_mp{s} gives M.P. with M = 10^(s+1), P = 10^(7-s), so that Vp = v1 M + floor(v2/P) has nine digits (first guess of q0, see the two-block divisor above).

```
\def\CALX_nudiv_mp #1{\csname CALX_nudiv_mp\the\numexpr#1\endcsname}%
\expandafter\def\csname CALX_nudiv_mp0\endcsname{10.10000000.}%
\expandafter\def\csname CALX_nudiv_mp1\endcsname{100.1000000.}%
\expandafter\def\csname CALX_nudiv_mp2\endcsname{1000.100000.}%
\expandafter\def\csname CALX_nudiv_mp3\endcsname{10000.10000.}%
\expandafter\def\csname CALX_nudiv_mp4\endcsname{100000.1000.}%
\expandafter\def\csname CALX_nudiv_mp5\endcsname{1000000.100.}%
\expandafter\def\csname CALX_nudiv_mp6\endcsname{10000000.10.}%
\expandafter\def\csname CALX_nudiv_mp7\endcsname{100000000.1.}%
\def\CALX_nudiv_tv #1.#2.#3#4#5#6{%
    \expandafter\CALX_nudiv_tt
    \the\numexpr#5#6*#1+(#3#4+#2/\xint_c_ii)/#2-\xint_c_i\xint:%
    {#1}{#5}{#6}{#3}{#4}%
}%
\def\CALX_nudiv_tt #1\xint:#2#3#4#5#6#7#8{%
    \expandafter\CALX_nudiv_tf
    \romannumeral0\CALX_rev_nounsep {}#8\R!\R!\R!\R!\R!\R!\R!\W
    \xint:{{#1}{#2}{#3}{#4}{#5}{#6}}{#7}%
}%
```

#1, #2 = two top blocks of A, #3 = the others (MS-first), #4 = K = {Vp}{M}{v1h}{v1l}{v2h}{v2l}, #5 = na. If (a1, a2) < (v1, v2) the window starts at (a1, a2, a3) (na-2 steps; if na = 2, Q = 0 and R = A), else at (0, a1, a2) (na-1 steps): in both cases the top quotient block is not 0.

```
\def\CALX_nudiv_tcmp #1#2#3#4#5#6#7#8{%
    \ifnum#1<#5#6 -\xint_c_i\else\ifnum#1>#5#6 \xint_c_\else
    \ifnum#2<#7#8 -\xint_c_i\else\xint_c_\fi\fi\fi
}%
\def\CALX_nudiv_tf 1#1!1#2!#3\xint:#4#5{%
    \ifnum\CALX_nudiv_tcmp{#1}{#2}#4<\xint_c_
       \ifnum#5=\xint_c_ii \CALX_nudiv_tq\fi
       \expandafter\CALX_nudiv_tn
    \else
       \expandafter\CALX_nudiv_tp
    \fi {#1}{#2}{#3}{#4}{#5}%
}%
\def\CALX_nudiv_tq\fi\expandafter\CALX_nudiv_tn\else\expandafter%
    \CALX_nudiv_tp\fi
    #1#2#3#4#5#6{\fi\fi#6{0}{}{0}{1#2!1#1!1;!}}%
\def\CALX_nudiv_tn #1#2#3#4#5{%
    \expandafter\CALX_ndtw_z\the\numexpr\CALX_ndtw #4{#1}{#2}#31;!{#5-%
        \xint_c_i}%
}%
\def\CALX_nudiv_tp #1#2#3#4{%
    \expandafter\CALX_ndtw_z\the\numexpr\CALX_ndtw #4{0}{#1}1#2!#31;!%
}%
```

29.14.7. \CALX_ndtw

```
\def\CALX_ndtw #1#2#3#4#5#6#7#81#9!{%
    \xint_gob_til_sc #9\CALX_ndtw_end;%
    \expandafter\CALX_ndtw_q
    \the\numexpr(#7*#2)*\xint_c_e_viii/#1+#8*#2/#1+(#9*#2/\xint_c_e_viii)/#1%
        \xint:%
    {#3}{#4}{#5}{#6}{#7}{#8}{#9}{{#1}{#2}}%
}%
\def\CALX_ndtw_end;\expandafter\CALX_ndtw_q\the\numexpr#1%
    \xint:#2#3#4#5#6#7#8#9%
    {1\relax;!\xint:{#6}{#7}}%
\def\CALX_ndtw_q #1\xint:%
    {\ifnum#1>\xint_c_e_viii_mone \CALX_ndtw_qm\fi\CALX_ndtw_h #1\xint:}%
\def\CALX_ndtw_qm\fi#1\xint:{\fi\CALX_ndtw_h 99999999\xint:}%
\def\CALX_ndtw_h #1\xint:{%
    \expandafter\CALX_ndtw_x
    \the\numexpr(#1+5000)/\xint_c_e_iv-\xint_c_i\expandafter\xint:%
    \the\numexpr#1-\xint_c_e_iv*((#1+5000)/\xint_c_e_iv-\xint_c_i)\xint:%
}%
\def\CALX_ndtw_x #1\xint:#2\xint:#3#4#5#6#7#8#9{%
    \expandafter\CALX_ndtw_w\the\numexpr
    \xint_c_iii*\xint_c_e_viii+#9-#2*#6%
    -(#1*#6+#2*#5-\xint_c_e_iv*((#1*#6+#2*#5+5000)/\xint_c_e_iv-\xint_c_i))%
     *\xint_c_e_iv
    \expandafter\xint:\the\numexpr
    (#7-#1*#3-(#1*#4+#2*#3+5000)/\xint_c_e_iv+\xint_c_i)*\xint_c_e_viii+#8%
    -(#1*#4+#2*#3-\xint_c_e_iv*((#1*#4+#2*#3+5000)/\xint_c_e_iv-\xint_c_i))%
     *\xint_c_e_iv
    -#2*#4-#1*#5-(#1*#6+#2*#5+5000)/\xint_c_e_iv+\xint_c_i
```

```
    \xint:{#1*\xint_c_e_iv+#2}{#3}{#4}{#5}{#6}%
}%
\def\CALX_ndtw_w #1#2\xint:#3\xint:{%
    \expandafter\CALX_ndtw_l\the\numexpr#3+#1-%
        \xint_c_iii.{#2}%
}%
\def\CALX_ndtw_l #1.#2#3#4#5#6#7{%
    \ifnum#1<\xint_c_ \xint_dothis\CALX_ndtw_d\fi
    \ifnum\numexpr\xint_c_ii*(#1-#4#5)\ifnum#2<#6#7 -\xint_c_i\fi
        <\xint_c_ \xint_dothis\CALX_ndtw_g\fi
    \ifnum\numexpr#3=\xint_c_e_viii_mone \xint_dothis\CALX_ndtw_g\fi
    \xint_orthat\CALX_ndtw_i {#1}{#2}{#3}{#4}{#5}{#6}{#7}%
}%
\def\CALX_ndtw_d #1#2#3#4#5#6#7{%
    \expandafter\CALX_ndtw_e\the\numexpr\xint_c_e_viii+#2+#6#7\xint:%
    {#1+#4#5}{#3-\xint_c_i}{#4}{#5}{#6}{#7}%
}%
\def\CALX_ndtw_i #1#2#3#4#5#6#7{%
    \expandafter\CALX_ndtw_e\the\numexpr\xint_c_ii*\xint_c_e_viii+#2-#6#7%
        \xint:%
    {#1-#4#5-\xint_c_i}{#3+\xint_c_i}{#4}{#5}{#6}{#7}%
}%
\def\CALX_ndtw_e #1#2\xint:#3#4{%
    \expandafter\CALX_ndtw_f\the\numexpr#3+#1-\xint_c_i\expandafter\xint:%
    \the\numexpr#4\xint:{#2}%
}%
\def\CALX_ndtw_f #1\xint:#2\xint:#3{\CALX_ndtw_l #1.{#3}{#2}}%
\def\CALX_ndtw_g #1#2#3#4#5#6#7#8{%
    \xint_c_e_viii+#3\expandafter!\the\numexpr\CALX_ndtw %
        #8{#4}{#5}{#6}{#7}{#1}{#2}%
}%
```

Exit: <Q MS-first>1;!\xint:{u0}{u1}{na}{s}<cont>; R' = 1u1!1u0!1;!.
\CALX_nudiv_qx <Q MS-first>\xint:{R'}{steps}{s}<cont> drops a zero top block, reverses Q and delivers <cont>{nq}{Q}{s}{R'}.

```
\def\CALX_ndtw_z #11;!\xint:#2#3#4{%
    \expandafter\CALX_ndtw_zz\the\numexpr\xint_c_e_viii+#2\xint:{#3}{#4}{#1}%
}%
\def\CALX_ndtw_zz #1\xint:#2#3#4{\CALX_nudiv_qx #4\xint:{1#2!#1!1;!}{#3-%
    \xint_c_i}{0}}%
\def\CALX_nudiv_qx 1#1!#2\xint:#3#4{%
    \ifcase\numexpr\ifnum#1=\xint_c_
                     \ifnum\numexpr#4=\xint_c_i \xint_c_ii\else\xint_c_i\fi
                   \else\xint_c_\fi\relax
         \expandafter\CALX_nudiv_qx_n
    \or  \expandafter\CALX_nudiv_qx_z
    \else\expandafter\CALX_nudiv_qx_zero
    \fi {#1}{#2}{#3}{#4}%
}%
\def\CALX_nudiv_qx_zero #1#2#3#4{\CALX_nudiv_qy \xint:{0}{#3}}%
\def\CALX_nudiv_qx_z #1#2#3#4{%
    \expandafter\CALX_nudiv_qy
    \romannumeral0\CALX_rev_nounsep {}#2\R!\R!\R!\R!\R!\R!\R!\W\xint:{#4-%
```

```
        \xint_c_i}{#3}%
}%
\def\CALX_nudiv_qx_n #1#2#3#4{%
    \expandafter\CALX_nudiv_qy
    \romannumeral0\CALX_rev_nounsep {}1#1!#2\R!\R!\R!\R!\R!\R!\R!\W%
        \xint:{#4}{#3}%
}%
\def\CALX_nudiv_qy #1\xint:#2#3#4#5{%
    \expandafter#5\expandafter{\the\numexpr#2}{#1}{#4}{#3}%
}%
```

Tiny one-block divisor d <= 45249. With 10^8 = Bq d + Bm (0 <= Bm < d) and incoming remainder r < d, a block a gives

y = r Bm + a, q = r Bq + floor(y/d), r' = y - d floor(y/d),

exact since r 10^8 + a = r Bq d + y; y + round(d/2) < 2^31 for d <= 45249. floor(y/d) = (y + round(d/2))/d - 1. One explicit \numexpr per block plus the emission (10^8 + q is evaluated by the enclosing \numexpr chain).

\CALX_ndts <r>\xint:<d>\xint:<round(d/2)>\xint:<Bq>\xint:<Bm>!
<MS-first blocks>1;!

\CALX_ndts_k <d>.<macro> expands to <macro>0\xint:d\xint:dh\xint:Bq\xint:Bm!

```
\def\CALX_ndts_k #1.{%
    \expandafter\CALX_ndts_ka\the\numexpr(\xint_c_e_viii+#1/\xint_c_ii)/#1-%
        \xint_c_i
    \xint:#1\xint:%
}%
\def\CALX_ndts_ka #1\xint:#2\xint:{%
    \expandafter\CALX_ndts_kb\the\numexpr\xint_c_e_viii-#1*#2\expandafter%
        \xint:%
    \the\numexpr#2/\xint_c_ii\xint:#1\xint:#2\xint:%
}%
\def\CALX_ndts_kb #1\xint:#2\xint:#3\xint:#4\xint:#5{#50\xint:#4\xint:#2%
    \xint:#3\xint:#1!}%
\def\CALX_nudiv_ot #1#2#3#4{%
    \expandafter\CALX_ndts_s
    \romannumeral0\CALX_rev_nounsep {}#4\R!\R!\R!\R!\R!\R!\R!\W%
        \xint:{#3}{#2}#1%
}%
\def\CALX_ndts_s #1\xint:#2{%
    \expandafter\CALX_ndts_z\the\numexpr\CALX_ndts_k #2.\CALX_ndts #11;!%
}%
```

29.14.8. \CALX_ndts

```
\def\CALX_ndts #1\xint:#2\xint:#3\xint:#4\xint:#5!1#6#7!{%
    \xint_gob_til_sc #6\CALX_ndts_end;%
    \expandafter\CALX_ndts_b\the\numexpr(#1)*#5+#6#7\xint:#1\xint:#2\xint:#3%
        \xint:#4\xint:#5!%
}%
\def\CALX_ndts_end;#1\xint:#2\xint:#3!{1\relax;!\xint:{#2}}%
\def\CALX_ndts_b #1\xint:#2\xint:#3\xint:#4\xint:#5\xint:#6!{%
    \xint_c_e_viii+(#2)*#5+(#1+#4)/#3-\xint_c_i
    \expandafter!\the\numexpr\CALX_ndts #1-#3*((#1+#4)/#3-\xint_c_i)\xint:#3%
        \xint:#4\xint:#5\xint:#6!%
```

```
}%
\def\CALX_ndts_z #11;!\xint:#2#3{%
    \expandafter\CALX_ndts_zz\the\numexpr\xint_c_e_viii+(#2)\xint:{#3}{#1}%
}%
\def\CALX_ndts_zz #1\xint:#2#3{\CALX_nudiv_qx #3\xint:{#1!1;!}{#2}{0}}%
```

Unrolled entries for na = 2, 3, 4 (tiny divisor): the MS-first input is written from the parameters and the quotient is assembled directly (the top block may be zero, the others are present).

```
\def\CALX_nudiv_otd #1#2{%
    \ifcase#2
    \or\or  \expandafter\CALX_ndts_ii
    \or     \expandafter\CALX_ndts_iii
    \or     \expandafter\CALX_ndts_iv
    \else   \expandafter\CALX_nudiv_ot
    \fi #1{#2}%
}%
\def\CALX_ndts_ii #1#2#3#4{\CALX_ndts_iia #4{#3}#1}%
\def\CALX_ndts_iia 1#1!1#2!#3{%
    \expandafter\CALX_ndts_iib\the\numexpr\CALX_ndts_k #3.\CALX_ndts %
        1#2!1#1!1;!%
}%
\def\CALX_ndts_iib 1#1!1#2!1;!\xint:#3{%
    \expandafter\CALX_ndts_iic\the\numexpr\xint_c_e_viii+(#3)\xint:{#1}{#2}%
}%
\def\CALX_ndts_iic #1\xint:#2#3#4{%
    \ifnum#2=\xint_c_ \xint_dothis{#4{1}{1#3!}}\fi
    \xint_orthat{#4{2}{1#3!1#2!}}{0}{#1!1;!}%
}%
\def\CALX_ndts_iii #1#2#3#4{\CALX_ndts_iiia #4{#3}#1}%
\def\CALX_ndts_iiia 1#1!1#2!1#3!#4{%
    \expandafter\CALX_ndts_iiib\the\numexpr\CALX_ndts_k #4.\CALX_ndts %
        1#3!1#2!1#1!1;!%
}%
\def\CALX_ndts_iiib 1#1!1#2!1#3!1;!\xint:#4{%
    \expandafter\CALX_ndts_iiic\the\numexpr\xint_c_e_viii+(#4)%
        \xint:{#1}{1#3!1#2!}%
}%
\def\CALX_ndts_iiic #1\xint:#2#3#4{%
    \ifnum#2=\xint_c_ \xint_dothis{#4{2}{#3}}\fi
    \xint_orthat{#4{3}{#31#2!}}{0}{#1!1;!}%
}%
\def\CALX_ndts_iv #1#2#3#4{\CALX_ndts_iva #4{#3}#1}%
\def\CALX_ndts_iva 1#1!1#2!1#3!1#4!#5{%
    \expandafter\CALX_ndts_ivb\the\numexpr\CALX_ndts_k #5.\CALX_ndts %
        1#4!1#3!1#2!1#1!1;!%
}%
\def\CALX_ndts_ivb 1#1!1#2!1#3!1#4!1;!\xint:#5{%
    \expandafter\CALX_ndts_ivc\the\numexpr\xint_c_e_viii+(#5)%
        \xint:{#1}{1#4!1#3!1#2!}%
}%
\def\CALX_ndts_ivc #1\xint:#2#3#4{%
    \ifnum#2=\xint_c_ \xint_dothis{#4{3}{#3}}\fi
    \xint_orthat{#4{4}{#31#2!}}{0}{#1!1;!}%
```

```
}%
```

Three-block divisor (nb = 3). Estimate and correction loop as for nb = 2 (U = u0 u1 u2, V = v1 v2: q is the true quotient block or one more), the macros are duplicated (\CALX_ndth_q ... \CALX_ndth_f), only the exit \CALX_ndth_g differs: with T = a b + c = U - qV, the window minus q B' is

```
   a b^2 + c b + u3 - q v3,  q v3 = (qh v3h + M3h) b + 10^4 M3l + ql v3l.
   X = 3b + u3 - ql v3l - 10^4 M3l   in (b, 4b): digits e1<r2>,
   Y = 3b + c - qh v3h - M3h - 3 + e1  in (b, 4b): digits e2<r1>,
   r0 = a + e2 - 3; if r0 < 0, B' is added back and q decreases.
Loop state: \CALX_ndth {C12}{V3}{u0}{u1}{u2}1<u3>!...1;! C12 = {Vp}{M}{{v1h}{v1l}{v2h}{v2l}},
V3 = {v3h}{v3l}; q0 as for nb = 2.
\def\CALX_nudiv_three #1#2#3#4#5{\CALX_nudiv_hw #5{#2}{#3}#1}%
\def\CALX_nudiv_hw 1#1#2#3#4#5!1#6#7#8#9{\CALX_nudiv_ha %
    {#1#2#3#4}{#5}{#6#7#8#9}}%
\def\CALX_nudiv_ha #1#2#3#4!1#5#6#7#8{\CALX_nudiv_hb %
    {#1}{#2}{#3}{#4}{#5#6#7#8}}%
\def\CALX_nudiv_hb #1#2#3#4#5#6!{%
    \expandafter\CALX_nudiv_hv
    \expanded{\CALX_nudiv_mp{\CALX_nudiv_gs 1#5#6!%
        \xint:}}{#1}{#2}{#3}{#4}{#5}{#6}%
}%
\def\CALX_nudiv_hv #1.#2.#3#4#5#6#7#8{%
    \expandafter\CALX_nudiv_hc
    \the\numexpr#7#8*#1+(#5#6+#2/\xint_c_ii)/#2-\xint_c_i\xint:%
    {#1}{{#7}{#8}{#5}{#6}}{{#3}{#4}}%
}%
\def\CALX_nudiv_hc #1\xint:#2#3#4#5#6{%
    \expandafter\CALX_nudiv_he
    \romannumeral0\CALX_rev_nounsep {}#6\R!\R!\R!\R!\R!\R!\R!\W
    \xint:{{#1}{#2}{#3}}{#4}{#5}%
}%
\def\CALX_nudiv_hcmp #1#2#3#4#5{%
    \ifnum#1<\CALX_ndth_vone#4 -\xint_c_i\else\ifnum#1>\CALX_ndth_vone#4 %
        \xint_c_\else
    \ifnum#2<\CALX_ndth_vtwo#4 -\xint_c_i\else\ifnum#2>\CALX_ndth_vtwo#4 %
        \xint_c_\else
    \ifnum#3<\CALX_ndth_vthree#5 -\xint_c_i\else\xint_c_\fi\fi\fi\fi
}%
\def\CALX_nudiv_he 1#1!1#2!1#3!#4\xint:#5#6#7{%
    \ifnum\CALX_nudiv_hcmp{#1}{#2}{#3}{#5}{#6}<\xint_c_
       \ifnum#7=\xint_c_iii \CALX_nudiv_hq\fi
       \expandafter\CALX_nudiv_hn
    \else
       \expandafter\CALX_nudiv_hp
    \fi {#1}{#2}{#3}{#4}{#5}{#6}{#7}%
}%
\def\CALX_nudiv_hq\fi\expandafter\CALX_nudiv_hn\else\expandafter%
    \CALX_nudiv_hp\fi
    #1#2#3#4#5#6#7#8{\fi\fi#8{0}{}{0}{1#3!1#2!1#1!1;!}}%
\def\CALX_nudiv_hn #1#2#3#4#5#6#7{%
    \expandafter\CALX_ndth_z\the\numexpr\CALX_ndth %
        {#5}{#6}{#1}{#2}{#3}#41;!{#7-\xint_c_iii}{0}%
```

```
}%
\def\CALX_nudiv_hp #1#2#3#4#5#6#7{%
    \expandafter\CALX_ndth_z\the\numexpr\CALX_ndth %
        {#5}{#6}{0}{#1}{#2}1#3!#41;!{#7-\xint_c_ii}{0}%
}%
```

29.14.9. \CALX_ndth

```
\def\CALX_ndth #1{\CALX_ndth_a #1}%
\def\CALX_ndth_a #1#2#3#4#5#6#7{%
    \expandafter\CALX_ndth_q
    \the\numexpr(#5*#2)*\xint_c_e_viii/#1+#6*#2/#1+(#7*#2/\xint_c_e_viii)/#1%
        \xint:%
    #3{#5}{#6}{#7}{{#1}{#2}{#3}}{#4}%
}%
\def\CALX_ndth_q #1\xint:%
    {\ifnum#1>\xint_c_e_viii_mone \CALX_ndth_qm\fi\CALX_ndth_h #1\xint:}%
\def\CALX_ndth_qm\fi#1\xint:{\fi\CALX_ndth_h 99999999\xint:}%
\def\CALX_ndth_h #1\xint:{%
    \expandafter\CALX_ndth_x
    \the\numexpr(#1+5000)/\xint_c_e_iv-\xint_c_i\expandafter\xint:%
    \the\numexpr#1-\xint_c_e_iv*((#1+5000)/\xint_c_e_iv-\xint_c_i)\xint:%
}%
\def\CALX_ndth_x #1\xint:#2\xint:#3#4#5#6#7#8#9{%
    \expandafter\CALX_ndth_w\the\numexpr
    \xint_c_iii*\xint_c_e_viii+#9-#2*#6%
    -(#1*#6+#2*#5-\xint_c_e_iv*((#1*#6+#2*#5+5000)/\xint_c_e_iv-\xint_c_i))%
     *\xint_c_e_iv
    \expandafter\xint:\the\numexpr
    (#7-#1*#3-(#1*#4+#2*#3+5000)/\xint_c_e_iv+\xint_c_i)*\xint_c_e_viii+#8%
    -(#1*#4+#2*#3-\xint_c_e_iv*((#1*#4+#2*#3+5000)/\xint_c_e_iv-\xint_c_i))%
     *\xint_c_e_iv
    -#2*#4-#1*#5-(#1*#6+#2*#5+5000)/\xint_c_e_iv+\xint_c_i
    \xint:{#1*\xint_c_e_iv+#2}{#3}{#4}{#5}{#6}%
}%
\def\CALX_ndth_w #1#2\xint:#3\xint:{%
    \expandafter\CALX_ndth_l\the\numexpr#3+#1-%
        \xint_c_iii.{#2}%
}%
\def\CALX_ndth_l #1.#2#3#4#5#6#7{%
    \ifnum#1<\xint_c_ \xint_dothis\CALX_ndth_d\fi
    \ifnum\numexpr\xint_c_ii*(#1-#4#5)\ifnum#2<#6#7 -\xint_c_i\fi
        <\xint_c_ \xint_dothis\CALX_ndth_g\fi
    \ifnum\numexpr#3=\xint_c_e_viii_mone \xint_dothis\CALX_ndth_g\fi
    \xint_orthat\CALX_ndth_i {#1}{#2}{#3}{#4}{#5}{#6}{#7}%
}%
\def\CALX_ndth_d #1#2#3#4#5#6#7{%
    \expandafter\CALX_ndth_e\the\numexpr\xint_c_e_viii+#2+#6#7\xint:%
    {#1+#4#5}{#3-\xint_c_i}{#4}{#5}{#6}{#7}%
}%
\def\CALX_ndth_i #1#2#3#4#5#6#7{%
    \expandafter\CALX_ndth_e\the\numexpr\xint_c_ii*\xint_c_e_viii+#2-#6#7%
        \xint:%
```

```
    {#1-#4#5-\xint_c_i}{#3+\xint_c_i}{#4}{#5}{#6}{#7}%
}%
\def\CALX_ndth_e #1#2\xint:#3#4{%
    \expandafter\CALX_ndth_f\the\numexpr#3+#1-\xint_c_i\expandafter\xint:%
    \the\numexpr#4\xint:{#2}%
}%
\def\CALX_ndth_f #1\xint:#2\xint:#3{\CALX_ndth_l #1.{#3}{#2}}%
```

Exit of the correction loop: #1=a, #2=c, #3=q, #4-#7=v1h v1l v2h v2l, #8=C12, #9=V3, then 1<u3>! in the stream.

```
\def\CALX_ndth_g #1#2#3#4#5#6#7#8#9{%
    \expandafter\CALX_ndth_m\the\numexpr(#3+5000)/\xint_c_e_iv-\xint_c_i%
        \xint:#9%
    {#1}{#2}{#3}{#8}%
}%
\def\CALX_ndth_m #1\xint:#2#3#4#5#6#71#8!{%
    \expandafter\CALX_ndth_n\the\numexpr
    \xint_c_iii*\xint_c_e_viii+#8-(#6-\xint_c_e_iv*#1)*#3%
    -(#1*#3+(#6-\xint_c_e_iv*#1)*#2%
      -\xint_c_e_iv*((#1*#3+(#6-\xint_c_e_iv*#1)*#2+5000)/\xint_c_e_iv-%
          \xint_c_i))%
     *\xint_c_e_iv
    \xint:{#1*#2+(#1*#3+(#6-\xint_c_e_iv*#1)*#2+5000)/\xint_c_e_iv-\xint_c_i}%
    {#4}{#5}{#6}{#7}{{#2}{#3}}%
}%
\def\CALX_ndth_n #1#2\xint:#3#4#5{%
    \expandafter\CALX_ndth_o\the\numexpr
    \xint_c_iii*\xint_c_e_viii+#5-(#3)-\xint_c_iii+#1%
        \xint:{#2}{#4}%
}%
\def\CALX_ndth_o #1#2\xint:#3#4{%
    \expandafter\CALX_ndth_oa\the\numexpr#4+#1-\xint_c_iii\xint:{#2}{#3}%
}%
\def\CALX_ndth_oa #1\xint:{%
    \ifnum#1<\xint_c_ \expandafter\CALX_ndth_ab\else\expandafter\CALX_ndth_p%
        \fi {#1}%
}%
\def\CALX_ndth_p #1#2#3#4#5#61#7#8!{%
    \xint_gob_til_sc #7\CALX_ndth_end;%
    \xint_c_e_viii+#4\expandafter!\the\numexpr\CALX_ndth %
        {#5}{#6}{#1}{#2}{#3}1#7#8!%
}%
\def\CALX_ndth_end;\xint_c_e_viii+#1\expandafter!\the\numexpr\CALX_ndth %
    #2#3#4#5#61;!%
    {\xint_c_e_viii+#1\relax!\xint:{#4}{#5}{#6}}%
```

Add back (rare): #1=r0 (<0), #2=r1, #3=r2, #4=q, #5=C12, #6=V3.

```
\def\CALX_ndth_vone   #1#2#3{\CALX_ndth_vonea #3}%
\def\CALX_ndth_vonea  #1#2#3#4{#1#2}%
\def\CALX_ndth_vtwo   #1#2#3{\CALX_ndth_vtwoa #3}%
\def\CALX_ndth_vtwoa  #1#2#3#4{#3#4}%
\def\CALX_ndth_vthree #1#2{#1#2}%
\def\CALX_ndth_ab #1#2#3#4#5#6{%
    \expandafter\CALX_ndth_ac\the\numexpr\xint_c_e_viii+#3+%
```

```
        \CALX_ndth_vthree#6\xint:%
    {#1}{#2}{#4}{#5}{#6}%
}%
\def\CALX_ndth_ac #1#2\xint:#3#4#5#6{%
    \expandafter\CALX_ndth_ad\the\numexpr\xint_c_e_viii+#4+%
        \CALX_ndth_vtwo#6+#1-\xint_c_i\xint:%
    {#2}{#3}{#5}{#6}%
}%
\def\CALX_ndth_ad #1#2\xint:#3#4#5#6{%
    \expandafter\CALX_ndth_ae\the\numexpr#4+\CALX_ndth_vone#6+#1-\xint_c_i%
        \xint:%
    {#2}{#3}{#5-\xint_c_i}{#6}%
}%
\def\CALX_ndth_ae #1\xint:{\CALX_ndth_p {#1}}%
\def\CALX_ndth_z #1\xint:#2#3#4{%
    \expandafter\CALX_ndth_zz\the\numexpr\xint_c_e_viii+(#2)%
        \xint:{#3}{#4}{#1}%
}%
\def\CALX_ndth_zz #1\xint:#2#3#4{\CALX_nudiv_qx #4\xint:{1#3!1#2!#1!1;!}}%
```

Four-block divisor (nb = 4), no normalization, estimate and loop as for nb = 2 (duplicated as \CALX_ndfo_q ... \CALX_ndfo_f). With T = a b + c, the window minus q B' is a b^3 + c b^2 + u3 b + u4 - q (v3 b + v4):

```
   X4 = 3b + u4 - ql v4l - 10^4 M4l                  digits e4<r3>
   X3 = 5b + u3 - ql v3l - 10^4 M3l - H4 - 3 + e4    digits e3<r2>
   Y  = 3b + c - H3 - 5 + e3                          digits e2<r1>
   r0 = a + e2 - 3   (add-back if negative)
```

with Hk = qh vkh + Mkh, Mk = qh vkl + ql vkh.

Loop state: \CALX_ndfo {C12}{V34}{u0}{u1}{u2}{u3}1<u4>!...1;! C12 = {Vp}{M}{{v1h}{v1l}{v2h}{v2l}}, V34 = {v3h}{v3l}{v4h}{v4l}.

After the correction loop: pack P3 = {a}{c}{q}{C12}.

```
\def\CALX_nudiv_four #1#2#3#4#5{\CALX_nudiv_fw #5{#2}{#3}#1}%
\def\CALX_nudiv_fw 1#1#2#3#4#5!1#6#7#8#9{\CALX_nudiv_fa %
    {#1#2#3#4}{#5}{#6#7#8#9}}%
\def\CALX_nudiv_fa #1#2#3#4!1#5#6#7#8{\CALX_nudiv_fb %
    {{#3}{#4}{#1}{#2}}{#5#6#7#8}}%
\def\CALX_nudiv_fb #1#2#3!1#4#5#6#7{\CALX_nudiv_fc {#1}{#2}{#3}{#4#5#6#7}}%
\def\CALX_nudiv_fc #1#2#3#4#5!{%
    \expandafter\CALX_nudiv_fv
    \expanded{\CALX_nudiv_mp{\CALX_nudiv_gs 1#4#5!%
        \xint:}}{#4}{#5}{#2}{#3}{#1}%
}%
\def\CALX_nudiv_fv #1.#2.#3#4#5#6{%
    \expandafter\CALX_nudiv_fd
    \the\numexpr#3#4*#1+(#5#6+#2/\xint_c_ii)/#2-\xint_c_i\xint:%
    {#1}{{#3}{#4}{#5}{#6}}%
}%
\def\CALX_nudiv_fd #1\xint:#2#3#4#5#6{%
    \expandafter\CALX_nudiv_fe
    \romannumeral0\CALX_rev_nounsep {}#6\R!\R!\R!\R!\R!\R!\R!\W
    \xint:{{#1}{#2}{#3}}{#4}{#5}%
}%
\def\CALX_nudiv_fcmp #1#2#3#4#5#6{%
```

```
    \ifnum#1<\CALX_ndth_vone#5 -\xint_c_i\else\ifnum#1>\CALX_ndth_vone#5 %
        \xint_c_\else
    \ifnum#2<\CALX_ndth_vtwo#5 -\xint_c_i\else\ifnum#2>\CALX_ndth_vtwo#5 %
        \xint_c_\else
    \ifnum#3<\CALX_ndfo_vthree#6 -\xint_c_i\else\ifnum#3>\CALX_ndfo_vthree#6 %
        \xint_c_\else
    \ifnum#4<\CALX_ndfo_vfour#6 -\xint_c_i\else\xint_c_\fi\fi\fi\fi\fi\fi\fi
}%
\def\CALX_nudiv_fe 1#1!1#2!1#3!1#4!#5\xint:#6#7#8{%
    \ifnum\CALX_nudiv_fcmp{#1}{#2}{#3}{#4}{#6}{#7}<\xint_c_
       \ifnum#8=\xint_c_iv \CALX_nudiv_fq\fi
       \expandafter\CALX_nudiv_fn
    \else
       \expandafter\CALX_nudiv_fp
    \fi {#1}{#2}{#3}{#4}{#5}{#6}{#7}{#8}%
}%
\def\CALX_nudiv_fq\fi\expandafter\CALX_nudiv_fn\else\expandafter%
    \CALX_nudiv_fp\fi
    #1#2#3#4#5#6#7#8#9{\fi\fi#9{0}{}{0}{1#4!1#3!1#2!1#1!1;!}}%
\def\CALX_nudiv_fn #1#2#3#4#5#6#7#8{%
    \expandafter\CALX_ndfo_z
    \the\numexpr\CALX_ndfo {#6}{#7}{#1}{#2}{#3}{#4}#51;!{#8-\xint_c_iv}{0}%
}%
\def\CALX_nudiv_fp #1#2#3#4#5#6#7#8{%
    \expandafter\CALX_ndfo_z
    \the\numexpr\CALX_ndfo {#6}{#7}{0}{#1}{#2}{#3}1#4!#51;!{#8-%
        \xint_c_iii}{0}%
}%
```

29.14.10. \CALX_ndfo

```
\def\CALX_ndfo #1{\CALX_ndfo_a #1}%
\def\CALX_ndfo_a #1#2#3#4#5#6#7{%
    \expandafter\CALX_ndfo_q
    \the\numexpr(#5*#2)*\xint_c_e_viii/#1+#6*#2/#1+(#7*#2/\xint_c_e_viii)/#1%
        \xint:%
    #3{#5}{#6}{#7}{{#1}{#2}{#3}}{#4}%
}%
\def\CALX_ndfo_q #1\xint:%
    {\ifnum#1>\xint_c_e_viii_mone \CALX_ndfo_qm\fi\CALX_ndfo_h #1\xint:}%
\def\CALX_ndfo_qm\fi#1\xint:{\fi\CALX_ndfo_h 99999999\xint:}%
\def\CALX_ndfo_h #1\xint:{%
    \expandafter\CALX_ndfo_x
    \the\numexpr(#1+5000)/\xint_c_e_iv-\xint_c_i\expandafter\xint:%
    \the\numexpr#1-\xint_c_e_iv*((#1+5000)/\xint_c_e_iv-\xint_c_i)\xint:%
}%
\def\CALX_ndfo_x #1\xint:#2\xint:#3#4#5#6#7#8#9{%
    \expandafter\CALX_ndfo_w\the\numexpr
    \xint_c_iii*\xint_c_e_viii+#9-#2*#6%
    -(#1*#6+#2*#5-\xint_c_e_iv*((#1*#6+#2*#5+5000)/\xint_c_e_iv-\xint_c_i))%
     *\xint_c_e_iv
    \expandafter\xint:\the\numexpr
    (#7-#1*#3-(#1*#4+#2*#3+5000)/\xint_c_e_iv+\xint_c_i)*\xint_c_e_viii+#8%
```

```
    -(#1*#4+#2*#3-\xint_c_e_iv*((#1*#4+#2*#3+5000)/\xint_c_e_iv-\xint_c_i))%
     *\xint_c_e_iv
    -#2*#4-#1*#5-(#1*#6+#2*#5+5000)/\xint_c_e_iv+\xint_c_i
    \xint:{#1*\xint_c_e_iv+#2}{#3}{#4}{#5}{#6}%
}%
\def\CALX_ndfo_w #1#2\xint:#3\xint:{%
    \expandafter\CALX_ndfo_l\the\numexpr#3+#1-%
        \xint_c_iii.{#2}%
}%
\def\CALX_ndfo_l #1.#2#3#4#5#6#7{%
    \ifnum#1<\xint_c_ \xint_dothis\CALX_ndfo_d\fi
    \ifnum\numexpr\xint_c_ii*(#1-#4#5)\ifnum#2<#6#7 -\xint_c_i\fi
        <\xint_c_ \xint_dothis\CALX_ndfo_g\fi
    \ifnum\numexpr#3=\xint_c_e_viii_mone \xint_dothis\CALX_ndfo_g\fi
    \xint_orthat\CALX_ndfo_i {#1}{#2}{#3}{#4}{#5}{#6}{#7}%
}%
\def\CALX_ndfo_d #1#2#3#4#5#6#7{%
    \expandafter\CALX_ndfo_e\the\numexpr\xint_c_e_viii+#2+#6#7\xint:%
    {#1+#4#5}{#3-\xint_c_i}{#4}{#5}{#6}{#7}%
}%
\def\CALX_ndfo_i #1#2#3#4#5#6#7{%
    \expandafter\CALX_ndfo_e\the\numexpr\xint_c_ii*\xint_c_e_viii+#2-#6#7%
        \xint:%
    {#1-#4#5-\xint_c_i}{#3+\xint_c_i}{#4}{#5}{#6}{#7}%
}%
\def\CALX_ndfo_e #1#2\xint:#3#4{%
    \expandafter\CALX_ndfo_f\the\numexpr#3+#1-\xint_c_i\expandafter\xint:%
    \the\numexpr#4\xint:{#2}%
}%
\def\CALX_ndfo_f #1\xint:#2\xint:#3{\CALX_ndfo_l #1.{#3}{#2}}%
```

```
Exit: #1=a, #2=c, #3=q, #4-#7 v1h v1l v2h v2l, #8=C12, #9=V34, then {u3}1<u4>! in the stream.
```

```
\def\CALX_ndfo_pa #1#2#3#4{#1}%
\def\CALX_ndfo_pc #1#2#3#4{#2}%
\def\CALX_ndfo_pq #1#2#3#4{#3}%
\def\CALX_ndfo_g #1#2#3#4#5#6#7#8#9{\CALX_ndfo_m {{#1}{#2}{#3}{#8}}#9}%
\def\CALX_ndfo_m #1#2#3#4#5#61#7!{%
    \expandafter\CALX_ndfo_n
    \the\numexpr(\CALX_ndfo_pq#1+5000)/\xint_c_e_iv-\xint_c_i\xint:%
    {#1}{#2}{#3}{#4}{#5}{#6}{#7}%
}%
\def\CALX_ndfo_n #1\xint:#2#3#4#5#6#7#8{%
    \expandafter\CALX_ndfo_o\the\numexpr
    \xint_c_iii*\xint_c_e_viii+#8-(\CALX_ndfo_pq#2-\xint_c_e_iv*#1)*#6%
    -(#1*#6+(\CALX_ndfo_pq#2-\xint_c_e_iv*#1)*#5-\xint_c_e_iv*((#1*#6+(%
        \CALX_ndfo_pq#2-\xint_c_e_iv*#1)*#5+5000)/\xint_c_e_iv-\xint_c_i))*%
        \xint_c_e_iv
    \xint:{#1*#5+(#1*#6+(\CALX_ndfo_pq#2-\xint_c_e_iv*#1)*#5+5000)/%
        \xint_c_e_iv-\xint_c_i}{#1}{#2}{#3}{#4}{#7}%
    {{#3}{#4}{#5}{#6}}%
}%
\def\CALX_ndfo_o #1#2\xint:#3#4#5#6#7#8{%
    \expandafter\CALX_ndfo_r\the\numexpr
```

```
    \xint_c_v*\xint_c_e_viii+#8-(\CALX_ndfo_pq#5-\xint_c_e_iv*#4)*#7%
    -(#4*#7+(\CALX_ndfo_pq#5-\xint_c_e_iv*#4)*#6-\xint_c_e_iv*((#4*#7+(%
        \CALX_ndfo_pq#5-\xint_c_e_iv*#4)*#6+5000)/\xint_c_e_iv-\xint_c_i))*%
        \xint_c_e_iv
    -(#3)-\xint_c_iii+#1%
    \xint:{#4*#6+(#4*#7+(\CALX_ndfo_pq#5-\xint_c_e_iv*#4)*#6+5000)/%
        \xint_c_e_iv-\xint_c_i}{#2}{#5}%
}%
\def\CALX_ndfo_r #1#2\xint:#3#4#5{%
    \expandafter\CALX_ndfo_s\the\numexpr
    \xint_c_iii*\xint_c_e_viii+\CALX_ndfo_pc#5-(#3)-\xint_c_v+#1%
        \xint:{#2}{#4}{#5}%
}%
\def\CALX_ndfo_s #1#2\xint:#3#4#5{%
    \expandafter\CALX_ndfo_t\the\numexpr\CALX_ndfo_pa#5+#1-\xint_c_iii\xint:%
    {#2}{#3}{#4}{#5}%
}%
\def\CALX_ndfo_t #1\xint:#2#3#4#5{%
    \ifnum#1<\xint_c_ \expandafter\CALX_ndfo_ab\else\expandafter\CALX_ndfo_tu%
        \fi
    #5{#1}{#2}{#3}{#4}%
}%
\def\CALX_ndfo_tu #1#2{\CALX_ndfo_y}%
\def\CALX_ndfo_y #1#2#3#4#5#6#71#8#9!{%
    \xint_gob_til_sc #8\CALX_ndfo_end;%
    \xint_c_e_viii+#1\expandafter!\the\numexpr\CALX_ndfo %
        {#2}{#7}{#3}{#4}{#5}{#6}1#8#9!%
}%
\def\CALX_ndfo_end;\xint_c_e_viii+#1\expandafter!\the\numexpr\CALX_ndfo %
    #2#3#4#5#6#71;!%
    {\xint_c_e_viii+#1\relax!\xint:{#4}{#5}{#6}{#7}}%
```

Add back (rare): #1=a, #2=c, #3=q, #4=C12 (P3 unpacked), #5=r0 (<0), #6=r1, #7=r2, #8=r3, #9=V34.

```
\def\CALX_ndfo_vthree #1#2#3#4{#1#2}%
\def\CALX_ndfo_vfour  #1#2#3#4{#3#4}%
\def\CALX_ndfo_pvone  #1#2#3#4{\CALX_ndth_vone#4}%
\def\CALX_ndfo_pvtwo  #1#2#3#4{\CALX_ndth_vtwo#4}%
\def\CALX_ndfo_ab #1#2#3#4#5#6#7#8#9{%
    \expandafter\CALX_ndfo_ac\the\numexpr\xint_c_e_viii+#8+\CALX_ndfo_vfour#9%
        \xint:%
    {#5}{#6}{#7}{#9}{{#1}{#2}{#3}{#4}}%
}%
\def\CALX_ndfo_ac #1#2\xint:#3#4#5#6{%
    \expandafter\CALX_ndfo_ad\the\numexpr\xint_c_e_viii+#5+%
        \CALX_ndfo_vthree#6+#1-\xint_c_i
    \xint:{#2}{#3}{#4}{#6}%
}%
\def\CALX_ndfo_ad #1#2\xint:#3#4#5#6#7{%
    \expandafter\CALX_ndfo_ae\the\numexpr\xint_c_e_viii+#5+%
        \CALX_ndfo_pvtwo#7+#1-\xint_c_i
    \xint:{#2}{#3}{#4}{#6}{#7}%
}%
\def\CALX_ndfo_ae #1#2\xint:#3#4#5#6#7{%
```

```
    \expandafter\CALX_ndfo_af\the\numexpr#5+\CALX_ndfo_pvone#7+#1-\xint_c_i%
        \xint:%
    {#2}{#3}{#4}{#6}{#7}%
}%
\def\CALX_ndfo_af #1\xint:#2#3#4#5#6{\CALX_ndfo_ah #6{#1}{#2}{#3}{#4}{#5}}%
\def\CALX_ndfo_ah #1#2#3{\CALX_ndfo_y {#3-\xint_c_i}}%
\def\CALX_ndfo_z #1\xint:#2#3#4#5{%
    \expandafter\CALX_ndfo_zz\the\numexpr\xint_c_e_viii+(#2)%
        \xint:{#3}{#4}{#5}{#1}%
}%
\def\CALX_ndfo_zz #1\xint:#2#3#4#5{\CALX_nudiv_qx #5%
    \xint:{1#4!1#3!1#2!#1!1;!}}%
```

29.15. Native signed division

29.15.1. \CALX_ndivtrunc, \CALX_nmodtrunc, \CALX_ndivfloor, \CALX_nmod, \CALX_nquo, \CALX_nrem, \CALX_ndivround, \CALX_ndivision, \CALX_ndivmod

Syntax (use with \romannumeral0; A, B f-expanded native integers):
  \CALX_n<op>{A}{B} -> <space>Z (op: divtrunc, modtrunc,
                                        divfloor, mod, quo, rem, divround)
    \CALX_ndivision{A}{B} -> {Q}{R}
    \CALX_ndivmod{A}{B}   -> {Q}{R}
Z, Q, R native integers (zero divisor: I values or the trap value, see below).
  Signed division. All macros are used as \romannumeral0\CALX_n...{A}{B}, the arguments being f-expanded native integers (A first). With a = |A|, b = |B|, q = floor(a/b), r = a - q b (unsigned core \CALX_nudiv), sA, sB the signs:
  - \CALX_ndivtrunc: sA sB q (truncation);
  - \CALX_nmodtrunc: sA r, remainder of the truncated division;
  - \CALX_ndivfloor: sA sB (q + d), d = 1 if sA <> sB and r <> 0, else 0;
  - \CALX_nmod: A - B \CALX_ndivfloor(A,B), i.e. sA r + e sB b with e = 1 if sA <> sB and r <> 0, else 0 (sign of B or zero);
  - \CALX_nquo: sA sB (q + d), d = 1 if sA = - and r <> 0, else 0; this is sB floor(A/b) (xint's \xintiiQuo);
  - \CALX_nrem: A - B \CALX_nquo(A,B) = sA r + e b, e = 1 if sA = - and r <> 0, else 0; it is >= 0 (xint's \xintiiRem);
  - \CALX_ndivround: sA sB (q + d), d = 1 if 2r >= b, else 0 (rounding half away from zero, xint's \xintiiDivRound);
  - \CALX_ndivision: {\CALX_nquo}{\CALX_nrem} (xint's \xintiiDivision);
  - \CALX_ndivmod: {\CALX_ndivfloor}{\CALX_nmod} (xint's \xintiiDivMod).
  A zero divisor is signalled as in Python's decimal module (conditions of the signals section, trap value N+0. which is what xint returns):
  - quotient of x <> 0 by 0: DivisionByZero, handler argument the sign of x, default result I+ or I-; quotient 0 by 0: DivisionUndefined, I?;
  - remainder of x <> 0 by 0: InvalidOperation, I?; remainder 0 by 0: DivisionUndefined, I?;
  - pair, x <> 0 by 0: DivisionByZero then InvalidOperation, {I+ or I-}{I?}; pair 0 by 0: one DivisionUndefined, {I?}{I?}.
  Implementation. \CALX_ndiv_y receives B, A and three tokens of the family: a starter, a macro for B = 0, a macro for A = 0. The starter is followed by {na}{A}{nb}{B}<sA><sB>. For the families which do not need B after the division the starter is a parameterless macro expanding to \CALX_nudiv<cont>, so that nothing is copied; the others copy {nb}{B} once after the arguments of \CALX_nudiv. \CALX_nudiv leaves the following tokens untouched, hence <cont> receives
    {nq}{Q}{s}{R'}<sA><sB>   or   {nq}{Q}{s}{R'}{nb}{B}<sA><sB>

Quotient-only families never compute R (the test r <> 0 is \CALX_nrz on R', which does not need the normalization of R'). q + 1 is one pass of the addition kernel (\CALX_nqinc). The rounding test 2r >= b doubles R with the addition kernel, compares block counts, and for equal counts reads the final borrow state of the subtraction kernel computing 2R - B. The added terms e b and e sB b are done by \CALX_nadd_y.

```
\def\CALX_ndiv_x #1#2\xint:#3#4#5#6%
   {\expandafter\CALX_nsp_t\expandafter\CALX_ndiv_y\expandafter\CALX_ndiv_sp
    \expandafter#1\romannumeral`&&@#6\xint:#1#2\xint:#3#4#5}%
```

\CALX_ndiv_sp <tA><tB><sB>..\xint:<tA><sA>..\xint:<go><z><zz> (A dividend, B divisor; <z> = \CALX_ndiv_zq, _zr or _zp tells quotient, remainder or pair). Quotient: NaN: I?; Inf/Inf: InvalidOperation (I?); Inf/0: Inf with the sign of A; Inf/c: Inf, product of signs; c/Inf: 0.
Remainder: NaN: I?; Inf dividend: InvalidOperation (I?); finite A by Inf: A (Python decimal model, for all members of the family). Pair: {quotient}{remainder}.

```
\def\CALX_ndiv_sp #1#2#3#4\xint:#5#6#7\xint:#8#9%
{%
    \ifx#9\CALX_ndiv_zq\xint_dothis\CALX_ndiv_spq\fi
    \ifx#9\CALX_ndiv_zr\xint_dothis\CALX_ndiv_spr\fi
    \xint_orthat\CALX_ndiv_spp #2#3{#4}#5#6{#7}%
}%
\def\CALX_ndiv_spq #1#2#3#4#5#6#7%
{%
    \if ?#2\xint_dothis{ I?}\fi
    \if ?#5\xint_dothis{ I?}\fi
    \if N#4\xint_dothis{ N+0.}\fi
    \if I#1\xint_dothis{\CALX_nsig{InvalidOperation}{inf // inf}{}}\fi
    \ifnum\CALX_nsp_z#3\xint:=\xint_c_ \xint_dothis{ I#5}\fi
    \xint_orthat{\CALX_nsp_ps#2#5}%
}%
\def\CALX_ndiv_spr #1#2#3#4#5#6#7%
{%
    \if ?#2\xint_dothis{ I?}\fi
    \if ?#5\xint_dothis{ I?}\fi
    \if I#4\xint_dothis{\CALX_nsig{InvalidOperation}{inf mod x}{}}\fi
    \xint_orthat{ N#5#6}%
}%
\def\CALX_ndiv_spp #1#2#3#4#5#6#7%
   {\CALX_npair{\CALX_ndiv_spq#1#2{#3}#4#5{#6}{}}{%
       \CALX_ndiv_spr#1#2{#3}#4#5{#6}{}}}%
\def\CALX_ndiv_y N#1#2.#3\xint:N#4#5.#6\xint:#7#8#9{%
    \ifnum#2=\xint_c_ \xint_dothis{#8#4{#5}}\fi
    \ifnum#5=\xint_c_ \xint_dothis#9\fi
    \xint_orthat{#7{#5}{#6}{#2}{#3}#4#1}%
}%
\def\CALX_ndivtrunc #1{\expandafter\CALX_ndiv_x\romannumeral`&&@#1\xint:%
    \CALX_ndt_go\CALX_ndiv_zq\CALX_ndiv_zz}%
\def\CALX_nmodtrunc #1{\expandafter\CALX_ndiv_x\romannumeral`&&@#1\xint:%
    \CALX_nmt_go\CALX_ndiv_zr\CALX_ndiv_zz}%
\def\CALX_ndivfloor #1{\expandafter\CALX_ndiv_x\romannumeral`&&@#1\xint:%
    \CALX_ndf_go\CALX_ndiv_zq\CALX_ndiv_zz}%
\def\CALX_nmod      #1{\expandafter\CALX_ndiv_x\romannumeral`&&@#1\xint:%
    \CALX_nmd_go\CALX_ndiv_zr\CALX_ndiv_zz}%
\def\CALX_nquo      #1{\expandafter\CALX_ndiv_x\romannumeral`&&@#1\xint:%
```

```
    \CALX_nqu_go\CALX_ndiv_zq\CALX_ndiv_zz}%
\def\CALX_nrem      #1{\expandafter\CALX_ndiv_x\romannumeral`&&@#1\xint:%
    \CALX_nre_go\CALX_ndiv_zr\CALX_ndiv_zz}%
\def\CALX_ndivround #1{\expandafter\CALX_ndiv_x\romannumeral`&&@#1\xint:%
    \CALX_ndr_go\CALX_ndiv_zq\CALX_ndiv_zz}%
\def\CALX_ndivision #1{\expandafter\CALX_ndiv_x\romannumeral`&&@#1\xint:%
    \CALX_ndn_go\CALX_ndiv_zp\CALX_ndiv_zzp}%
\def\CALX_ndivmod   #1{\expandafter\CALX_ndiv_x\romannumeral`&&@#1\xint:%
    \CALX_ndm_go\CALX_ndiv_zp\CALX_ndiv_zzp}%
```

29.15.2. \CALX_nsig

Zero operands. \CALX_ndiv_zq, _zr, _zp receive <sign of A>{nA}.
\CALX_nsig{<condition>}{<message>}{<handler argument>} signals and puts the space token ending \romannumeral0 in front of the result.

```
\def\CALX_ndiv_zz{ N+0.}%
\def\CALX_ndiv_zzp{{N+0.}{N+0.}}%
\def\CALX_nsig #1#2#3%
   {\expandafter\CALX_sp\romannumeral`&&@%
       \CALX_signalcondition{#1}{#2}{#3}{N+0.}}%
\def\CALX_ndiv_zq #1#2{%
    \ifnum#2=\xint_c_ \xint_dothis{\CALX_nsig{DivisionUndefined}{0 // 0}{}}\fi
    \xint_orthat{\CALX_nsig{DivisionByZero}{x // 0}{#1}}%
}%
\def\CALX_ndiv_zr #1#2{%
    \ifnum#2=\xint_c_ \xint_dothis{\CALX_nsig{DivisionUndefined}{0 mod 0}{}}%
        \fi
    \xint_orthat{\CALX_nsig{InvalidOperation}{x mod 0}{}}%
}%
\def\CALX_ndiv_zp #1#2{%
    \ifnum#2=\xint_c_ \xint_dothis\CALX_ndiv_zpu\fi
    \xint_orthat{\CALX_npair{\CALX_nsig{DivisionByZero}{x // 0}{#1}}%
                            {\CALX_nsig{InvalidOperation}{x mod 0}{}}}%
}%
\def\CALX_ndiv_zpu{\expandafter\CALX_ndiv_zpv\romannumeral`&&@%
    \CALX_signalcondition{DivisionUndefined}{divmod(0, 0)}{}{N+0.}\xint:}%
\def\CALX_ndiv_zpv #1\xint:{{#1}{#1}}%
```

29.15.3. \CALX_nq, \CALX_nqinc, \CALX_nrz, \CALX_nrs, \CALX_nrb, \CALX_npair

Helpers.
- \CALX_nq <sign>{nq}{Q}: the native integer (N+0. if nq = 0).
- \CALX_nqinc <sign>{nq}{Q}: the same for Q + 1 (addition kernel with the one block list 100000001!, which is not longer than Q).
- \CALX_nrz <R'>\xint:{<zero>}{<non zero>}: selects according to R' = 0 (R' terminated by 1;!; zero top blocks are allowed).
- \CALX_nrs <sign>{s}{R'}: <sign> R as native integer (N+0. if R = 0).
- \CALX_nrb N+<nR>.<R>\xint:<sR><sB>{nb}{B}: N+0. if R = 0, else sR R + sB B by \CALX_nadd_y.
- \CALX_npair {<code Q>}{<code R>}: {Q}{R}, each code being expanded by \romannumeral0 (R first).

```
\def\CALX_nq #1#2#3{%
    \ifnum#2=\xint_c_ \xint_dothis{ N+0.}\fi
    \xint_orthat{ N#1#2.#3}%
```

```
}%
\def\CALX_nqinc #1#2#3{%
    \ifnum#2=\xint_c_ \xint_dothis{ N#11.100000001!}\fi
    \xint_orthat{\expandafter\CALX_naddm\expandafter#1\the\numexpr#2%
        \expandafter.\the\numexpr\CALX_nadd_a\xint_c_ii
        100000001!1;!1;!1;!1;!\W #31;!1;!1;!1;!\W\xint:}%
}%
\def\CALX_nrz 1#1#2!{%
    \xint_gob_til_sc #1\CALX_nrz_z;%
    \ifnum#1#2=\xint_c_ \expandafter\CALX_nrz\else\expandafter\CALX_nrz_nz\fi
}%
\def\CALX_nrz_z;#1\xint:#2#3{#2}%
\def\CALX_nrz_nz #1\xint:#2#3{#3}%
\def\CALX_nrs #1#2#3%
   {\expandafter\CALX_nrs_a\expandafter#1\romannumeral0%
       \CALX_nudiv_rem{#2}{#3}\xint:}%
\def\CALX_nrs_a #1N+#2.{%
    \ifnum#2=\xint_c_ \xint_dothis\CALX_nrs_z\fi
    \xint_orthat\CALX_nrs_n #1N+#2.%
}%
\def\CALX_nrs_z #1N+#2.#3\xint:{ N+0.}%
\def\CALX_nrs_n #1N+#2.#3\xint:{ N#1#2.#3}%
\def\CALX_nrb N+#1.{%
    \ifnum#1=\xint_c_ \xint_dothis\CALX_nrb_z\fi
    \xint_orthat\CALX_nrb_n #1.%
}%
\def\CALX_nrb_z #1.#2\xint:#3#4#5#6{ N+0.}%
\def\CALX_nrb_n #1.#2\xint:#3#4#5#6%
   {\CALX_nadd_y N#4#5.#6\xint:N#3#1.#21;!1;!1;!1;!\W\xint:}%
\def\CALX_npair #1#2{\expandafter\CALX_npair_a\romannumeral0#2\xint:{#1}}%
\def\CALX_npair_a #1\xint:#2{\expandafter\CALX_npair_b\romannumeral0#2%
    \xint:{#1}}%
\def\CALX_npair_b #1\xint:{{#1}}%
```

Quotients and remainders without B after the division.

```
\def\CALX_ndt_go{\CALX_nudiv\CALX_ndt_q}%
\def\CALX_ndt_q #1#2#3#4#5#6{\CALX_psign#5#6\CALX_nq{#1}{#2}}%
\def\CALX_nmt_go{\CALX_nudiv\CALX_nmt_r}%
\def\CALX_nmt_r #1#2#3#4#5#6{\CALX_nrs#5{#3}{#4}}%
\def\CALX_ndf_go{\CALX_nudiv\CALX_ndf_q}%
\def\CALX_ndf_q #1#2#3#4#5#6{%
    \if#5#6\xint_dothis{\CALX_nq+}\fi
    \xint_orthat{\CALX_nrz#4\xint:{\CALX_nq-}{\CALX_nqinc-}}{#1}{#2}%
}%
\def\CALX_nqu_go{\CALX_nudiv\CALX_nqu_q}%
\def\CALX_nqu_q #1#2#3#4#5#6{%
    \if#5+\xint_dothis{\CALX_nq#6}\fi
    \xint_orthat{\CALX_psign-#6\CALX_nqu_a#4\xint:}{#1}{#2}%
}%
\def\CALX_nqu_a #1#2\xint:{\CALX_nrz#2\xint:{\CALX_nq#1}{\CALX_nqinc#1}}%
```

Families using B after the division: the starter copies {nb}{B}.
  Rounding: \CALX_ndr_a gets R, doubles it (\CALX_nsum_p), \CALX_ndr_c compares 2R with B, \CALX_ndr_e reads the borrow state (1: 2R >= B).

```
\def\CALX_ndr_go #1#2#3#4{\CALX_nudiv\CALX_ndr_q{#1}{#2}{#3}{#4}{#3}{#4}}%
\def\CALX_ndr_q #1#2#3#4#5#6#7#8{%
    \expandafter\CALX_ndr_a\romannumeral0\CALX_nudiv_rem{#3}{#4}\xint:%
    {#5}{#6}#7#8{#1}{#2}%
}%
\def\CALX_ndr_a N+#1.{%
    \ifnum#1=\xint_c_ \xint_dothis\CALX_ndr_a_z\fi
    \xint_orthat\CALX_ndr_a_n #1.%
}%
\def\CALX_ndr_a_z #1.#2\xint:#3#4{\CALX_ndr_dn}%
\def\CALX_ndr_a_n #1.#2\xint:#3#4{%
    \expandafter\CALX_ndr_c
    \romannumeral0\expandafter\CALX_naddm\expandafter+%
    \the\numexpr#1\expandafter.\the\numexpr\CALX_nadd_a\xint_c_ii
    #21;!1;!1;!1;!\W#21;!1;!1;!1;!\W\xint:\xint:{#3}{#4}%
}%
\def\CALX_ndr_c N+#1.#2\xint:#3#4{%
    \ifnum#1>#3 \xint_dothis\CALX_ndr_up\fi
    \ifnum#1<#3 \xint_dothis\CALX_ndr_dn\fi
    \xint_orthat{\expandafter\CALX_ndr_e\the\numexpr
                 \CALX_nsub_a\xint_c_i #41;!1;!1;!1;!\W #21;!1;!1;!1;!\W}%
}%
\def\CALX_ndr_e #11;!#2%
   {\if#21\expandafter\CALX_ndr_up\else\expandafter\CALX_ndr_dn\fi}%
\def\CALX_ndr_up #1#2{\CALX_psign#1#2\CALX_nqinc}%
\def\CALX_ndr_dn #1#2{\CALX_psign#1#2\CALX_nq}%
\def\CALX_nmd_go #1#2#3#4{\CALX_nudiv\CALX_nmd_r{#1}{#2}{#3}{#4}{#3}{#4}}%
\def\CALX_nmd_r #1#2#3#4#5#6#7#8{%
    \if#7#8\xint_dothis{\CALX_nrs#7{#3}{#4}}\fi
    \xint_orthat{\expandafter\CALX_nrb\romannumeral0\CALX_nudiv_rem{#3}{#4}%
                 \xint:#7#8{#5}{#6}}%
}%
\def\CALX_nre_go #1#2#3#4{\CALX_nudiv\CALX_nre_r{#1}{#2}{#3}{#4}{#3}{#4}}%
\def\CALX_nre_r #1#2#3#4#5#6#7#8{%
    \if#7+\xint_dothis{\CALX_nudiv_rem{#3}{#4}}\fi
    \xint_orthat{\expandafter\CALX_nrb\romannumeral0\CALX_nudiv_rem{#3}{#4}%
                 \xint:-+{#5}{#6}}%
}%
\def\CALX_ndn_go #1#2#3#4{\CALX_nudiv\CALX_ndn_r{#1}{#2}{#3}{#4}{#3}{#4}}%
\def\CALX_ndn_r #1#2#3#4#5#6#7#8{%
    \if#7+\xint_dothis{\CALX_npair{\CALX_nq#8{#1}{#2}}{%
        \CALX_nudiv_rem{#3}{#4}}}\fi
    \xint_orthat{\expandafter\CALX_ndn_a\romannumeral0\CALX_nudiv_rem{#3}{#4}%
                 \xint:{#5}{#6}#8{#1}{#2}}%
}%
\def\CALX_ndn_a N+#1.{%
    \ifnum#1=\xint_c_ \xint_dothis\CALX_ndn_a_z\fi
    \xint_orthat\CALX_ndn_a_n #1.%
}%
\def\CALX_ndn_a_z #1.#2\xint:#3#4#5#6#7%
   {\CALX_npair{\CALX_psign-#5\CALX_nq{#6}{#7}}{ N+0.}}%
\def\CALX_ndn_a_n #1.#2\xint:#3#4#5#6#7%
```

```
    {\CALX_npair{\CALX_psign-#5\CALX_nqinc{#6}{#7}}%
         {\CALX_nadd_y N+#3.#4\xint:N-#1.#21;!1;!1;!1;!\W\xint:}}%
\def\CALX_ndm_go #1#2#3#4{\CALX_nudiv\CALX_ndm_r{#1}{#2}{#3}{#4}{#3}{#4}}%
\def\CALX_ndm_r #1#2#3#4#5#6#7#8{%
    \if#7#8\xint_dothis{\CALX_npair{\CALX_nq+{#1}{#2}}{\CALX_nrs#7{#3}{#4}}}%
        \fi
    \xint_orthat{\expandafter\CALX_ndm_a\romannumeral0\CALX_nudiv_rem{#3}{#4}%
                 \xint:{#5}{#6}#7#8{#1}{#2}}%
}%
\def\CALX_ndm_a N+#1.{%
    \ifnum#1=\xint_c_ \xint_dothis\CALX_ndm_a_z\fi
    \xint_orthat\CALX_ndm_a_n #1.%
}%
\def\CALX_ndm_a_z #1.#2\xint:#3#4#5#6#7#8%
   {\CALX_npair{\CALX_nq-{#7}{#8}}{ N+0.}}%
\def\CALX_ndm_a_n #1.#2\xint:#3#4#5#6#7#8%
   {\CALX_npair{\CALX_nqinc-{#7}{#8}}%
         {\CALX_nadd_y N#6#3.#4\xint:N#5#1.#21;!1;!1;!1;!\W\xint:}}%
```

29.16. Already expanded native arithmetic

Inputs to the following interfaces are two expanded native integers, both with N tags. No argument expansion, tag dispatch, or conversion is needed. Each kernel appends its own block terminators.

```
\def\CALX_naddready #1#2{\CALX_nadd_y #1\xint:#21;!1;!1;!1;!\W\xint:}%
\def\CALX_nsubready #1#2{\CALX_nsub_y #2\xint:#11;!1;!1;!1;!\W\xint:}%
\def\CALX_nmulready #1#2{\CALX_nmul_y #1\xint:#21;!\xint:}%
```

29.17. Native power

Syntax (use with \romannumeral0; k a \numexpr expression, k >= 0; X f-expanded native integer, not I):
  \CALX_npow{k}{X} -> <space>N<s><n>.<blocks> (X^k; X^0 = 1)
Binary powering, least significant bit of k first. The even low bits of k only square X (\CALX_npow_e); at the first odd bit the accumulator is the current base itself (no multiplication by 1), then \CALX_npow_l squares the base and multiplies it into the accumulator for each further bit, the last squaring being skipped. The new base and accumulator are computed inside one \expanded and handed over as braced groups.

```
\def\CALX_npow #1#2%
   {\expandafter\CALX_npow_a\the\numexpr#1\expandafter.%
    \romannumeral`&&@#2\xint:}%
\def\CALX_npow_a #1.#2\xint:%
   {\ifnum#1=\xint_c_ \xint_dothis{ N+1.100000001!}\fi
    \xint_orthat{\CALX_npow_e #1.{#2}}}%
\def\CALX_npow_e #1.#2%
{%
    \ifnum#1=\xint_c_i \xint_dothis{ #2}\fi
    \ifodd#1 \xint_dothis{\expandafter\CALX_npow_l
      \the\numexpr(#1-\xint_c_i)/\xint_c_ii\expandafter.%
      \expandafter{\romannumeral0\CALX_nmul{#2}{#2}}{#2}}\fi
    \xint_orthat{\expandafter\CALX_npow_e\the\numexpr#1/\xint_c_ii%
      \expandafter.\expandafter{\romannumeral0\CALX_nmul{#2}{#2}}}%
}%
```

```
\def\CALX_npow_l #1.#2#3%
{%
    \ifnum#1=\xint_c_i \xint_dothis{\CALX_nmul{#3}{#2}}\fi
    \ifodd#1 \xint_dothis{\expandafter\CALX_npow_l
      \the\numexpr(#1-\xint_c_i)/\xint_c_ii\expandafter.%
      \expanded{{\romannumeral0\CALX_nmul{#2}{#2}}%
                {\romannumeral0\CALX_nmul{#3}{#2}}}}\fi
    \xint_orthat{\expandafter\CALX_npow_l\the\numexpr#1/\xint_c_ii%
      \expandafter.\expandafter{\romannumeral0\CALX_nmul{#2}{#2}}{#3}}%
}%
```

29.18. Native partial factorial and factorial

Syntax (use with \romannumeral0; a, b \numexpr expressions, 0 <= a < b < 10^8, k = 1 .. 10^4, not checked):
  \CALX_npfac{a}{b} -> <space>N+<n>.<blocks> ((a+1)(a+2)...b)
    \CALX_nfac{k}   -> <space>N+<n>.<blocks>   (k! = pfac(0, k))
The accumulator is a block list, least significant first, terminated by 1;!; each step multiplies it by one block y < 10^8, a product of four (factors <= 98), three (<= 463), two (<= 9999) or one consecutive integers, with \CALX_smallmul, inside one \numexpr chain which puts the new list in place. The last step (one to four factors, \CALX_pfac_end_..) hands the list to \CALX_nnorm (count, header): the kernel never produces a zero top block.
  Moved here from xint.sty with release 1.2 (to be usable by \bnumexpr).
  Partially rewritten with release 1.2 to benefit from the inner format of the 1.2 multiplication.
  With current default settings of the etex memory and a.t.t.o.w (11/2015) the maximal possible computation is 5971! (which has 19956 digits).
  Note (end november 2015): I also tried out a quickly written recursive (binary split) implementation

```
  \catcode`_ 11
  \catcode`^ 11
  \long\def\xint_firstofthree  #1#2#3{#1}%
  \long\def\xint_secondofthree #1#2#3{#2}%
  \long\def\xint_thirdofthree  #1#2#3{#3}%
  % quickly written factorial using binary split recursive method
  \def\tFac   {\romannumeral-`0\tfac }%
  \def\tfac #1{\expandafter\XINT_mul_out
               \romannumeral-`0\ufac {1}{#1}1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W}%
  \def\ufac #1#2{\ifcase\numexpr#2-#1\relax
                   \expandafter\xint_firstofthree
                 \or
                   \expandafter\xint_secondofthree
                 \else
                   \expandafter\xint_thirdofthree
                 \fi
                 {\the\numexpr\xint_c_e_viii+#1!1;!}%
                 {\the\numexpr\xint_c_e_viii+#1*#2!1;!}%
                 {\expandafter\vfac\the\numexpr (#1+#2)/\xint_c_ii.#1.#2.}%
  }%
  \def\vfac #1.#2.#3.%
  {%
      \expandafter
      \wfac\expandafter
          {\romannumeral-`0\expandafter
```

```
            \ufac\expandafter{\the\numexpr #1+\xint_c_i}{#3}}%
           {\ufac {#2}{#1}}%
   }%
   \def\wfac #1#2{\expandafter\zfac\romannumeral-`0#2\W #1}%
   \def\zfac {\the\numexpr\XINT_mul_loop 100000000!1;!\W }% core multiplication...
   \catcode`_ 8
   \catcode`^ 7
```

and I was quite surprised that it was only about 1.6x--2x slower in the range N=200 to 2000 than the \xintiiFac here which attempts to be smarter...

Note (2017, 1.2l): I found out some code comment of mine that the code here should be more in the style of \xintiiBinomial, but I left matters untouched.

```
\def\CALX_nfac #1%
   {\expandafter\CALX_pfac_b\expandafter1\expandafter.\the\numexpr#1.%
    100000001!1;!}%
\def\CALX_npfac #1#2%
   {\expandafter\CALX_pfac_b\the\numexpr\xint_c_i+#1\expandafter.%
    \the\numexpr#2.100000001!1;!}%
\def\CALX_pfac_b #1.%
{%
    \ifnum #1>9999 \xint_dothis\CALX_pfac_vbigloop \fi
    \ifnum #1>463  \xint_dothis\CALX_pfac_bigloop  \fi
    \ifnum #1>98   \xint_dothis\CALX_pfac_medloop  \fi
                   \xint_orthat\CALX_pfac_smallloop #1.%
}%
\def\CALX_pfac_smallloop #1.#2.%
{%
    \ifcase\numexpr #2-#1\relax
        \expandafter\CALX_pfac_end_
    \or \expandafter\CALX_pfac_end_i
    \or \expandafter\CALX_pfac_end_ii
    \or \expandafter\CALX_pfac_end_iii
    \else\expandafter\CALX_pfac_smallloop_a
    \fi #1.#2.%
}%
\def\CALX_pfac_smallloop_a #1.#2.%
{%
    \expandafter\CALX_pfac_smallloop_b
    \the\numexpr #1+\xint_c_iv\expandafter.%
    \the\numexpr #2\expandafter.%
    \the\numexpr\expandafter\CALX_smallmul
    \the\numexpr \xint_c_e_viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)%
        *(#1+\xint_c_iii)!%
}%
\def\CALX_pfac_smallloop_b #1.%
{%
    \ifnum #1>98  \expandafter\CALX_pfac_medloop   \else
                  \expandafter\CALX_pfac_smallloop \fi #1.%
}%
\def\CALX_pfac_medloop #1.#2.%
{%
    \ifcase\numexpr #2-#1\relax
        \expandafter\CALX_pfac_end_
```

```
    \or \expandafter\CALX_pfac_end_i
    \or \expandafter\CALX_pfac_end_ii
    \else\expandafter\CALX_pfac_medloop_a
    \fi #1.#2.%
}%
\def\CALX_pfac_medloop_a #1.#2.%
{%
    \expandafter\CALX_pfac_medloop_b
    \the\numexpr #1+\xint_c_iii\expandafter.%
    \the\numexpr #2\expandafter.%
    \the\numexpr\expandafter\CALX_smallmul
    \the\numexpr \xint_c_e_viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
}%
\def\CALX_pfac_medloop_b #1.%
{%
    \ifnum #1>463 \expandafter\CALX_pfac_bigloop   \else
                  \expandafter\CALX_pfac_medloop   \fi #1.%
}%
\def\CALX_pfac_bigloop #1.#2.%
{%
    \ifcase\numexpr #2-#1\relax
        \expandafter\CALX_pfac_end_
    \or \expandafter\CALX_pfac_end_i
    \else\expandafter\CALX_pfac_bigloop_a
    \fi #1.#2.%
}%
\def\CALX_pfac_bigloop_a #1.#2.%
{%
    \expandafter\CALX_pfac_bigloop_b
    \the\numexpr #1+\xint_c_ii\expandafter.%
    \the\numexpr #2\expandafter.%
    \the\numexpr\expandafter
    \CALX_smallmul\the\numexpr \xint_c_e_viii+#1*(#1+\xint_c_i)!%
}%
\def\CALX_pfac_bigloop_b #1.%
{%
    \ifnum #1>9999 \expandafter\CALX_pfac_vbigloop  \else
                   \expandafter\CALX_pfac_bigloop   \fi #1.%
}%
\def\CALX_pfac_vbigloop #1.#2.%
{%
    \ifnum #2=#1
         \expandafter\CALX_pfac_end_
    \else\expandafter\CALX_pfac_vbigloop_a
    \fi #1.#2.%
}%
\def\CALX_pfac_vbigloop_a #1.#2.%
{%
    \expandafter\CALX_pfac_vbigloop
    \the\numexpr #1+\xint_c_i\expandafter.%
    \the\numexpr #2\expandafter.%
    \the\numexpr\expandafter\CALX_smallmul\the\numexpr\xint_c_e_viii+#1!%
```

```
}%
\def\CALX_pfac_end_iii #1.#2.%
{%
    \expandafter\CALX_nnorm\expandafter+%
    \the\numexpr\expandafter\CALX_smallmul
    \the\numexpr \xint_c_e_viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)%
        *(#1+\xint_c_iii)!%
}%
\def\CALX_pfac_end_ii #1.#2.%
{%
    \expandafter\CALX_nnorm\expandafter+%
    \the\numexpr\expandafter\CALX_smallmul
    \the\numexpr \xint_c_e_viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
}%
\def\CALX_pfac_end_i #1.#2.%
{%
    \expandafter\CALX_nnorm\expandafter+%
    \the\numexpr\expandafter\CALX_smallmul
    \the\numexpr \xint_c_e_viii+#1*(#1+\xint_c_i)!%
}%
\def\CALX_pfac_end_ #1.#2.%
{%
    \expandafter\CALX_nnorm\expandafter+%
    \the\numexpr\expandafter\CALX_smallmul\the\numexpr \xint_c_e_viii+#1!%
}%
\catcode`N=11 \catcode`I=11
```

29.19. \XINT_split_fromleft

Moved verbatim from xint.sty at 2.0a1 (used there by \xintDecSplit and others, and by the float printer core \XINT_pfloat_P of calxfp.sty).

```
\def\XINT_split_fromleft
    {\expandafter\XINT_split_fromleft_a\the\numexpr\xint_c_viii-}%
\def\XINT_split_fromleft_a #1%
{%
    \xint_UDsignfork
      #1\XINT_split_fromleft_b
       -{\XINT_split_fromleft_end_a #1}%
    \krof
}%
\def\XINT_split_fromleft_b #1.#2#3#4#5#6#7#8#9%
{%
    \expandafter\XINT_split_fromleft_clean
    \the\numexpr1#2#3#4#5#6#7#8#9\expandafter
    \XINT_split_fromleft_a\the\numexpr\xint_c_viii-#1.%
}%
\def\XINT_split_fromleft_end_a #1.%
{%
    \expandafter\XINT_split_fromleft_clean
    \the\numexpr1\csname XINT_split_fromleft_end#1\endcsname
}%
\def\XINT_split_fromleft_clean 1{ }%
```

```
\expandafter\def\csname XINT_split_fromleft_end7\endcsname #1%
   {#1\XINT_split_fromleft_end_b}%
\expandafter\def\csname XINT_split_fromleft_end6\endcsname #1#2%
   {#1#2\XINT_split_fromleft_end_b}%
\expandafter\def\csname XINT_split_fromleft_end5\endcsname #1#2#3%
   {#1#2#3\XINT_split_fromleft_end_b}%
\expandafter\def\csname XINT_split_fromleft_end4\endcsname #1#2#3#4%
   {#1#2#3#4\XINT_split_fromleft_end_b}%
\expandafter\def\csname XINT_split_fromleft_end3\endcsname #1#2#3#4#5%
   {#1#2#3#4#5\XINT_split_fromleft_end_b}%
\expandafter\def\csname XINT_split_fromleft_end2\endcsname #1#2#3#4#5#6%
   {#1#2#3#4#5#6\XINT_split_fromleft_end_b}%
\expandafter\def\csname XINT_split_fromleft_end1\endcsname #1#2#3#4#5#6#7%
   {#1#2#3#4#5#6#7\XINT_split_fromleft_end_b}%
\expandafter\def\csname XINT_split_fromleft_end0\endcsname #1#2#3#4#5#6#7#8%
   {#1#2#3#4#5#6#7#8\XINT_split_fromleft_end_b}%
\def\XINT_split_fromleft_end_b #1\xint_bye#2\xint_bye.{.#1}% puis .
\XINTrestorecatcodesendinput%
```

30. Package calxint implementation

Added at calx 2.0a1.

Contents

.1 Catcodes, ε-TEX and reload detection . . 573
.2 Package identification 574
.3 \xintBool, \xintToggle 574
.4 Integer square root: small and medium sizes 574
.4.1 \CALX_sqrt_start 574
.5 Native integer square root 577
.5.1 \CALX_nsqrt_two 577
.5.2 \CALX_nsqrt_iv 577
.5.3 \CALX_nsqrt_vi 579
.5.4 \CALX_nsqrt_natv, \CALX_nsqrt_nathl, \CALX_nsqrt_natr 582
.5.5 \CALX_nsqrt_core 583
.5.6 \CALX_nsqrt_k 583
.5.7 \CALX_nsqrt_kec 584
.5.8 \CALX_nsqrt_kq 586
.5.9 \CALX_nsqrt_kz 588
.5.10 \CALX_nsqrtrem 590
.6 Decimal shifts 593
.6.1 \CALX_nshr, \CALX_nshl, \CALX_nshrr . . 593
.7 Interface for xintexpr usage 597
.8 Binomial coefficients 619
.9 \xintiiifLt 624
.10 (WIP) \xintRandomDigits 625
.11 (WIP) \XINT_eightrandomdigits, \xintEightRandomDigits 625
.12 (WIP) \xintRandBit 626
.13 (WIP) \xintXRandomDigits 626
.14 (WIP) \xintiiRandRangeAtoB 626
.15 (WIP) \xintiiRandRange 626
.16 \calxintRandRangeAtoB, \calxintRandRange 628
.17 (WIP) Adjustments for engines without uniformdeviate primitive 629

calx: sister of xint.sty. It loads calxcore.sty and adds the native integer square root. The xint macros it uses are copied unmodified and renamed CALX_; xint.sty is not modified.

30.1. Catcodes, ε-TEX and reload detection

The code of this subsection is copied over from the xint packages, with the package names changed.

```
\begingroup\catcode61\catcode48\catcode32=10\relax%
  \catcode13=5    % ^^M
  \endlinechar=13 %
  \catcode123=1   % {
  \catcode125=2   % }
  \catcode64=11   % @
  \catcode44=12   % ,
  \catcode46=12   % .
  \catcode58=12   % :
  \catcode94=7    % ^
  \def\empty{}\def\space{ }\newlinechar10
  \def\z{\endgroup}%
  \expandafter\let\expandafter\x\csname ver@calxint.sty\endcsname
  \expandafter\let\expandafter\w\csname ver@calxcore.sty\endcsname
  \expandafter\ifx\csname numexpr\endcsname\relax
    \expandafter\ifx\csname PackageWarningNoLine\endcsname\relax
      \immediate\write128{^^JPackage calxint Warning:^^J%
                          \space\space\space\space
                          \numexpr not available, aborting input.^^J}%
    \else
      \PackageWarningNoLine{calxint}{\numexpr not available, aborting input}%
    \fi
    \def\z{\endgroup\endinput}%
```

```
  \else
    \ifx\x\relax   % plain-TeX, first loading of calxint.sty
      \ifx\w\relax % but calxcore.sty not yet loaded.
         \def\z{\endgroup\input calxcore.sty\relax}%
      \fi
    \else
      \ifx\x\empty % LaTeX, first loading,
      % variable is initialized, but \ProvidesPackage not yet seen
          \ifx\w\relax % calxcore.sty not yet loaded.
            \def\z{\endgroup\RequirePackage{calxcore}}%
          \fi
      \else
        \def\z{\endgroup\endinput}% calxint already loaded.
      \fi
    \fi
  \fi
\z%
\XINTsetupcatcodes% defined in xintkernel.sty
```

30.2. Package identification

```
\XINT_providespackage
\ProvidesPackage{calxint}%
  [2026/11/26 v2.0a2 A calx component (JFB)]%
```

30.3. \xintBool, \xintToggle

1.09c

```
\def\xintBool #1{\romannumeral`&&@%
                 \csname if#1\endcsname\expandafter1\else\expandafter0\fi }%
\def\xintToggle #1{\romannumeral`&&@\iftoggle{#1}{1}{0}}%
```

30.4. Integer square root: small and medium sizes

30.4.1. \CALX_sqrt_start

xint's square root start and its small and medium paths (inputs of at most 16 digits), renamed. It is only called with at most 16 digits (calx, and xint's \xintiiSquareRoot since s58).
  Output: {N}{D} with D = N^2 - X > 0, N smallest (N = isqrt(X) + 1).

```
\def\CALX_sqrt_start #1.%
{%
    \ifnum #1<\xint_c_x\xint_dothis\CALX_sqrt_small_a\fi
    \xint_orthat\CALX_sqrt_big_a #1.%
}%
\def\CALX_sqrt_small_a #1.{\CALX_sqrt_a #1.\CALX_sqrt_small_d }%
\def\CALX_sqrt_big_a   #1.{\CALX_sqrt_a #1.\CALX_sqrt_big_d   }%
\def\CALX_sqrt_a #1.%
{%
   \ifodd #1
     \expandafter\CALX_sqrt_bO
   \else
     \expandafter\CALX_sqrt_bE
   \fi
```

```
   #1.%
}%
\def\CALX_sqrt_bE #1.#2#3#4%
{%
    \CALX_sqrt_c {#3#4}#2{#1}#3#4%
}%
\def\CALX_sqrt_bO #1.#2#3%
{%
    \CALX_sqrt_c #3#2{#1}#3%
}%
\def\CALX_sqrt_c #1#2%
{%
    \expandafter #2%
    \the\numexpr \ifnum #1>\xint_c_ii
                 \ifnum #1>\xint_c_vi
                 \ifnum #1>12 \ifnum #1>20 \ifnum #1>30
                 \ifnum #1>42 \ifnum #1>56 \ifnum #1>72
                 \ifnum #1>90
      10\else 9\fi \else 8\fi \else 7\fi \else 6\fi \else 5\fi
        \else 4\fi \else 3\fi \else 2\fi \else 1\fi .%
}%
\def\CALX_sqrt_small_d #1.#2%
{%
   \expandafter\CALX_sqrt_small_e
   \the\numexpr #1\ifcase \numexpr #2/\xint_c_ii-\xint_c_i\relax
                    \or 0\or 00\or 000\or 0000\fi .%
}%
\def\CALX_sqrt_small_e #1.#2.%
{%
   \expandafter\CALX_sqrt_small_ea\the\numexpr #1*#1-#2.#1.%
}%
\def\CALX_sqrt_small_ea #1%
{%
    \if0#1\xint_dothis\CALX_sqrt_small_ez\fi
    \if-#1\xint_dothis\CALX_sqrt_small_eb\fi
    \xint_orthat\CALX_sqrt_small_f #1%
}%
\def\CALX_sqrt_small_ez 0.#1.{\expandafter{\the\numexpr#1+\xint_c_i
          \expandafter}\expandafter{\the\numexpr #1*\xint_c_ii+\xint_c_i}}%
\def\CALX_sqrt_small_eb -#1.#2.%
{%
    \expandafter\CALX_sqrt_small_ec \the\numexpr
    (#1-\xint_c_i+#2)/(\xint_c_ii*#2).#1.#2.%
}%
\def\CALX_sqrt_small_ec #1.#2.#3.%
{%
    \expandafter\CALX_sqrt_small_f \the\numexpr
      -#2+\xint_c_ii*#3*#1+#1*#1\expandafter.\the\numexpr #3+#1.%
}%
\def\CALX_sqrt_small_f #1.#2.%
{%
   \expandafter\CALX_sqrt_small_g
```

```
   \the\numexpr (#1+#2)/(\xint_c_ii*#2)-\xint_c_i.#1.#2.%
}%
\def\CALX_sqrt_small_g #1#2.%
{%
    \if 0#1%
       \expandafter\CALX_sqrt_small_end
    \else
       \expandafter\CALX_sqrt_small_h
    \fi
    #1#2.%
}%
\def\CALX_sqrt_small_h #1.#2.#3.%
{%
    \expandafter\CALX_sqrt_small_f
    \the\numexpr #2-\xint_c_ii*#1*#3+#1*#1\expandafter.%
    \the\numexpr #3-#1.%
}%
\def\CALX_sqrt_small_end #1.#2.#3.{{#3}{#2}}%
\def\CALX_sqrt_big_d #1.#2%
{%
   \ifodd #2 \xint_dothis{\expandafter\CALX_sqrt_big_eO}\fi
   \xint_orthat{\expandafter\CALX_sqrt_big_eE}%
   \the\numexpr (#2-\xint_c_i)/\xint_c_ii.#1;%
}%
\def\CALX_sqrt_big_eE  #1;#2#3#4#5#6#7#8#9%
{%
    \CALX_sqrt_big_eE_a #1;{#2#3#4#5#6#7#8#9}%
}%
\def\CALX_sqrt_big_eE_a #1.#2;#3%
{%
    \expandafter\CALX_sqrt_bigormed_f
    \romannumeral0\CALX_sqrt_small_e #2000.#3.#1;%
}%
\def\CALX_sqrt_big_eO #1;#2#3#4#5#6#7#8#9%
{%
    \CALX_sqrt_big_eO_a #1;{#2#3#4#5#6#7#8#9}%
}%
\def\CALX_sqrt_big_eO_a #1.#2;#3#4%
{%
    \expandafter\CALX_sqrt_bigormed_f
    \romannumeral0\CALX_sqrt_small_e #20000.#3#4.#1;%
}%
```

calx (s58): \CALX_sqrt_start is only ever called with at most 16 digits (calxint, xint's \xintiiSquareRoot), hence here k = #3 <= 8: the medium path only. xint's big branch (\XINT_sqrt_big_f, more than 16 digits) is deleted: xint and xintfrac use the native square roots.

```
\def\CALX_sqrt_bigormed_f #1#2#3;%
   {\csname CALX_sqrt_med_f#3\endcsname #1.#2.#3;}%
\expandafter\def\csname CALX_sqrt_med_f5\endcsname   {\CALX_sqrt_med_fa .}%
\expandafter\def\csname CALX_sqrt_med_f6\endcsname  {\CALX_sqrt_med_fa 0.}%
\expandafter\def\csname CALX_sqrt_med_f7\endcsname {\CALX_sqrt_med_fa 00.}%
\expandafter\def\csname CALX_sqrt_med_f8\endcsname{\CALX_sqrt_med_fa 000.}%
\def\CALX_sqrt_med_fa #1.#2.#3.#4;%
```

```
{%
    \expandafter\CALX_sqrt_med_fb
    \the\numexpr (#30#1-5#1)/(\xint_c_ii*#2).#1.#2.#3.%
}%
\def\CALX_sqrt_med_fb #1.#2.#3.#4.#5.%
{%
    \expandafter\CALX_sqrt_small_ea
    \the\numexpr (#40#2-\xint_c_ii*#3*#1)*10#2+(#1*#1-#5)\expandafter.%
    \the\numexpr #30#2-#1.%
}%
```

30.5. Native integer square root

Attention N and I are set to catcode 12 from here on. IN PARTICULAR DO NOT USE ANY \XINT... prefixed macro names with I or N of catcode 12!

```
\catcode`N=12 \catcode`I=12
```

30.5.1. \CALX_nsqrt_two

\CALX_nsqrt_two <cont>{h}{l}: h, l eight digit strings (l with leading zeros), X = h 10^8 + l, h >= 10^7. Expands to <cont>{s}{r} with s = isqrt(X), r = X - s^2 (explicit integers, r <= 2s). Uses (5): s = N - 1, r = 2N - 1 - D.

```
\def\CALX_nsqrt_two #1#2#3%
   {\expandafter\CALX_nsqrt_two_a\romannumeral0\CALX_sqrt_start 16.#2#3.#1}%
\def\CALX_nsqrt_two_a #1#2#3%
   {\expandafter#3\expandafter{\the\numexpr#1-\xint_c_i\expandafter}%
    \expandafter{\the\numexpr\xint_c_ii*#1-\xint_c_i-#2}}%
```

30.5.2. \CALX_nsqrt_iv

\CALX_nsqrt_iv <cont>{a3}{a2}{a1}{a0}: eight digit strings (leading zeros allowed except a3 >= 25000000), X = a3 B^3 + a2 B^2 + a1 B + a0, B=10^8. Expands to <cont>{S}{T}{H}{L}: isqrt(X) = S B + T with S explicit and T as a native block 1ddddddddd, remainder X - isqrt(X)^2 = H B + L, H, L explicit, 0 <= L < B, H <= 2 10^8.

One Zimmermann step on the two block square root (s',r') of a3 B + a2: q = floor((r' B + a1)/(2s')), estimate q0 = round(r' B/(2s')) (eTeX scaled product), exact u = r' B + a1 - 2s' q0 in 10^4 halves (every intermediate < 2^31), one +-1 correction; then r = u B + a0 - q^2 and the final correction r < 0: r += 2s - 1, s -= 1. q = B occurs only with r' = 2s' and then always gets corrected.

Floor of x/10^4 for x >= 0 is (x+5000)/10000-1 in \numexpr.

```
\def\CALX_nsqrt_iv #1#2#3#4#5{\CALX_nsqrt_two%
    \CALX_nsqrt_iv_a{#2}{#3}{#4}{#5}#1}%
\def\CALX_nsqrt_iv_a #1#2%
{%
    \expandafter\CALX_nsqrt_iv_b
    \the\numexpr#2*\xint_c_e_viii/(\xint_c_ii*#1).{#1}{#2}%
}%
\def\CALX_nsqrt_iv_b #1.#2%
{%
    \expandafter\CALX_nsqrt_iv_c
    \the\numexpr(#1+5000)/\xint_c_e_iv-\xint_c_i\expandafter.%
    \the\numexpr(\xint_c_ii*#2+5000)/\xint_c_e_iv-\xint_c_i.#1.{#2}%
}%
```

#1=qh, #2=th, #3=q0, #4=s', #5=r', #6=a1. With t=2s', ql=q0-qh 10^4, tl=t-th 10^4: u = ((r'-th qh) 10^4 - (th ql + tl qh)) 10^4 - tl ql + a1.

```
\def\CALX_nsqrt_iv_c #1.#2.#3.#4#5#6%
{%
    \expandafter\CALX_nsqrt_iv_d
    \the\numexpr((#5-#2*#1)*\xint_c_e_iv
       -(#2*(#3-#1*\xint_c_e_iv)+(\xint_c_ii*#4-#2*\xint_c_e_iv)*#1))*%
           \xint_c_e_iv
       -(\xint_c_ii*#4-#2*\xint_c_e_iv)*(#3-#1*\xint_c_e_iv)+#6.#3.{#4}%
}%
\def\CALX_nsqrt_iv_d #1.#2.#3%
{%
    \expandafter\CALX_nsqrt_iv_e\the\numexpr#1%
      \ifnum#1<\xint_c_ +\xint_c_ii*#3\else
      \ifnum#1<\numexpr\xint_c_ii*#3\relax\else-\xint_c_ii*#3\fi\fi
    \expandafter.\the\numexpr#2%
      \ifnum#1<\xint_c_ -\xint_c_i\else
      \ifnum#1<\numexpr\xint_c_ii*#3\relax\else+\xint_c_i\fi\fi
    .{#3}%
}%
```

#1=u, #2=q (0 <= q <= B), #3=s', #4=a0. With qh, ql the 10^4 halves of q and c = 2 qh ql = ch 10^4 + cl: r = (u - qh^2 - ch) B + (a0 - ql^2 - cl 10^4); the low part is > -2 10^8.

```
\def\CALX_nsqrt_iv_e #1.#2.%
{%
    \expandafter\CALX_nsqrt_iv_f
    \the\numexpr(#2+5000)/\xint_c_e_iv-\xint_c_i.#2.#1.%
}%
\def\CALX_nsqrt_iv_f #1.#2.%
{%
    \expandafter\CALX_nsqrt_iv_g
    \the\numexpr\xint_c_ii*#1*(#2-#1*\xint_c_e_iv).#1.#2.%
}%
\def\CALX_nsqrt_iv_g #1.%
{%
    \expandafter\CALX_nsqrt_iv_h\the\numexpr(#1+5000)/\xint_c_e_iv-%
        \xint_c_i.#1.%
}%
\def\CALX_nsqrt_iv_h #1.#2.#3.#4.#5.#6#7%
{%
    \expandafter\CALX_nsqrt_iv_i
    \the\numexpr#7-(#4-#3*\xint_c_e_iv)*(#4-#3*\xint_c_e_iv)
                -(#2-#1*\xint_c_e_iv)*\xint_c_e_iv\expandafter.%
    \the\numexpr#5-#3*#3-#1.#4.{#6}%
}%
```

#1=low part x in (-2 10^8, 10^8), #2=high part, #3=q, #4=s'. Borrow k = floor(x/B) = (x+250000000)/100000000-3.

```
\def\CALX_nsqrt_iv_i #1.%
{%
    \expandafter\CALX_nsqrt_iv_j\the\numexpr(#1+250000000)/\xint_c_e_viii-%
        \xint_c_iii.#1.%
}%
\def\CALX_nsqrt_iv_j #1.#2.#3.%
```

```
{%
    \expandafter\CALX_nsqrt_iv_k
    \the\numexpr#3+#1\expandafter.\the\numexpr#2-#1*\xint_c_e_viii.%
}%
```

#1=H, #2=L, #3=q, #4=s', #5=cont. If H < 0: H += 2s', L += 2q - 1 (L in [-1, 3 10^8)), carry (L+150000000)/100000000-2, q -= 1 with a borrow from s' if q was 0.

```
\def\CALX_nsqrt_iv_k #1.#2.#3.#4#5%
{%
    \ifnum#1<\xint_c_
      \xint_dothis{\expandafter\CALX_nsqrt_iv_l
         \the\numexpr#2+\xint_c_ii*#3-\xint_c_i\expandafter.%
         \the\numexpr#1+\xint_c_ii*#4\expandafter.%
         \the\numexpr#3-\xint_c_i.{#4}#5}\fi
    \xint_orthat{\expandafter#5\expandafter{\the\numexpr#4\expandafter}%
         \expandafter{\the\numexpr#3+\xint_c_e_viii}{#1}{#2}}%
}%
\def\CALX_nsqrt_iv_l #1.%
{%
    \expandafter\CALX_nsqrt_iv_m\the\numexpr(#1+150000000)/\xint_c_e_viii-%
        \xint_c_ii.#1.%
}%
\def\CALX_nsqrt_iv_m #1.#2.#3.#4.%
{%
    \expandafter\CALX_nsqrt_iv_n
    \the\numexpr#3+#1\expandafter.\the\numexpr#2-#1*\xint_c_e_viii.#4.%
}%
\def\CALX_nsqrt_iv_n #1.#2.#3.#4#5%
{%
    \expandafter#5\expandafter
    {\the\numexpr#4\ifnum#3<\xint_c_ -\xint_c_i\fi\expandafter}\expandafter
    {\the\numexpr\ifnum#3<\xint_c_ 199999999\else#3+\xint_c_e_viii\fi}%
    {#1}{#2}%
}%
```

30.5.3. \CALX_nsqrt_vi

\CALX_nsqrt_vi <cont>{a5}{a4}{a3}{a2}{a1}{a0}: eight digit strings, a5 >= 25000000. Expands to <cont>{S1}{T}{Q}{R2}{R1}{R0}: isqrt(X) = S1 B^2 + T B + Q (T, Q native blocks 1ddddddd), remainder R2 B^2 + R1 B + R0 (explicit, 0 <= R1, R0 < B).

One Zimmermann step with b = B on the four block root s' = S1 B + T (remainder r' = H B + L) of the top four blocks: t = 2s' = tH B + tL, 10^8 <= tH < 2 10^8; q = floor((r' B + a1)/t) is estimated by q0 = round(H B/tH) (scaled), q - q0 in [-2, 1]; u = r' B + a1 - q0 t is computed exactly as uH B + uL from 10^4 halves (qh, ql, tHh, tHl, tLh, tLl): with m0 = a1 - ql tLl, m1 = -(qh tLl + ql tLh), Z = ((H - qh tHh) 10^4 - (qh tHl + ql tHh)) 10^4 + L - (qh tLh + ql tHl), c0 = floor(m0/10^4), c1 = floor((m1+c0)/10^4): uH = Z + c1, uL = (m1 + c0 - c1 10^4) 10^4 + m0 - c0 10^4. Every intermediate is < 2^31 (Z: |u| < 3 t). floor(v/10^4) = (v+500005000)/10000-50001 for |v| < 5 10^8. Then the correction loop, r = u B + a0 - q^2 and the final correction as in \CALX_nsqrt_iv.

```
\def\CALX_nsqrt_vi #1#2#3#4#5#6#7%
   {\CALX_nsqrt_iv\CALX_nsqrt_vi_a{#2}{#3}{#4}{#5}{#6}{#7}#1}%
\def\CALX_nsqrt_vi_a #1#2%
   {\expandafter\CALX_nsqrt_vi_b\the\numexpr%
       \xint_c_ii*#2-200000000.{#1}{#2}}%
```

```
\def\CALX_nsqrt_vi_b #1.#2#3#4#5%
{%
    \expandafter\CALX_nsqrt_vi_c
    \the\numexpr\xint_c_ii*#2\ifnum#1<\xint_c_e_viii\else+\xint_c_i\fi
    \expandafter.\the\numexpr#1\ifnum#1<\xint_c_e_viii\else-\xint_c_e_viii\fi
    .{#4}{#5}{{#2}{#3}}%
}%
\def\CALX_nsqrt_vi_c #1.#2.#3#4#5#6#7%
   {\expandafter\CALX_nsqrt_vi_d\the\numexpr#3*%
       \xint_c_e_viii/#1.#1.#2.{#3}{#4}{#6}{#7}{#5}}%
\def\CALX_nsqrt_vi_d #1.#2.#3.%
{%
    \expandafter\CALX_nsqrt_vi_e
    \the\numexpr(#1+5000)/\xint_c_e_iv-\xint_c_i\expandafter.%
    \the\numexpr(#2+5000)/\xint_c_e_iv-\xint_c_i\expandafter.%
    \the\numexpr(#3+5000)/\xint_c_e_iv-\xint_c_i.#1.#2.#3.%
}%
```

#1=qh, #2=tHh, #3=tLh, #4=q0, #5=tH, #6=tL, #7=H, #8=L, #9=a1.

```
\def\CALX_nsqrt_vi_e #1.#2.#3.#4.#5.#6.#7#8#9%
{%
    \expandafter\CALX_nsqrt_vi_f
    \the\numexpr#9-(#4-#1*\xint_c_e_iv)*(#6-#3*\xint_c_e_iv)\expandafter.%
    \the\numexpr-#1*(#6-#3*\xint_c_e_iv)-(#4-#1*\xint_c_e_iv)*#3\expandafter.%
    \the\numexpr((#7-#1*#2)*\xint_c_e_iv
       -(#1*(#5-#2*\xint_c_e_iv)+(#4-#1*\xint_c_e_iv)*#2))*\xint_c_e_iv
       +#8-(#1*#3+(#4-#1*\xint_c_e_iv)*(#5-#2*\xint_c_e_iv)).#4.#5.#6.%
}%
\def\CALX_nsqrt_vi_f #1.#2.%
{%
    \expandafter\CALX_nsqrt_vi_g
    \the\numexpr#2+(#1+500005000)/\xint_c_e_iv-50001\expandafter.%
    \the\numexpr#1-((#1+500005000)/\xint_c_e_iv-50001)*\xint_c_e_iv.%
}%
\def\CALX_nsqrt_vi_g #1.#2.#3.%
{%
    \expandafter\CALX_nsqrt_vi_k
    \the\numexpr#3+(#1+500005000)/\xint_c_e_iv-50001\expandafter.%
    \the\numexpr(#1-((#1+500005000)/\xint_c_e_iv-50001)*\xint_c_e_iv)*%
        \xint_c_e_iv+#2.%
}%
```

Correction loop: #1=uH, #2=uL, #3=q, #4=tH, #5=tL. While u < 0: u += t, q -= 1; while u >= t: u -= t, q += 1.

```
\def\CALX_nsqrt_vi_k #1.%
{%
    \ifnum#1<\xint_c_ \expandafter\CALX_nsqrt_vi_kadd
    \else\expandafter\CALX_nsqrt_vi_kcmp\fi #1.%
}%
\def\CALX_nsqrt_vi_kadd #1.#2.#3.#4.#5.%
{%
    \expandafter\CALX_nsqrt_vi_k
    \the\numexpr#1+#4+(#2+#5+50000000)/\xint_c_e_viii-\xint_c_i\expandafter.%
    \the\numexpr#2+#5-((#2+#5+50000000)/\xint_c_e_viii-\xint_c_i)*%
```

```
        \xint_c_e_viii
    \expandafter.\the\numexpr#3-\xint_c_i.#4.#5.%
}%
\def\CALX_nsqrt_vi_kcmp #1.#2.#3.#4.#5.%
{%
    \ifcase\numexpr\ifnum#1<#4 0\else\ifnum#1>#4 1\else
                   \ifnum#2<#5 0\else 1\fi\fi\fi\relax
         \expandafter\CALX_nsqrt_vi_r
    \or  \expandafter\CALX_nsqrt_vi_ksub
    \fi #1.#2.#3.#4.#5.%
}%
\def\CALX_nsqrt_vi_ksub #1.#2.#3.#4.#5.%
{%
    \expandafter\CALX_nsqrt_vi_k
    \the\numexpr#1-#4+(#2-#5+150000000)/\xint_c_e_viii-\xint_c_ii%
        \expandafter.%
    \the\numexpr#2-#5-((#2-#5+150000000)/\xint_c_e_viii-\xint_c_ii)*%
        \xint_c_e_viii
    \expandafter.\the\numexpr#3+\xint_c_i.#4.#5.%
}%
```

r = u B + a0 - q^2: #1=uH, #2=uL, #3=q, #4=tH, #5=tL, #6=a0, then {S}. q^2 = (qh^2 + ch) B + ql^2 + cl 10^4 with c = 2 qh ql = ch 10^4 + cl.

```
\def\CALX_nsqrt_vi_r #1.#2.#3.#4.#5.#6%
{%
    \expandafter\CALX_nsqrt_vi_rb
    \the\numexpr(#3+5000)/\xint_c_e_iv-\xint_c_i.#3.#6.#2.#1.{#4}{#5}%
}%
\def\CALX_nsqrt_vi_rb #1.#2.%
   {\expandafter\CALX_nsqrt_vi_rc\the\numexpr\xint_c_ii*#1*(#2-#1*%
       \xint_c_e_iv).#1.#2.}%
\def\CALX_nsqrt_vi_rc #1.%
   {\expandafter\CALX_nsqrt_vi_rd\the\numexpr(#1+5000)/\xint_c_e_iv-%
       \xint_c_i.#1.}%
\def\CALX_nsqrt_vi_rd #1.#2.#3.#4.#5.#6.#7.%
{%
    \expandafter\CALX_nsqrt_vi_re
    \the\numexpr#5-(#4-#3*\xint_c_e_iv)*(#4-#3*\xint_c_e_iv)
                -(#2-#1*\xint_c_e_iv)*\xint_c_e_iv\expandafter.%
    \the\numexpr#6-#3*#3-#1.#7.#4.%
}%
\def\CALX_nsqrt_vi_re #1.#2.%
{%
    \expandafter\CALX_nsqrt_vi_rf
    \the\numexpr#2+(#1+250000000)/\xint_c_e_viii-\xint_c_iii\expandafter.%
    \the\numexpr#1-((#1+250000000)/\xint_c_e_viii-\xint_c_iii)*%
        \xint_c_e_viii.%
}%
\def\CALX_nsqrt_vi_rf #1.#2.#3.%
{%
    \expandafter\CALX_nsqrt_vi_rg
    \the\numexpr#3+(#1+250000000)/\xint_c_e_viii-\xint_c_iii\expandafter.%
    \the\numexpr#1-((#1+250000000)/\xint_c_e_viii-\xint_c_iii)*%
```

```
        \xint_c_e_viii.#2.%
}%
```

#1=R2, #2=R1, #3=R0, #4=q, #5=tH, #6=tL, #7=S, #8=cont. If R2 < 0: r += tH B^2 + tL B + 2q - 1, q -= 1 (q >= 1 here, see \CALX_nsqrt_iv).

```
\def\CALX_nsqrt_vi_rg #1.#2.#3.#4.#5#6#7#8%
{%
    \ifnum#1<\xint_c_
      \xint_dothis{\expandafter\CALX_nsqrt_vi_rh
         \the\numexpr#3+\xint_c_ii*#4-\xint_c_i\expandafter.%
         \the\numexpr#4-\xint_c_i.#2.#1.{#5}{#6}{#7}#8}\fi
    \xint_orthat{\expandafter\CALX_nsqrt_vi_out\expandafter
                {\the\numexpr#4+\xint_c_e_viii}{#1}{#2}{#3}#7#8}%
}%
\def\CALX_nsqrt_vi_out #1#2#3#4#5#6#7{#7{#5}{#6}{#1}{#2}{#3}{#4}}%
\def\CALX_nsqrt_vi_rh #1.#2.#3.#4.%
{%
    \expandafter\CALX_nsqrt_vi_ri
    \the\numexpr(#1+150000000)/\xint_c_e_viii-\xint_c_ii.#1.#3.#4.#2.%
}%
\def\CALX_nsqrt_vi_ri #1.#2.#3.#4.#5.#6#7%
{%
    \expandafter\CALX_nsqrt_vi_rj
    \the\numexpr#3+#7+#1\expandafter.\the\numexpr#2-#1*%
        \xint_c_e_viii.#4.#5.{#6}%
}%
\def\CALX_nsqrt_vi_rj #1.#2.#3.#4.#5#6#7%
{%
    \expandafter\CALX_nsqrt_vi_out\expandafter
    {\the\numexpr#4+\xint_c_e_viii\expandafter}\expandafter
    {\the\numexpr#3+#5+(#1+150000000)/\xint_c_e_viii-\xint_c_ii\expandafter}%
        \expandafter
    {\the\numexpr#1-((#1+150000000)/\xint_c_e_viii-\xint_c_ii)*%
        \xint_c_e_viii}%
    {#2}#6#7%
}%
```

30.5.4. \CALX_nsqrt_natv, \CALX_nsqrt_nathl, \CALX_nsqrt_natr

\CALX_nsqrt_natv{v} (0 <= v < 2^31), \CALX_nsqrt_nathl{H}{L} (H B + L), \CALX_nsqrt_natr{R2}{R1}{R0} (R2 B^2 + R1 B + R0), 0 <= H, R2 < 2^31, 0 <= L, R1, R0 < B: native integer (use with \romannumeral0); built through \CALX_nnorm which strips zero top blocks.
  floor(x/B) = (x+50000000)/100000000-1 for x >= 0.

```
\def\CALX_nsqrt_natv #1{\expandafter\CALX_nnorm\expandafter+%
    \the\numexpr#1-((#1+50000000)/\xint_c_e_viii-\xint_c_i)*\xint_c_e_viii+%
        \xint_c_e_viii\expandafter!%
    \the\numexpr((#1+50000000)/\xint_c_e_viii-\xint_c_i)+\xint_c_e_viii!1;!}%
\def\CALX_nsqrt_nathl #1#2{\expandafter\CALX_nnorm\expandafter+%
    \the\numexpr#2+\xint_c_e_viii\expandafter!%
    \the\numexpr#1-((#1+50000000)/\xint_c_e_viii-\xint_c_i)*\xint_c_e_viii+%
        \xint_c_e_viii\expandafter!%
    \the\numexpr((#1+50000000)/\xint_c_e_viii-\xint_c_i)+\xint_c_e_viii!1;!}%
\def\CALX_nsqrt_natr #1#2#3{\expandafter\CALX_nnorm\expandafter+%
```

```
    \the\numexpr#3+\xint_c_e_viii\expandafter!%
    \the\numexpr#2+\xint_c_e_viii\expandafter!%
    \the\numexpr#1-((#1+50000000)/\xint_c_e_viii-\xint_c_i)*\xint_c_e_viii+%
        \xint_c_e_viii\expandafter!%
    \the\numexpr((#1+50000000)/\xint_c_e_viii-\xint_c_i)+\xint_c_e_viii!1;!}%
```

30.5.5. \CALX_nsqrt_core

\CALX_nsqrt_core <cont>{n}{blocks}: n even >= 4, top block >= 25000000 (block of 8 digits). Gives <cont>{s}{r} (natives), s = isqrt, r = X - s^2.

n = 4, 6: \CALX_nsqrt_iv, \CALX_nsqrt_vi; n >= 8: \CALX_nsqrt_k (one block steps). With <cont> = \CALX_nsqrt_un (padded input, see \CALX_nsqrtrem) and n = 6 or n >= 8, the root is not built as a native: \CALX_nsqrt_cvu passes the output of \CALX_nsqrt_vi to \CALX_nsqrt_kb, and the final state goes to \CALX_nsqrt_kuz (S and R of the unpadded input directly).

```
\def\CALX_nsqrt_core #1#2%
{%
    \ifcase\numexpr#2/\xint_c_ii-\xint_c_ii\relax
         \xint_dothis\CALX_nsqrt_civ
    \or  \xint_dothis{\ifx\CALX_nsqrt_un#1\expandafter\CALX_nsqrt_cvu
                      \else\expandafter\CALX_nsqrt_cvi\fi}%
    \fi\xint_orthat\CALX_nsqrt_k #1{#2}%
}%
\def\CALX_nsqrt_cvu #1#2#3{\CALX_nsqrt_msvi#3\CALX_nsqrt_vi\CALX_nsqrt_kb%
    \xint:{}#1}%
\def\CALX_nsqrt_civ #1#2#3{\CALX_nsqrt_msiv#3\CALX_nsqrt_iv\CALX_nsqrt_ivo#1}%
\def\CALX_nsqrt_msiv 1#1!1#2!1#3!1#4!#5#6{#5#6{#4}{#3}{#2}{#1}}%
\def\CALX_nsqrt_ivo #1#2#3#4%
   {\expandafter\CALX_nsqrt_ivp\romannumeral0\CALX_nsqrt_nathl{#3}{#4}%
       \xint:{N+2.#2!1#1!}}%
\def\CALX_nsqrt_ivp #1\xint:#2#3{#3{#2}{#1}}%
\def\CALX_nsqrt_cvi #1#2#3{\CALX_nsqrt_msvi#3\CALX_nsqrt_vi\CALX_nsqrt_vio#1}%
\def\CALX_nsqrt_msvi %
    1#1!1#2!1#3!1#4!1#5!1#6!#7#8{#7#8{#6}{#5}{#4}{#3}{#2}{#1}}%
\def\CALX_nsqrt_vio #1#2#3#4#5#6%
   {\expandafter\CALX_nsqrt_ivp\romannumeral0\CALX_nsqrt_natr{#4}{#5}{#6}%
       \xint:{N+3.#3!#2!1#1!}}%
```

30.5.6. \CALX_nsqrt_k

\CALX_nsqrt_k <cont>{n}{blocks}: n even >= 8, top block >= 25000000. Same output as \CALX_nsqrt_core. The root of the top six blocks is obtained by \CALX_nsqrt_vi, then the root is extended by one block for each further pair of blocks (one Zimmermann step with b = B each time).

Step. Let s (k blocks, k >= 3) be the root of the top 2k blocks, r = (top 2k blocks) - s^2 (0 <= r <= 2s), t = 2s, and a1, a0 the next two blocks. With W = r B + a1, the next block of the root is the largest q such that W B + a0 - q (t B + q) >= 0; it is q = floor(W/t) or that value minus one. Then s <- s B + q, t <- t B + 2q, r <- that difference. As s >= B^k/2 (normalized input), B^k <= t < 2 B^k: the top block t_k of t is 1.

State. P = 1t_0!1r_0!1t_1!1r_1!...1t_k!1r_k!1;;;;;! (the blocks of t and r interleaved, least significant first, r padded with zero blocks). Then L = 1<a1>!P is exactly the layout of the window W and the divisor t used by the multiply-subtract pass of the division (\CALX_ndms_a): the pairs (w_j, t_j) with w_0 = a1, w_j = r_(j-1), then w_(k+1) = r_k and the marker.

\CALX_nsqrt_k reverses the blocks (most significant first), gives the top six to \CALX_nsqrt_vi with continuation \CALX_nsqrt_kb, which builds P (k = 3) from its output. \CALX_nsqrt_ks {k}{P}

<pairs>\xint: loops while pairs of blocks remain, then goes to \CALX_nsqrt_kz. \CALX_nsqrt_kt copies L and skips 2k-4 blocks (\CALX_nskip) to read the top blocks of r and t for the estimate (\CALX_nsqrt_ku).
\CALX_nsqrt_kfl{x}: floor(x/B) for 0 <= x < 2B, in \numexpr.

```
\def\CALX_nsqrt_k #1#2#3%
   {\expandafter\CALX_nsqrt_ka\romannumeral0\CALX_rev_nounsep{}#3\R!\R!\R!\R!%
       \R!\R!\R!\R!\W
    \xint:{}#1}%
\def\CALX_nsqrt_ka 1#1!1#2!1#3!1#4!1#5!1#6!#7\xint:%
   {\CALX_nsqrt_vi\CALX_nsqrt_kb{#1}{#2}{#3}{#4}{#5}{#6}#7\xint:}%
\def\CALX_nsqrt_kfl #1{((#1+50000000)/\xint_c_e_viii-\xint_c_i)}%
\def\CALX_nsqrt_kb #1#2#3#4#5#6%
{%
    \expandafter\CALX_nsqrt_ks\expanded{{3}{%
    \the\numexpr\xint_c_ii*#3-\xint_c_e_viii
       -\CALX_nsqrt_kfl{\xint_c_ii*#3-200000000}*\xint_c_e_viii!%
    \the\numexpr\xint_c_e_viii+#6!%
    \the\numexpr\xint_c_ii*#2-\xint_c_e_viii+\CALX_nsqrt_kfl{%
        \xint_c_ii*#3-200000000}%
       -\CALX_nsqrt_kfl{\xint_c_ii*#2-200000000
            +\CALX_nsqrt_kfl{\xint_c_ii*#3-200000000}}*\xint_c_e_viii!%
    \the\numexpr\xint_c_e_viii+#5!%
    \the\numexpr\xint_c_ii*#1+\CALX_nsqrt_kfl{\xint_c_ii*#2-200000000
            +\CALX_nsqrt_kfl{\xint_c_ii*#3-200000000}}!%
    \the\numexpr#4-\CALX_nsqrt_kfl{#4}*\xint_c_e_viii+\xint_c_e_viii!%
    100000001!%
    \the\numexpr\CALX_nsqrt_kfl{#4}+\xint_c_e_viii!%
    1;;;;;!}}%
}%
\def\CALX_nsqrt_ks #1#2#3%
   {\ifx\xint:#3\expandafter\CALX_nsqrt_kz\else\expandafter\CALX_nsqrt_kt%
       \fi{#1}{#2}#3}%
\def\CALX_nsqrt_kt #1#21#3!1#4!%
   {\CALX_nskip\CALX_nsqrt_ku\xint_c_ii*#1-\xint_c_iv.1#3!#2{#1}{#3}{#4}{#2}}%
\def\CALX_nsqrt_ku 1#1!1#2!1#3!1#4!1#5!1#6!1#7!1;;;;;!#8#9%
{%
    \expandafter\CALX_nsqrt_kec\the\numexpr\xint_c_e_viii+#4\expandafter.%
    \the\numexpr#2\expandafter.\expandafter{\the\numexpr#7*\xint_c_e_viii+#5}%
    {#3}{{#8}{#9}}{#1}{}%
}%
```

30.5.7. \CALX_nsqrt_kec

Estimate. With U = (r_k B + r_(k-1)) B^2 + r_(k-2) B + r_(k-3) (three limbs, the top one below 2 10^8) and V = (B + t_(k-1)) B + t_(k-2), the value q* = floor(U/V) satisfies q <= q* <= q + 1 where q = floor(W/t), and q* = q + 1 only if the lower blocks matter (probability about 10^-8): W/t < (U + 1)/V and W/t >= U/(V + 1), and U/V - U/(V + 1) < 1. The computation of q* is that of \CALX_nsqrt_vi (same limbs, same bounds: tH = B + t_(k-1) is in [10^8, 2 10^8)): \CALX_nsqrt_kec ... _keksub are copies of \CALX_nsqrt_vi_c ... _vi_ksub, renamed, which end with \CALX_nsqrt_kq instead of \CALX_nsqrt_vi_r.

```
\def\CALX_nsqrt_kec #1.#2.#3#4#5#6#7%
   {\expandafter\CALX_nsqrt_ked\the\numexpr#3*%
```

```
        \xint_c_e_viii/#1.#1.#2.{#3}{#4}{#6}{#7}{#5}}%
\def\CALX_nsqrt_ked #1.#2.#3.%
{%
    \expandafter\CALX_nsqrt_kee
    \the\numexpr(#1+5000)/\xint_c_e_iv-\xint_c_i\expandafter.%
    \the\numexpr(#2+5000)/\xint_c_e_iv-\xint_c_i\expandafter.%
    \the\numexpr(#3+5000)/\xint_c_e_iv-\xint_c_i.#1.#2.#3.%
}%
\def\CALX_nsqrt_kee #1.#2.#3.#4.#5.#6.#7#8#9%
{%
    \expandafter\CALX_nsqrt_kef
    \the\numexpr#9-(#4-#1*\xint_c_e_iv)*(#6-#3*\xint_c_e_iv)\expandafter.%
    \the\numexpr-#1*(#6-#3*\xint_c_e_iv)-(#4-#1*\xint_c_e_iv)*#3\expandafter.%
    \the\numexpr((#7-#1*#2)*\xint_c_e_iv
       -(#1*(#5-#2*\xint_c_e_iv)+(#4-#1*\xint_c_e_iv)*#2))*\xint_c_e_iv
       +#8-(#1*#3+(#4-#1*\xint_c_e_iv)*(#5-#2*\xint_c_e_iv)).#4.#5.#6.%
}%
\def\CALX_nsqrt_kef #1.#2.%
{%
    \expandafter\CALX_nsqrt_keg
    \the\numexpr#2+(#1+500005000)/\xint_c_e_iv-50001\expandafter.%
    \the\numexpr#1-((#1+500005000)/\xint_c_e_iv-50001)*\xint_c_e_iv.%
}%
\def\CALX_nsqrt_keg #1.#2.#3.%
{%
    \expandafter\CALX_nsqrt_kek
    \the\numexpr#3+(#1+500005000)/\xint_c_e_iv-50001\expandafter.%
    \the\numexpr(#1-((#1+500005000)/\xint_c_e_iv-50001)*\xint_c_e_iv)*%
        \xint_c_e_iv+#2.%
}%
\def\CALX_nsqrt_kek #1.%
{%
    \ifnum#1<\xint_c_ \expandafter\CALX_nsqrt_kekadd
    \else\expandafter\CALX_nsqrt_kekcmp\fi #1.%
}%
\def\CALX_nsqrt_kekadd #1.#2.#3.#4.#5.%
{%
    \expandafter\CALX_nsqrt_kek
    \the\numexpr#1+#4+(#2+#5+50000000)/\xint_c_e_viii-\xint_c_i\expandafter.%
    \the\numexpr#2+#5-((#2+#5+50000000)/\xint_c_e_viii-\xint_c_i)*%
        \xint_c_e_viii
    \expandafter.\the\numexpr#3-\xint_c_i.#4.#5.%
}%
\def\CALX_nsqrt_kekcmp #1.#2.#3.#4.#5.%
{%
    \ifcase\numexpr\ifnum#1<#4 0\else\ifnum#1>#4 1\else
                    \ifnum#2<#5 0\else 1\fi\fi\fi\relax
         \expandafter\CALX_nsqrt_kq
    \or  \expandafter\CALX_nsqrt_keksub
    \fi #1.#2.#3.#4.#5.%
}%
\def\CALX_nsqrt_keksub #1.#2.#3.#4.#5.%
```

```
{%
    \expandafter\CALX_nsqrt_kek
    \the\numexpr#1-#4+(#2-#5+150000000)/\xint_c_e_viii-\xint_c_ii%
        \expandafter.%
    \the\numexpr#2-#5-((#2-#5+150000000)/\xint_c_e_viii-\xint_c_ii)*%
        \xint_c_e_viii
    \expandafter.\the\numexpr#3+\xint_c_i.#4.#5.%
}%
```

30.5.8. \CALX_nsqrt_kq

Multiply-subtract and new remainder. \CALX_nsqrt_kq clamps q* to B - 1 (the root block is at most B - 1, and W - (B - 1) t < B^(k+1) still fits the window), then \CALX_nsqrt_kqa runs \CALX_ndms_a on L with q*: the output is 1t_0!1u_0!...1t_k!1u_k!1;;;;;!{fin} with u = W - q* t. If fin < 0, t is added back (\CALX_ndab) and q* decreases by one (\CALX_nsqrt_kp, _kpa, _kpc).

\CALX_nsqrt_kw computes r' = u B + a0 - q^2 on the two low blocks: with qh, ql the halves of q and c = 2 qh ql = ch 10^4 + cl, the low block is a0 - ql^2 - cl 10^4 (borrow b0 in {-2, -1, 0}) and the next one u_0 - qh^2 - ch + b0 (borrow b1 in {-2, -1, 0}); further blocks are u_1, ..., u_k.

If b1 < 0 the borrow is propagated into u_1, u_2, ... (\CALX_nsqrt_kbs, \CALX_nsqrt_kbt). If it runs past u_k, r' < 0 (this happens for near squares, e.g. 10^48 + 10^24): the step is redone from \CALX_nsqrt_kqa with q - 1 (\CALX_nsqrt_kfb); then u >= t >= B^k and r' >= 0. The data of the step, {<a1>}{<a0>}{P}, is carried along for this purpose.

New state (\CALX_nsqrt_kwh): t' = t B + 2q has the blocks 2q - c B, t_0 + c (c = 1 if 2q >= B, else 0; t_0 is even, no carry), t_1, ..., t_k; r' has the blocks computed above, then u_1, ..., u_k. So P' = 1t'_0!1r'_0!1t'_1!1r'_1! followed by the output of the pass without its first two blocks.

```
\def\CALX_nsqrt_kq #1.#2.#3.#4.#5.#6#7#8#9%
{%
    \expandafter\CALX_nsqrt_kqa\the\numexpr
      \ifnum#3>99999999 199999999\else\xint_c_e_viii+#3\fi\xint:#7{#8}{#9}%
}%
\def\CALX_nsqrt_kqa 1#1#2#3#4#5\xint:#6#7#8#9%
{%
    \expandafter\CALX_nsqrt_kp\the\numexpr
    \CALX_ndms_a 0\xint:#1#2#3#4%
        \xint:#5!1#7!#9{#1#2#3#4#5}{#6}{#8}{{#7}{#8}{#9}}%
}%
\def\CALX_nsqrt_kp #1;;;;;!#2%
{%
    \ifnum\numexpr#2<\xint_c_ \xint_dothis\CALX_nsqrt_kpa\fi
    \xint_orthat\CALX_nsqrt_kw #1;;;;;!%
}%
\def\CALX_nsqrt_kpa #1;;;;;!%
   {\expandafter\CALX_nsqrt_kpc\the\numexpr\CALX_ndab 0\xint:#1;;;;;!}%
\def\CALX_nsqrt_kpc #1;;;;;!#2{\CALX_nsqrt_kw #1;;;;;!{#2-\xint_c_i}}%
\def\CALX_nsqrt_kw 1#1!1#2!#3;;;;;!#4#5#6#7%
{%
    \expandafter\CALX_nsqrt_kwa\the\numexpr#4\expandafter.%
    \the\numexpr(#4+5000)/\xint_c_e_iv-\xint_c_i.#1.#2.#6.{#5}{#7}#3;;;;;!%
}%
\def\CALX_nsqrt_kwa #1.#2.%
   {\expandafter\CALX_nsqrt_kwb\the\numexpr\xint_c_ii*#2*(#1-#2*%
       \xint_c_e_iv).#1.#2.}%
\def\CALX_nsqrt_kwb #1.%
```

```
    {\expandafter\CALX_nsqrt_kwc\the\numexpr(#1+5000)/\xint_c_e_iv-%
        \xint_c_i.#1.}%
\def\CALX_nsqrt_kwc #1.#2.#3.#4.#5.#6.#7.%
{%
    \expandafter\CALX_nsqrt_kwd
    \the\numexpr#7-(#3-#4*\xint_c_e_iv)*(#3-#4*\xint_c_e_iv)
                -(#2-#1*\xint_c_e_iv)*\xint_c_e_iv\expandafter.%
    \the\numexpr#6-#4*#4-#1.#3.#5.%
}%
\def\CALX_nsqrt_kwd #1.%
    {\expandafter\CALX_nsqrt_kwe\the\numexpr(#1+250000000)/\xint_c_e_viii-%
        \xint_c_iii.#1.}%
\def\CALX_nsqrt_kwe #1.#2.#3.%
    {\expandafter\CALX_nsqrt_kwf\the\numexpr#3+#1\expandafter.\the%
        \numexpr#2-#1*\xint_c_e_viii.}%
\def\CALX_nsqrt_kwf #1.%
    {\expandafter\CALX_nsqrt_kwg\the\numexpr(#1+150000000)/\xint_c_e_viii-%
        \xint_c_ii.#1.}%
\def\CALX_nsqrt_kwg #1.#2.#3.#4.#5.%
{%
    \ifnum#1=\xint_c_ \expandafter\CALX_nsqrt_kwh\else\expandafter%
        \CALX_nsqrt_kwp\fi
    {#1}{#2}{#3}{#4}{#5}%
}%
\def\CALX_nsqrt_kwp #1#2#3#4#5#6#7#8;;;;;!%
    {\expandafter\CALX_nsqrt_kwq\expanded{\CALX_nsqrt_kbs{#1}#8;;;;;!}%
        \xint:{#1}{#2}{#3}{#4}{#5}{#6}{#7}}%
\def\CALX_nsqrt_kwq #1;;;;;!#2\xint:%
    {\ifx\relax#2\relax\expandafter\CALX_nsqrt_kwk\else\expandafter%
        \CALX_nsqrt_kfb\fi{#1}}%
\def\CALX_nsqrt_kwk #1#2#3#4#5#6#7#8{%
    \CALX_nsqrt_kwh{#2}{#3}{#4}{#5}{#6}{#7}{#8}#1;;;;;!}%
\def\CALX_nsqrt_kfb #1#2#3#4#5#6#7#8%
    {\expandafter\CALX_nsqrt_kqa\the\numexpr\xint_c_e_viii+#5-\xint_c_i%
        \xint:{#7}#8}%
\def\CALX_nsqrt_kbs #11#2#3!{\xint_gob_til_sc #2\CALX_nsqrt_kbz;%
    \CALX_nsqrt_kbt{#1}{#2#3}}%
```

Attention to this F. It (probably) should be of catcode 12, so this is what we do.

```
\catcode`F 12
\def\CALX_nsqrt_kbz;\CALX_nsqrt_kbt#1#2{1;;;;;!F}%
\catcode`F 11
\def\CALX_nsqrt_kbt #1#21#3!%
{%
    1#2!\the\numexpr#3+#1+\ifnum\numexpr#3+#1<\xint_c_ 200000000\else%
        \xint_c_e_viii\fi!%
    \ifnum\numexpr#3+#1<\xint_c_ \CALX_nsqrt_kbf\fi
}%
\def\CALX_nsqrt_kbf\fi{\fi\CALX_nsqrt_kbs{-1}}%
\def\CALX_nsqrt_kwh #1#2#3#4#5#6#7#8;;;;;!%
{%
    \expandafter\CALX_nsqrt_ks\expanded{{\the\numexpr#6+\xint_c_i}{%
    \the\numexpr\xint_c_ii*#4-\CALX_nsqrt_kfl{\xint_c_ii*#4}*\xint_c_e_viii+%
```

```
        \xint_c_e_viii!%
    \the\numexpr#3+\xint_c_e_viii!%
    \the\numexpr#5+\CALX_nsqrt_kfl{\xint_c_ii*#4}+\xint_c_e_viii!%
    \the\numexpr#2-#1*\xint_c_e_viii+\xint_c_e_viii!%
    #8;;;;;!}}%
}%
```

30.5.9. \CALX_nsqrt_kz

End of the steps. \CALX_nsqrt_kzn: R is the native made of the r blocks of P (\CALX_nsqrt_kzr), S = t/2 (t blocks extracted by \CALX_nsqrt_kzt, halved by \CALX_nsqrt_kh: block t_i/2, plus 5 10^7 if t_(i+1) is odd).

\CALX_nsqrt_kuz is used instead when <cont> is \CALX_nsqrt_un, i.e. the core worked on Y = X D0^2, D0 = 2^j 10^t' (see \CALX_nsqrt_bgh): it receives P, then {D0}{j}{2t'}{5^j}{25^j}. With s, r the root and remainder of Y and m = 2^(j+1): one pass over t = 2s, most significant block first (\CALX_nsqrt_kva), divides by m (remainder c < m, so c B + t_i < 2^31) and shifts right by t' digits (remainder rho), giving the blocks of S = floor(s/D0) and, at the end, e = s - S D0 = rho 2^j + c/2. Then R = (r + e (t - e))/D0^2 (exact): native operations, \CALX_nshr by 2t' digits, then division by 4^j. floor(v/10^t') is (2v - 10^t' + 1)/(2 10^t') in \numexpr (no tie, also for t' = 0).

```
\def\CALX_nsqrt_kz #1#2\xint:#3#4%
{%
    \ifx\CALX_nsqrt_un#4\expandafter\CALX_nsqrt_kuz\else\expandafter%
        \CALX_nsqrt_kzn\fi
    {#2}#4%
}%
\def\CALX_nsqrt_kzn #1%
{%
    \expandafter\CALX_nsqrt_kzz\romannumeral0\expandafter\CALX_nnorm%
        \expandafter+\expanded{\CALX_nsqrt_kzr#1}1;!%
    \xint:{#1}%
}%
\def\CALX_nsqrt_kuz #1#2#3#4#5#6#7%
{%
    \expandafter\CALX_nsqrt_kub\expanded{%
      \expandafter\CALX_nsqrt_kvx\romannumeral0\expandafter\CALX_rev_nounsep%
          \expandafter{\expandafter}%
      \expanded{\CALX_nsqrt_kzt#1}\R!\R!\R!\R!\R!\R!\R!\W 1;!\xint:%
      {\ifcase#4 2\or4\or8\or16\fi}{\ifcase#4 1\or2\or4\or8\fi}{%
          \CALX_pten{#5/\xint_c_ii}}}%
    {#1}{#4}{#5}%
}%
\def\CALX_nsqrt_kvx #1\xint:#2#3#4{\CALX_nsqrt_kva 0.0.{#2}{#3}{#4}#1}%
\def\CALX_nsqrt_kva #1.#2.#3#4#51#6#7!%
{%
    \if;#6\expandafter\CALX_nsqrt_kve\else\expandafter\CALX_nsqrt_kvf\fi
    {#1}{#2}{#3}{#4}{#5}{#6#7}%
}%
\def\CALX_nsqrt_kve #1#2#3#4#5#6{\xint:{\the\numexpr#2*#4+#1/\xint_c_ii}}%
\def\CALX_nsqrt_kvf #1#2#3#4#5#6%
{%
    \expandafter\CALX_nsqrt_kvb
    \the\numexpr(#1*\xint_c_e_viii+#6+#4)/#3-\xint_c_i.#1.#2.#6.{#3}{#4}{#5}%
}%
```

```
\def\CALX_nsqrt_kvb #1.#2.#3.#4.#5#6#7%
{%
    \expandafter\CALX_nsqrt_kvc
    \the\numexpr(\xint_c_ii*#1-#7+\xint_c_i)/(\xint_c_ii*#7).#1.%
    \the\numexpr#2*\xint_c_e_viii+#4-#1*#5.#3.{#5}{#6}{#7}%
}%
\def\CALX_nsqrt_kvc #1.#2.#3.#4.#5#6#7%
{%
    \the\numexpr#1+#4*(\xint_c_e_viii/#7)+\xint_c_e_viii!%
    \expandafter\CALX_nsqrt_kva\the\numexpr#3\expandafter.\the%
        \numexpr#2-#1*#7.%
    {#5}{#6}{#7}%
}%
\def\CALX_nsqrt_kub #1\xint:%
{%
    \expandafter\CALX_nsqrt_kuc\romannumeral0\expandafter\CALX_nnorm%
        \expandafter+%
    \romannumeral0\CALX_rev_nounsep{}#1\R!\R!\R!\R!\R!\R!\R!\W 1;!%
    \xint:%
}%
\def\CALX_nsqrt_kuc #1\xint:#2#3#4#5%
{%
    \expandafter\CALX_nsqrt_kud\romannumeral0\CALX_nshr{#5}%
    {\romannumeral0\CALX_nadd
       {\romannumeral0\expandafter\CALX_nnorm\expandafter+\expanded{%
           \CALX_nsqrt_kzr#3}1;!}%
       {\romannumeral0\CALX_nmul{\romannumeral0\CALX_nsqrt_natv{#2}}%
          {\romannumeral0\CALX_nsub
             {\romannumeral0\expandafter\CALX_nnorm\expandafter+\expanded{%
                 \CALX_nsqrt_kzt#3}1;!}%
             {\romannumeral0\CALX_nsqrt_natv{#2}}}}}%
    \xint:{#1}{#4}%
}%
\def\CALX_nsqrt_kud #1\xint:#2#3%
{%
    \ifnum#3=\xint_c_ \xint_dothis{{#2}{#1}}\fi
    \xint_orthat{\expandafter\CALX_nsqrt_kue\romannumeral0\CALX_ndivtrunc{#1}%
      {\expanded{\ifcase#3\or N+1.100000004!\or N+1.100000016!\or %
          N+1.100000064!\fi}}%
      \xint:{#2}}%
}%
\def\CALX_nsqrt_kue #1\xint:#2{{#2}{#1}}%
\def\CALX_nsqrt_kzr 1#1#2!{\xint_gob_til_sc #1\CALX_nsqrt_kze;%
    \CALX_nsqrt_kzrb}%
\def\CALX_nsqrt_kze;\CALX_nsqrt_kzrb{}%
\def\CALX_nsqrt_kzrb 1#1!{1#1!\CALX_nsqrt_kzr}%
\def\CALX_nsqrt_kzt 1#1#2!{\xint_gob_til_sc #1\CALX_nsqrt_kzte;%
    \CALX_nsqrt_kztb 1#1#2!}%
\def\CALX_nsqrt_kzte;\CALX_nsqrt_kztb 1#1!{}%
\def\CALX_nsqrt_kztb 1#1!1#2!{1#1!\CALX_nsqrt_kzt}%
\def\CALX_nsqrt_kzz #1\xint:#2%
{%
```

```
    \expandafter\CALX_nsqrt_kzy\romannumeral0\expandafter\CALX_nnorm%
        \expandafter+%
    \expanded{\expandafter\CALX_nsqrt_kh\expanded{\CALX_nsqrt_kzt#2}1;!}1;!%
        \xint:{#1}%
}%
\def\CALX_nsqrt_kzy #1\xint:#2#3{#3{#1}{#2}}%
\def\CALX_nsqrt_kh 1#1!1#2#3!%
{%
    \the\numexpr\xint_c_e_viii+(#1\ifodd#1 -\xint_c_i\fi)/\xint_c_ii
    \xint_gob_til_sc#2\CALX_nsqrt_khe;\ifodd#2#3 +50000000\fi\relax!%
        \CALX_nsqrt_kh 1#2#3!%
}%
\def\CALX_nsqrt_khe;#1\CALX_nsqrt_kh 1;!{\relax!}%
```

30.5.10. \CALX_nsqrtrem

Syntax (use with \romannumeral0; X f-expanded native integer or I):
  \CALX_nsqrtrem{X} -> {S}{R} S = isqrt(X), R = X - S^2
\CALX_nsqrtrem{X} (X f-expands to a native integer; use with \romannumeral0): {S}{R}, S = isqrt(X), R = X - S^2. Negative X or I-: InvalidOperation, {I?}{I?} (trap value {N+0.}{N+0.}); I+: {I+}{I?}; I?: {I?}{I?} (no signal).
One or two blocks: \CALX_sqrt_start (from xint) on the digits. Three blocks or more: X has Dg digits; K = ceil(Dg/16) = (Dg+7)/16, n' = 2K blocks; X is shifted by 2t digits to Dp' = 16K - o digits (o = Dg mod 2), so the top block has 8 - o digits, then multiplied by 4^j (j minimal, j <= 1 if o = 0, j <= 3 if o = 1) so that it is >= 25000000; the multiplier is M = 10^(2t) 4^j. If M = 1: (S,R) = core(X). Else (s,r) = core(Y) and S = floor(s/(2^j 10^t)), R = X - S^2 (see \CALX_nsqrt_bgh). The leading 8 - o digits of X come from the top block v1 (d1 digits) and the next block v2: lead = floor((v1 B + v2)/10^e), e = d1 + o; floor(x/10^e) = (x+10^e/2)/10^e-1 (x >= 0).

```
\def\CALX_nsqrtrem #1{\expandafter\CALX_nsqrt_x\romannumeral`&&@#1\xint:}%
\def\CALX_nsqrt_x #1#2%
{%
    \if I#1\xint_dothis{\CALX_nsqrt_sp#2}\fi
    \if -#2\xint_dothis\CALX_nsqrt_inv\fi
    \xint_orthat\CALX_nsqrt_p
}%
\def\CALX_nsqrt_inv #1\xint:{\expandafter\CALX_ndiv_zpv\romannumeral`&&@%
    \CALX_signalcondition{InvalidOperation}{isqrt of negative or %
        special}{}{N+0.}\xint:}%
\def\CALX_nsqrt_p #1.%
{%
    \ifcase#1
         \xint_dothis\CALX_nsqrt_zero
    \or  \xint_dothis\CALX_nsqrt_pone
    \or  \xint_dothis\CALX_nsqrt_ptwo
    \fi\xint_orthat\CALX_nsqrt_big #1.%
}%
\def\CALX_nsqrt_sp #1\xint:%
{%
    \if +#1\xint_dothis{{I+}{I?}}\fi
    \if ?#1\xint_dothis{{I?}{I?}}\fi
    \xint_orthat{\CALX_nsqrt_inv\xint:}%
}%
\def\CALX_nsqrt_zero #1\xint:{{N+0.}{N+0.}}%
```

```
\def\CALX_nsqrt_pone #1.1#2!\xint:{\expandafter\CALX_nsqrt_onea\the%
    \numexpr#2.}%
\def\CALX_nsqrt_onea #1.%
   {\expandafter\CALX_nsqrt_sm\romannumeral0\expandafter\CALX_sqrt_start
    \the\numexpr\CALX_ndigs{#1}.#1.}%
\def\CALX_nsqrt_ptwo #1.1#2!1#3!\xint:{\expandafter\CALX_nsqrt_twoa\the%
    \numexpr#3.#2.}%
\def\CALX_nsqrt_twoa #1.#2.%
   {\expandafter\CALX_nsqrt_sm\romannumeral0\expandafter\CALX_sqrt_start
    \the\numexpr\xint_c_viii+\CALX_ndigs{#1}.#1#2.}%
\def\CALX_nsqrt_sm #1#2%
{%
    \expandafter\CALX_nsqrt_smb
    \the\numexpr\xint_c_ii*#1-\xint_c_i-#2\expandafter.%
    \the\numexpr#1-\xint_c_i+\xint_c_e_viii.%
}%
\def\CALX_nsqrt_smb #1.#2.%
   {\expandafter\CALX_nsqrt_ivp\romannumeral0\CALX_nsqrt_natv{#1}%
       \xint:{N+1.#2!}\CALX_nsqrt_out}%
\def\CALX_nsqrt_out #1#2{{#1}{#2}}%
\def\CALX_nsqrt_big #1.#2\xint:%
   {\CALX_nskip\CALX_nsqrt_bgt#1-\xint_c_ii.#2\xint:{#1}{#2}}%
\def\CALX_nsqrt_bgt 1#1!1#2!\xint:{\expandafter\CALX_nsqrt_bgv\the%
    \numexpr#2.#1.}%
\def\CALX_nsqrt_bgv #1.%
   {\expandafter\CALX_nsqrt_bga\the\numexpr\CALX_ndigs{#1}.#1.}%
\def\CALX_nsqrt_bga #1.#2.#3.#4%
   {\expandafter\CALX_nsqrt_bgb\the\numexpr\xint_c_viii*(#4-%
       \xint_c_i)+#1.#1.#2.#3.{#4}}%
\def\CALX_nsqrt_bgb #1.%
   {\expandafter\CALX_nsqrt_bgc\the\numexpr\ifodd#1 1\else 0\fi.#1.}%
\def\CALX_nsqrt_bgc #1.#2.#3.#4.#5.%
{%
    \expandafter\CALX_nsqrt_bgf\the\numexpr
    \ifcase\numexpr#3+#1\relax 0%
    \or #4*10000000+(#5+5)/10-\xint_c_i
    \or #4*1000000+(#5+50)/100-\xint_c_i
    \or #4*100000+(#5+500)/1000-\xint_c_i
    \or #4*10000+(#5+5000)/10000-\xint_c_i
    \or #4*1000+(#5+50000)/100000-\xint_c_i
    \or #4*100+(#5+500000)/1000000-\xint_c_i
    \or #4*10+(#5+5000000)/10000000-\xint_c_i
    \or #4%
    \or (#4+5)/10-\xint_c_i
    \fi.#1.#2.%
}%
\def\CALX_nsqrt_bgf #1.#2.#3.%
{%
    \expandafter\CALX_nsqrt_bgg\the\numexpr
    \ifnum#2=\xint_c_ \ifnum#1<25000000 1\else 0\fi
    \else\ifnum#1<1562500 3\else\ifnum#1<6250000 2\else 1\fi\fi\fi%
        \expandafter.%
```

```
    \the\numexpr16*((#3+\xint_c_vii)/16)-#2-#3\expandafter.%
    \the\numexpr\xint_c_ii*((#3+\xint_c_vii)/16).%
}%
\def\CALX_nsqrt_bgg #1.#2.%
{%
    \ifnum\numexpr#1+#2=\xint_c_
      \expandafter\CALX_nsqrt_bgg_a
    \else
      \expandafter\CALX_nsqrt_bgg_b
      \the\numexpr\ifcase#1 1\or 2\or 4\or 8\fi*\CALX_pten{#2/%
          \xint_c_ii}%
                  +\xint_c_e_viii\expandafter.%
      \the\numexpr\ifcase#1 1\or 4\or 16\or 64\fi+\xint_c_e_viii\expandafter.%
      \the\numexpr\ifcase#1 1\or 5\or 25\or 125\fi+\xint_c_e_viii%
          \expandafter.%
      \the\numexpr\ifcase#1 1\or 25\or 625\or 15625\fi+\xint_c_e_viii\expandafter;%
    \fi
      #1.#2.%
}%
\def\CALX_nsqrt_bgg_a #1.#2.#3.#4#5{\CALX_nsqrt_core\CALX_nsqrt_out{#3}{#5}}%
\def\CALX_nsqrt_bgg_b #1;#2.#3.#4.#5#6{\CALX_nsqrt_bgh #2.#3.#1.{N+#5.#6}}%
```

\CALX_nsqrt_bgh <j>.<2t>.<D0>.<4^j>.<5^j>.<25^j>.{X}: the four numbers after j and 2t are given as 1ddddddddd (value + 10^8), D0 = 2^j 10^t.
  Y = X 10^(2t) 4^j = X D0^2 (\CALX_nshl, then one one block multiplication if j > 0). With (s,r) = core(Y), no division at all (1/D0 = 5^j/10^(t+j)):
     S = floor(s 5^j / 10^(t+j))   (one block mul + \CALX_nshr),
     e = s - S D0   (one block mul + \CALX_nsub),
     R = (r + e(2s - e)) 25^j / 10^(2t+2j)   (exact \CALX_nshr),
as Y = s^2 + r = X D0^2 and X - S^2 = (r + e(2s-e))/D0^2. The multiplications by 5^j, 25^j are skipped when j = 0.

```
\def\CALX_nsqrt_bgh #1#2.%
{%
    \expandafter\CALX_nsqrt_bgi\romannumeral0%
    \if0#1\expandafter\CALX_nsqrt_bgh_a\else\expandafter\CALX_nsqrt_bgh_b
    \fi #1#2.%
}%
\def\CALX_nsqrt_bgh_a #1.#2.#3.#4.#5.#6.#7%
{%
    \CALX_nshl{#2}{#7}\xint:{#3}{#1}{#2}{#5}{#6}%
}%
\def\CALX_nsqrt_bgh_b #1.#2.#3.#4.#5.#6.#7%
{%
    \CALX_nmul{\romannumeral0\CALX_nshl{#2}{#7}}{N+1.#4!}%
    \xint:{#3}{#1}{#2}{#5}{#6}%
}%
\def\CALX_nsqrt_bgi N+#1.#2\xint:{\CALX_nsqrt_core\CALX_nsqrt_un{#1}{#2}}%
\def\CALX_nsqrt_un #1#2#3#4#5#6%
{%
    \expandafter\CALX_nsqrt_uo\romannumeral0\CALX_nshr{#5/\xint_c_ii+#4}%
      {\ifnum#4=\xint_c_ \expandafter\xint_firstoftwo\else
       \expandafter\xint_secondoftwo\fi
       {#1}{\romannumeral0\CALX_nmul{#1}{N+1.#6!}}}%
```

```
            \xint:{#1}{#2}{#3}{#4}{#5}%
}%
\def\CALX_nsqrt_uo #1\xint:#2#3#4%
{%
    \expandafter\CALX_nsqrt_up
    \romannumeral0\CALX_nsub{#2}{\romannumeral0\CALX_nmul{#1}{N+1.#4!}}%
        \xint:{#1}{#2}{#3}%
}%
\def\CALX_nsqrt_up #1\xint:#2#3#4%
{%
    \expandafter\CALX_nsqrt_uq\romannumeral0\CALX_nadd{#4}%
      {\romannumeral0\CALX_nmul{#1}{\romannumeral0\CALX_nsub
                                    {\romannumeral0\CALX_nadd{#3}{#3}}{#1}}}%
                                        \xint:{#2}%
}%
\def\CALX_nsqrt_uq #1\xint:#2#3#4#5%
{%
    \expandafter\CALX_nsqrt_ur\romannumeral0\CALX_nshr{#4+\xint_c_ii*#3}%
      {\ifnum#3=\xint_c_ \expandafter\xint_firstoftwo\else
       \expandafter\xint_secondoftwo\fi
       {#1}{\romannumeral0\CALX_nmul{#1}{N+1.#5!}}}\xint:{#2}%
}%
\def\CALX_nsqrt_ur #1\xint:#2{{#2}{#1}}%
```

30.6. Decimal shifts

30.6.1. \CALX_nshr, \CALX_nshl, \CALX_nshrr

Syntax (use with \romannumeral0; k a \numexpr expression, X f-expanded native integer or I):
  \CALX_nshr{k}{X} -> <space>Y Y = sign(X) floor(|X|/10^k)
    \CALX_nshl{k}{X}   -> <space>Y   Y = X 10^k
    \CALX_nshrr{k}{X} -> <space>Y   Y = sign(X) round(|X|/10^k)
Decimal shifts of native integers (analogues of \xintDSH, \xintDSx, \xintDSRr). Use with \romannumeral0; arguments f-expanded.
  \CALX_nshr{k}{X}: sign(X) floor(|X|/10^k) (truncation towards zero).
  \CALX_nshl{k}{X}: X 10^k. \CALX_nshrr{k}{X}: |X|/10^k rounded half away from zero, sign kept. k >= 0 (k <= 0: X unchanged).
  k = 8a + c. Whole blocks: \CALX_nskip (right) or prepended zero blocks (left). c > 0: one pass inside \expanded (generated macros \CALX_nshr_f<c>, \CALX_nshr_l<c>, \CALX_nshr_e<c>, \CALX_nshl_l<c>, \CALX_nshl_e<c>): each new block is a token rearrangement of two consecutive blocks, the pending part being the macro argument; no arithmetic. End marker 1;;;;;;;;!. The pass ends with \relax<0|1> telling whether a top block was dropped (right) or added (left); the header N<s><n'>. is built after the pass (one grab of the result). I+, I-, I? are returned unchanged (\CALX_nsh_t, one \if test).
  \CALX_nshrr: first dropped digit = digit p = (k-1) mod 8 of block (k-1) div 8 = (2k-9)/16 (\numexpr, k >= 1), read by \CALX_nskip; +1 by \CALX_nadd if >= 5. A result with no block left is N+0.

```
\def\CALX_nsh_t #1#2.#3{\if I#3\xint_dothis\CALX_nsh_sp\fi%
    \xint_orthat{#1#2.#3}}%
\def\CALX_nsh_sp #1\xint:{ I#1}%
\def\CALX_nsh_zero\fi#1\xint:{\fi\CALX_sp N+0.}%
\def\CALX_nsh_id\fi#1.#2.#3.#4\xint:{\fi\CALX_sp N#3.#4}%
\def\CALX_nshr #1#2{\expandafter\CALX_nsh_t\expandafter\CALX_nshr_a\the%
```

```
    \numexpr#1\expandafter.\romannumeral`&&@#2\xint:}%
\def\CALX_nshr_a #1.N#2#3.#4\xint:%
{%
    \ifnum#3=\xint_c_ \CALX_nsh_zero\fi
    \ifnum#1<\xint_c_i \CALX_nsh_id\fi
    \expandafter\CALX_nshr_b\the\numexpr(#1-\xint_c_iv)/\xint_c_viii.%
    #1.#2#3.#4\xint:
}%
\def\CALX_nshr_b #1.#2.#3#4.#5\xint:%
{%
    \ifnum#1<#4 \xint_dothis{\expandafter\CALX_nshr_c\the\numexpr#2-%
        \xint_c_viii*#1.#1.#3#4.{#5}}\fi
    \xint_orthat{ N+0.}%
}%
\def\CALX_nshr_c #1.#2.#3#4.#5%
{%
    \ifnum#1=\xint_c_
      \xint_dothis{\CALX_nskip\CALX_nshr_z#2.#51;;;;;;;;!\xint:{#3}{#4-#2}}\fi
    \xint_orthat{\expandafter\CALX_nshr_o\expanded{\expandafter\CALX_nskip
      \csname CALX_nshr_f#1\endcsname#2.#51;;;;;;;;!}%
          \xint:{#3}{#4-#2}}%
}%
\def\CALX_nshr_z #11;;;;;;;;!\xint:#2#3{\expanded{ N#2\the\numexpr#3.}#1}%
\def\CALX_nshr_o #1\relax#2\xint:#3#4%
{%
    \ifnum\numexpr#4-#2=\xint_c_ \xint_dothis{ N+0.}\fi
    \xint_orthat{\expanded{ N#3\the\numexpr#4-#2.}#1}%
}%
\def\CALX_nshl #1#2{\expandafter\CALX_nsh_t\expandafter\CALX_nshl_a\the%
    \numexpr#1\expandafter.\romannumeral`&&@#2\xint:}%
\def\CALX_nshl_a #1.N#2#3.#4\xint:%
{%
    \ifnum#3=\xint_c_ \CALX_nsh_zero\fi
    \ifnum#1<\xint_c_i \CALX_nsh_id\fi
    \expandafter\CALX_nshl_b\the\numexpr(#1-\xint_c_iv)/\xint_c_viii.%
    #1.#2#3.#4\xint:
}%
\def\CALX_nshl_b #1.#2.#3#4.#5\xint:%
   {\expandafter\CALX_nshl_c\the\numexpr#2-\xint_c_viii*#1.#1.#3#4.#5\xint:}%
\def\CALX_nshl_c #1.#2.#3#4.#5\xint:%
{%
    \ifnum#1=\xint_c_
      \xint_dothis{\expanded{ N#3\the\numexpr#4+#2.\romannumeral%
          \xintreplicate{#2}{100000000!}}#5}\fi
    \xint_orthat{\expandafter\CALX_nshl_o\expanded{%
      \romannumeral\xintreplicate{#2}{100000000!}%
      \csname CALX_nshl_l#1\endcsname{\romannumeral%
          \xintreplicate{#1}0}%
      #51;;;;;;;;!}\xint:{#3}{#4+#2}}%
}%
\def\CALX_nshl_o #1\relax#2\xint:#3#4{\expanded{ N#3\the\numexpr#4+#2.}#1}%
\def\CALX_nshrr #1#2{\expandafter\CALX_nsh_t\expandafter\CALX_nshrr_a\the%
```

```
    \numexpr#1\expandafter.\romannumeral`&&@#2\xint:}%
\def\CALX_nshrr_a #1.N#2#3.#4\xint:%
{%
    \ifnum#1<\xint_c_i \xint_dothis{ N#2#3.#4}\fi
    \ifnum#3=\xint_c_ \xint_dothis{ N+0.}\fi
    \ifnum#1>\numexpr\xint_c_viii*#3\relax \xint_dothis{ N+0.}\fi
    \xint_orthat{\expandafter\CALX_nshrr_b\the\numexpr
      \CALX_nskip\CALX_nshrr_d(\xint_c_ii*#1-9)/16.#41;!%
      #1-\xint_c_i-\xint_c_viii*((\xint_c_ii*#1-9)/16).%
      #1.N#2#3.#4\xint:}%
}%
```

`\CALX_nshrr_d 1<block>!<rest>1;!<p>.` (inside `\numexpr`) gives the digit p from the right of the block followed by a full stop; on the sentinel 1;! (block beyond the number) it gives 0. With y = 10^p, floor(b/y) = (b+y/2)/y-1 for p > 0 (y even), q mod 10 = q-10((q+5)/10-1).

```
\def\CALX_nshrr_d 1#1!{\xint_gob_til_sc#1\CALX_nshrr_dz;\CALX_nshrr_dn{#1}}%
\def\CALX_nshrr_dz;\CALX_nshrr_dn#1#2.{0.}%
\def\CALX_nshrr_dn #1#21;!#3.%
{%
    \ifnum\numexpr#3=\xint_c_
      \xint_dothis{(#1-((#1+5)/10-1)*10).}\fi
    \xint_orthat{\CALX_nshrr_dq{\CALX_pten{#3}}{#1}}%
}%
\def\CALX_nshrr_dq #1#2%
   {((((#2+#1/2)/#1-1)-((((#2+#1/2)/#1-1)+5)/10-1)*10).}%
\def\CALX_nshrr_b #1.#2.#3\xint:%
{%
    \ifnum#1>\xint_c_iv
      \xint_dothis{\expandafter\CALX_nshrr_up\romannumeral0\CALX_nshr{#2}{#3}%
          \xint:#3\xint:}\fi
    \xint_orthat{\CALX_nshr{#2}{#3}}%
}%
\def\CALX_nshrr_up N#1#2\xint:N#3#4\xint:%
   {\CALX_nadd{N#1#2}{N#31.100000001!}}%
\expandafter\def\csname CALX_nshr_f1\endcsname
   1#1#2#3#4#5#6#7#8!{\CALX_nshr_li{#1#2#3#4#5#6#7}}%
\def\CALX_nshr_li #11#2#3#4#5#6#7#8#9!%
   {\xint_gob_til_sc#2\CALX_nshr_ei;1#9#1!\CALX_nshr_li{#2#3#4#5#6#7#8}}%
\def\CALX_nshr_ei;1;#1!#2#3%
   {\ifnum#1=\xint_c_ \relax1\else 10#1!\relax0\fi}%
\def\CALX_nshl_li #11#2#3#4#5#6#7#8#9!%
   {\xint_gob_til_sc#2\CALX_nshl_ei;1#3#4#5#6#7#8#9#1!\CALX_nshl_li{#2}}%
\def\CALX_nshl_ei;1;;;;;;;#1!#2#3%
   {\ifnum#1=\xint_c_ \relax0\else 10000000#1!\relax1\fi}%
\expandafter\def\csname CALX_nshr_f2\endcsname
   1#1#2#3#4#5#6#7#8!{\CALX_nshr_lii{#1#2#3#4#5#6}}%
\def\CALX_nshr_lii #11#2#3#4#5#6#7#8#9!%
   {\xint_gob_til_sc#2\CALX_nshr_eii;1#8#9#1!\CALX_nshr_lii{#2#3#4#5#6#7}}%
\def\CALX_nshr_eii;1;;#1!#2#3%
   {\ifnum#1=\xint_c_ \relax1\else 100#1!\relax0\fi}%
\def\CALX_nshl_lii #11#2#3#4#5#6#7#8#9!%
   {\xint_gob_til_sc#2\CALX_nshl_eii;1#4#5#6#7#8#9#1!\CALX_nshl_lii{#2#3}}%
\def\CALX_nshl_eii;1;;;;;;#1!#2#3%
```

```
    {\ifnum#1=\xint_c_ \relax0\else 1000000#1!\relax1\fi}%
\expandafter\def\csname CALX_nshr_f3\endcsname
    1#1#2#3#4#5#6#7#8!{\CALX_nshr_liii{#1#2#3#4#5}}%
\def\CALX_nshr_liii #11#2#3#4#5#6#7#8#9!%
    {\xint_gob_til_sc#2\CALX_nshr_eiii;1#7#8#9#1!\CALX_nshr_liii{#2#3#4#5#6}}%
\def\CALX_nshr_eiii;1;;;#1!#2#3%
    {\ifnum#1=\xint_c_ \relax1\else 1000#1!\relax0\fi}%
\def\CALX_nshl_liii #11#2#3#4#5#6#7#8#9!%
    {\xint_gob_til_sc#2\CALX_nshl_eiii;1#5#6#7#8#9#1!\CALX_nshl_liii{#2#3#4}}%
\def\CALX_nshl_eiii;1;;;;;#1!#2#3%
    {\ifnum#1=\xint_c_ \relax0\else 100000#1!\relax1\fi}%
\expandafter\def\csname CALX_nshr_f4\endcsname
    1#1#2#3#4#5#6#7#8!{\CALX_nshr_liv{#1#2#3#4}}%
\def\CALX_nshr_liv #11#2#3#4#5#6#7#8#9!%
    {\xint_gob_til_sc#2\CALX_nshr_eiv;1#6#7#8#9#1!\CALX_nshr_liv{#2#3#4#5}}%
\def\CALX_nshr_eiv;1;;;;#1!#2#3%
    {\ifnum#1=\xint_c_ \relax1\else 10000#1!\relax0\fi}%
\def\CALX_nshl_liv #11#2#3#4#5#6#7#8#9!%
    {\xint_gob_til_sc#2\CALX_nshl_eiv;1#6#7#8#9#1!\CALX_nshl_liv{#2#3#4#5}}%
\def\CALX_nshl_eiv;1;;;;#1!#2#3%
    {\ifnum#1=\xint_c_ \relax0\else 10000#1!\relax1\fi}%
\expandafter\def\csname CALX_nshr_f5\endcsname
    1#1#2#3#4#5#6#7#8!{\CALX_nshr_lv{#1#2#3}}%
\def\CALX_nshr_lv #11#2#3#4#5#6#7#8#9!%
    {\xint_gob_til_sc#2\CALX_nshr_ev;1#5#6#7#8#9#1!\CALX_nshr_lv{#2#3#4}}%
\def\CALX_nshr_ev;1;;;;;#1!#2#3%
    {\ifnum#1=\xint_c_ \relax1\else 100000#1!\relax0\fi}%
\def\CALX_nshl_lv #11#2#3#4#5#6#7#8#9!%
    {\xint_gob_til_sc#2\CALX_nshl_ev;1#7#8#9#1!\CALX_nshl_lv{#2#3#4#5#6}}%
\def\CALX_nshl_ev;1;;;#1!#2#3%
    {\ifnum#1=\xint_c_ \relax0\else 1000#1!\relax1\fi}%
\expandafter\def\csname CALX_nshr_f6\endcsname
    1#1#2#3#4#5#6#7#8!{\CALX_nshr_lvi{#1#2}}%
\def\CALX_nshr_lvi #11#2#3#4#5#6#7#8#9!%
    {\xint_gob_til_sc#2\CALX_nshr_evi;1#4#5#6#7#8#9#1!\CALX_nshr_lvi{#2#3}}%
\def\CALX_nshr_evi;1;;;;;;#1!#2#3%
    {\ifnum#1=\xint_c_ \relax1\else 1000000#1!\relax0\fi}%
\def\CALX_nshl_lvi #11#2#3#4#5#6#7#8#9!%
    {\xint_gob_til_sc#2\CALX_nshl_evi;1#8#9#1!\CALX_nshl_lvi{#2#3#4#5#6#7}}%
\def\CALX_nshl_evi;1;;#1!#2#3%
    {\ifnum#1=\xint_c_ \relax0\else 100#1!\relax1\fi}%
\expandafter\def\csname CALX_nshr_f7\endcsname
    1#1#2#3#4#5#6#7#8!{\CALX_nshr_lvii{#1}}%
\def\CALX_nshr_lvii #11#2#3#4#5#6#7#8#9!%
    {\xint_gob_til_sc#2\CALX_nshr_evii;1#3#4#5#6#7#8#9#1!\CALX_nshr_lvii{#2}}%
\def\CALX_nshr_evii;1;;;;;;;#1!#2#3%
    {\ifnum#1=\xint_c_ \relax1\else 10000000#1!\relax0\fi}%
\def\CALX_nshl_lvii #11#2#3#4#5#6#7#8#9!%
    {\xint_gob_til_sc#2\CALX_nshl_evii;1#9#1!\CALX_nshl_lvii{#2#3#4#5#6#7#8}}%
\def\CALX_nshl_evii;1;#1!#2#3%
    {\ifnum#1=\xint_c_ \relax0\else 10#1!\relax1\fi}%
\expandafter\let\csname CALX_nshl_l1\endcsname\CALX_nshl_li
```

```
\expandafter\let\csname CALX_nshl_l2\endcsname\CALX_nshl_lii
\expandafter\let\csname CALX_nshl_l3\endcsname\CALX_nshl_liii
\expandafter\let\csname CALX_nshl_l4\endcsname\CALX_nshl_liv
\expandafter\let\csname CALX_nshl_l5\endcsname\CALX_nshl_lv
\expandafter\let\csname CALX_nshl_l6\endcsname\CALX_nshl_lvi
\expandafter\let\csname CALX_nshl_l7\endcsname\CALX_nshl_lvii
```

30.7. Interface for xintexpr usage

Attention: N and I are STILL of catcode 12. We now also make M of catcode 12. Be extra careful with macro names.

The integer type of \xintiieval: M<s><n>.<e>.<blocks>, the value of the N value N<s><n>.<blocks> (n blocks 1ddddddd!, least significant first) times 10^(8e), e >= 0, lowest block non-zero; zero is M+0.0. \CALX_ntom: N value (blocks in the stream) to M, the low zero blocks go into e. \CALX_mfromraw: raw integer (xintcore format) to M. \CALX_mtoraw: M to raw integer, 8e zeros written out. \CALX_ntomd d.N...: as \CALX_ntom, e starting at d; \CALX_ntoms: same, special values (I...) passed as they are.

Syntax of the entry points (A, B, X f-expanded; M, I values, or raw integers converted on entry; R a raw integer):

```
\CALX_ntom N<s><n>.<blocks> -> <space>M<s><n'>.<e>.<blocks'>
  \CALX_ntomd d.N<s><n>.<blocks>    -> the same, e counted from d
  \CALX_mfromraw{R}                 -> <space>M...
  \CALX_mtoraw{X}                   -> <raw integer>    (X an M value)
  \CALX_tom{X}                     -> <space>M...\xint: or <space>I<s>\xint:
  \CALX_mr{X}                       -> <raw integer> (X M, A, A[N] N >= 0),
                                         X otherwise (I, A[N] N < 0)
  \CALX_mrl <item>...<item>^        -> {\CALX_mr{<item>}}...  (in \expanded)
  \CALX_ii<op>{A}{B}, \CALX_ii<op>{X}  -> <space>M... or <space>I<s>
  \calxii<Name>{A}{B}, \calxii<Name>{X} -> M... or I<s>   (f-expandable)
  \CALX_iilit{A}{N}                 -> <space>M..., or <space>A[N]
```

\CALX_ntom, \CALX_ntomd, \CALX_mfromraw, \CALX_mtoraw, \CALX_tom and the \CALX_ii<op> are used with \romannumeral0. The comparisons \calxiiLt, \calxiiGt, \calxiiLtorEq, \calxiiGtorEq, \calxiiEq, \calxiiNotEq give raw 1 or 0; \calxiiDivMod gives {Q}{R}; \calxiiPow{A}{k}.

M layer (used as \romannumeral0\CALX_ii<op>{A}{B}, results M or I): \CALX_tom{X}: X f-expanded; M left as it is (the tag re-emitted with the catcode of this block: a value compiled in a user function may carry another one), raw integer converted; A[N] with N >= 0: A followed by N zeros (\CALX_tom_w); A[N] with N < 0 and A/B[N]: \CALX_tom_x, integer check (error, then truncation) by one native division (\CALX_iiq: A 10^max(N,0) by B 10^max(-N,0), B > 0; an integer iff the remainder is zero; truncation = floor quotient, plus one if negative); I+, I-, I? left as they are; \xint: follows. \CALX_iitwo<cont><spcont>{A}{B} hands M(A)\xint:M(B)\xint: to <cont>, or the same with an I value to <spcont> (tag test \CALX_iitwo_b). Special values propagate as in the N kernels (rules of calxcore): \CALX_iisp_g<fin><kernel> calls the N kernel on the operands, M ones converted with their e zero blocks written out (\CALX_mton), result through \CALX_ntoms (I kept). Unary ops: \CALX_iiun<Mcont><Icont>; opp, abs, sqr act on the I sign, isqrt calls \CALX_nsqrtrem, ^ with an I base: k = 0 gives 1, I? stays, k < 0 gives 0, else I+ or I- (odd k, I- base). Comparisons with an I? operand: InvalidOperation, 0 (== is false, != true, no signal); infinities ordered by \CALX_icl<tag><sign>. mul, sqr: e1+e2 (2e), N kernel \CALX_nmul. add, sub and the division family (divtrunc, modtrunc, divfloor, mod, quo, rem, divround, division, divmod, kernels \CALX_n... of calxcore): with d = min(e1,e2) the operand of larger e gets e - d zero blocks; the quotient has e = 0, a sum or remainder e = d. opp, abs: sign token. isqrt: all e zero blocks written out (\CALX_nsqrtrem), result e = 0.

\xintiieval. Literals (receivers of the scanner's fake operators): digits to M (\CALX_mfromraw); with [N] (decimal mark, exponent, raw [N]): \CALX_iilit, M if the value is an integer, else the

raw A[N] (for num(), round(), trunc()); in compiled functions (\XINT_expr_redefinemacros) literals stay raw. Operators and functions: \calxii<Name>, one token each (needed by the \xintdeffunc hooks). Native: Add, Sub, Mul, Sqr, Opp, Abs, Sqrt, DivFloor (//), Mod (/:), DivRound (/), Quo, Rem, DivMod, Pow and the comparisons. Pow (^): k = \numexpr of the exponent, \CALX_npow on the blocks only, result exponent e k; k < 0 (\CALX_ipr<class of the base>): 1 for 1, (-1)^k for -1, DivisionByZero (I+) for 0, Underflow (0) otherwise. Lt, Gt, LtorEq, GtorEq: keys s (n + e) (s the sign, 0 for zero) compared first; equal keys: the blocks if n = 1 for both, else the sign of the difference (\CALX_iisub_x); the three result digits for equal, greater, less follow the arguments. Eq, NotEq: \xintstrcmp of the canonical M values. Comparisons give raw 1 or 0. ^ with an I exponent: IEEE pow (\CALX_iipow_e, table \CALX_ipe<sign of exponent><class of base>, classes o (1), m (-1), z (0), b (other), i (I+, I-), n (I?)). Integer functions on M values (an I argument signals InvalidOperation, result I?): Fac (\CALX_nfac; negative or > 10^4: InvalidOperation), Odd, Even (parity of the lowest block if e = 0), IsOne, ifOne (an I argument selects the second branch), LogTen (8(n+e) - 9 + digits of the top block; zero: -"7FFF8000 as xint), SqrtR (\CALX_nsqrtrem: S + 1 iff R > S). On the \numexpr values of raw arguments (\CALX_mrd; \CALX_mr: M to raw digits, other values as they are): Binomial (\CALX_binom_a, native, by \CALX_ntom), PFactorial (\CALX_npfac), with xint's argument rules and InvalidOperation for the out of range cases. Raw output from xint macros: Round, Trunc (digit count), SeqB:tl:x (a..[d]..b). Lists: gcd, lcm: Burnol's \XINT_iigcdof/\XINT_iilcmof loops (delimiter \xint: instead of !, which occurs in M values) on \CALX_iigcd, \CALX_iilcm, of the native \CALX_ngcd, \CALX_nlcm (Euclid's algorithm by \CALX_nrem; the M versions factor out the common power 10^(8 min(e1,e2))); an I item: InvalidOperation, I?. `+`, `*`: Burnol's \XINT_sum/\XINT_prd loops (xintfrac) on \CALX_iiadd, \CALX_iimul (empty: 0, 1); max, min: Burnol's \XINT_iimaxof/\XINT_iiminof loops on \CALX_iimax, \CALX_iimin (the native comparisons; each item f-expanded once, the values returned expanded), an unordered comparison (I?) signals InvalidOperation and keeps the current value (as \calxFpMaxof). Printers: \xintiiexprPrintOne(Hex/Oct/Bin) through \CALX_mr.

```
\catcode`M=12
\def\CALX_ntom {\CALX_ntomd 0.}%
\def\CALX_ntomd #1.N#2#3.%
{%
    \ifnum#3=\xint_c_ \xint_dothis{ M+0.0.}\fi
    \xint_orthat{\CALX_ntom_a#2#3.#1.}%
}%
\def\CALX_ntom_a #1#2.#3.#4!%
{%
    \ifnum#4=\xint_c_e_viii
      \xint_dothis{\expandafter\CALX_ntom_a\expandafter#1%
        \the\numexpr#2-\xint_c_i\expandafter.\the\numexpr#3+\xint_c_i.}\fi
    \xint_orthat{ M#1#2.#3.#4!}%
}%
\def\CALX_mfromraw%
   {\expandafter\CALX_ntom\romannumeral0\CALX_nfromraw}%
\def\CALX_mtoraw #1{\expandafter\CALX_mtoraw_t\romannumeral`&&@#1\xint:}%
\def\CALX_mtoraw_t #1{\CALX_mtoraw_a M}%
\def\CALX_mtoraw_a M#1#2.#3.%
{%
    \ifnum#3=\xint_c_ \xint_dothis{\CALX_ntoraw_a N#1#2.}\fi
    \xint_orthat{\CALX_mtoraw_b{#3}N#1#2.}%
}%
\def\CALX_mtoraw_b #1#2\xint:%
   {\expanded{ \romannumeral0\CALX_ntoraw_a#2\xint:%
               \xintReplicate{\xint_c_viii*#1}0}}%
```

```
\def\CALX_infr_m #1{\CALX_infr_b #1\xint:}%
\def\CALX_infr_b #1#2#3.#4.#5\xint:%
   {\expandafter\CALX_infr_c\the\numexpr\xint_c_viii*#4\expandafter.%
    \romannumeral0\CALX_ntoraw_a N#2#3.#5\xint:.}%
\def\CALX_infr_c #1.#2.{{#1}{#2}{1}}%
\def\CALX_ntoms #1.#2%
   {\if I#2\xint_dothis{ #2}\fi\xint_orthat{\CALX_ntomd#1.#2}}%
\def\CALX_tom #1{\expandafter\CALX_tom_a\romannumeral`&&@#1\xint:}%
\def\CALX_tom_a #1%
   {\if M#1\xint_dothis{ M}\fi\if I#1\xint_dothis{ #1}\fi
    \xint_orthat{\CALX_tom_r#1}}%
\def\CALX_tom_r #1\xint:{\CALX_tom_s #1[.\xint:}%
\def\CALX_tom_s #1[#2{\xint_gob_til_dot#2\CALX_tom_b.\CALX_tom_k #2#1[}%
\def\CALX_tom_b.\CALX_tom_k.#1[\xint:%
   {\expandafter\CALX_ntom\romannumeral0\CALX_nfromraw_a#1\xint:\xint:}%
\def\CALX_tom_k #1%
{%
    \xint_UDsignfork
        #1\CALX_tom_neg
         -\CALX_tom_p
    \krof #1%
}%
\def\CALX_tom_neg -#1[#2][.\xint:{\CALX_tom_z #1/1/\xint:{-#2}{#1[-#2]}}%
\def\CALX_tom_p #1#2[#3][.\xint:{\CALX_tom_v #1#3.#2/\xint:}%
\def\CALX_tom_v #1.#2/#3%
   {\xint_gob_til_xint:#3\CALX_tom_w\xint:\CALX_tom_x #1.#2/#3}%
\def\CALX_tom_w\xint:\CALX_tom_x #1.#2/\xint:%
   {\expandafter\CALX_ntom\romannumeral0\expandafter\CALX_nfromraw_a
    \expanded{#2\xintReplicate{#1}0}\xint:\xint:}%
\def\CALX_iitwo #1#2#3#4%
   {\expandafter\CALX_iitwo_a\expandafter#1\expandafter#2%
    \romannumeral0\CALX_tom{#4}{#3}}%
\def\CALX_iitwo_a #1#2#3#4\xint:#5%
   {\expandafter\CALX_iitwo_b\expandafter#1\expandafter#2\expandafter#3%
    \romannumeral0\CALX_tom{#5}#3#4\xint:}%
\def\CALX_iitwo_b #1#2#3#4%
   {\if\if M#4#3\else ?\fi M\expandafter#1\else\expandafter#2\fi #4}%
\def\CALX_iisp_g #1#2#3\xint:#4\xint:%
   {#1 0.{#2{\CALX_mton#3\xint:}{\CALX_mton#4\xint:}}}%
\def\CALX_mton #1#2\xint:%
   {\if M#1\xint_dothis{\CALX_mton_m#2\xint:}\fi\xint_orthat{#1#2}}%
\def\CALX_mton_m #1#2.#3.#4\xint:%
   {\expanded{N#1\the\numexpr#2+#3.\xintReplicate{#3}{100000000!}#4}}%
\def\CALX_iiun #1#2#3{\if I#3\expandafter#2\else\expandafter#1\fi #3}%
\def\CALX_iimul {\CALX_iitwo\CALX_iimul_x\CALX_iimul_sp}%
\def\CALX_iimul_sp {\CALX_iisp_g\CALX_iifq\CALX_nmul}%
\def\CALX_iimul_x M#1#2.#3.#4\xint:M#5#6.#7.#8\xint:%
   {\expandafter\CALX_ntomd\the\numexpr#3+#7\expandafter.%
    \romannumeral0\CALX_nmul{N#1#2.#4}{N#5#6.#8}}%
\def\CALX_iisqr{\expandafter\CALX_iiun\expandafter\CALX_iisqr_x
   \expandafter\CALX_iiabs_i\romannumeral0\CALX_tom}%
\def\CALX_iisqr_x M#1#2.#3.#4\xint:%
```

```
    {\expandafter\CALX_ntomd\the\numexpr\xint_c_ii*#3\expandafter.%
     \romannumeral0\CALX_nmul{N+#2.#4}{N+#2.#4}}%
\def\CALX_iiopp{\expandafter\CALX_iiun\expandafter\CALX_iiopp_x
   \expandafter\CALX_iiopp_i\romannumeral0\CALX_tom}%
\def\CALX_iiopp_i #1#2\xint:%
   {\if ?#2\xint_dothis{ #1#2}\fi\if -#2\xint_dothis{ #1+}\fi
    \xint_orthat{ #1-}}%
\def\CALX_iiopp_x M#1#2#3\xint:%
   {\if0#2\xint_dothis{ M#1#2#3}\fi
    \if-#1\xint_dothis{ M+#2#3}\fi
    \xint_orthat{ M-#2#3}}%
\def\CALX_iiabs{\expandafter\CALX_iiun\expandafter\CALX_iiabs_x
   \expandafter\CALX_iiabs_i\romannumeral0\CALX_tom}%
\def\CALX_iiabs_i #1#2\xint:%
   {\if ?#2\xint_dothis{ #1#2}\fi\xint_orthat{ #1+}}%
\def\CALX_iiabs_x M#1#2\xint:{ M+#2}%
\def\CALX_iisqrt {\CALX_iisq\CALX_iisqrt_f}%
\def\CALX_iisqrtr{\CALX_iisq\CALX_iisqrtr_f}%
\def\CALX_iisq #1#2{\expandafter\CALX_iiun\expandafter\CALX_iisqrt_x
   \expandafter\CALX_iisqrt_i\romannumeral0\CALX_tom{#2}#1}%
\def\CALX_iisqrt_i #1\xint:#2%
   {\expandafter#2\romannumeral0\CALX_nsqrtrem{#1}}%
\def\CALX_iisqrt_x M#1#2.#3.#4\xint:#5%
{%
    \ifnum#3=\xint_c_
      \xint_dothis{\expandafter#5\romannumeral0\CALX_nsqrtrem{N#1#2.#4}}\fi
    \xint_orthat{\expandafter#5\romannumeral0\CALX_nsqrtrem
       {\expanded{N#1\the\numexpr#2+#3.\xintReplicate{#3}{100000000!}#4}}}%
}%
\def\CALX_iisqrt_f #1#2{\CALX_ntoms0.#1}%
\def\CALX_iisqrtr_f #1{\CALX_iisqrtr_a #1\xint:}%
\def\CALX_iisqrtr_a #1{%
    \if I#1\xint_dothis\CALX_iisqrtr_i\fi
    \xint_orthat\CALX_iisqrtr_n #1%
}%
\def\CALX_iisqrtr_i #1#2\xint:#3{ #1#2}%
\def\CALX_iisqrtr_n #1#2\xint:#3{%
    \expandafter\CALX_iisqrtr_g
      \romannumeral0\CALX_nsub{#3}{#1#2}\xint:{#1#2}%
}%
\def\CALX_iisqrtr_g N#1#2.#3\xint:#4%
{%
    \if-#1\xint_dothis{\CALX_ntomd0.#4}\fi
    \ifnum#2=\xint_c_ \xint_dothis{\CALX_ntomd0.#4}\fi
    \xint_orthat{\expandafter\CALX_ntom
      \romannumeral0\CALX_nadd{#4}{N+1.100000001!}}%
}%
\def\CALX_iifa #1.#2%
   {\expandafter\CALX_ntomd\the\numexpr#1\expandafter.\romannumeral0#2}%
\def\CALX_iifq #1.#2%
   {\expandafter\CALX_ntoms\expandafter0\expandafter.\romannumeral0#2}%
\def\CALX_iifr #1.#2%
```

```
    {\expandafter\CALX_ntoms\the\numexpr#1\expandafter.\romannumeral0#2}%
\def\CALX_iifp #1.#2{\expandafter\CALX_iifp_a\romannumeral0#2{#1}}%
\def\CALX_iifp_a #1#2#3%
    {\expanded{ {\romannumeral0\CALX_ntoms0.#1}%
                {\romannumeral0\CALX_ntoms#3.#2}}}%
\def\CALX_iiadd{\CALX_iitwo\CALX_iiadd_x\CALX_iiadd_sp}%
\def\CALX_iiadd_sp{\CALX_iisp_g\CALX_iifq\CALX_nadd}%
\def\CALX_iiadd_x M#1.#2.#3\xint:M#4.#5.%
{%
    \expandafter\CALX_iiadd_x_a\the\numexpr#2-#5.#2.M#1.#3\xint:M#4.%
}%
\def\CALX_iiadd_x_a #1%
{%
    \xint_UDzerominusfork
       #1-\CALX_iiadd_x_eq
       0#1\CALX_iiadd_x_lt
       0-\CALX_iiadd_x_gt
    \krof #1%
}%
\def\CALX_iiadd_x_eq 0.#1.M#2\xint:M#3\xint:
{%
    \expandafter\CALX_ntomd\the\numexpr#1\expandafter.%
    \romannumeral0\CALX_nadd_y N#2\xint: N#31;!1;!1;!1;!\W\xint:
}%
\def\CALX_iiadd_x_lt -#1.#2.M#3\xint:M#4#5.#6\xint:
{%
    \expandafter\CALX_ntomd\the\numexpr#2\expandafter.%
    \romannumeral0\expandafter\CALX_nadd_y
        \expanded{N#4\the\numexpr#1+#5.%
        \xintReplicate{#1}{100000000!}}#6\xint:N#31;!1;!1;!1;!\W\xint:
}%
\def\CALX_iiadd_x_gt #1.#2.M#3#4.#5\xint:M#6\xint:
{%
    \expandafter\CALX_ntomd\the\numexpr#2-#1\expandafter.%
    \romannumeral0\expandafter\CALX_nadd_y
       \expanded{N#3\the\numexpr#1+#4.%
       \xintReplicate{#1}{100000000!}}#5\xint:N#61;!1;!1;!1;!\W\xint:
}%
\def\CALX_iisub{\CALX_iitwo\CALX_iisub_x\CALX_iisub_sp}%
\def\CALX_iisub_sp{\CALX_iisp_g\CALX_iifq\CALX_nsub}%
\def\CALX_iisub_x M#1.#2.#3\xint:M#4.#5.%
{%
    \expandafter\CALX_iisub_x_a\the\numexpr#2-#5.#2.M#1.#3\xint:M#4.%
}%
\def\CALX_iisub_x_a #1%
{%
    \xint_UDzerominusfork
       #1-\CALX_iisub_x_eq
       0#1\CALX_iisub_x_lt
       0-\CALX_iisub_x_gt
    \krof #1%
}%
```

```
\def\CALX_iisub_x_eq 0.#1.M#2\xint:M#3\xint:
{%
    \expandafter\CALX_ntomd\the\numexpr#1\expandafter.%
    \romannumeral0\CALX_nsub_y N#3\xint: N#21;!1;!1;!1;!\W\xint:
}%
\def\CALX_iisub_x_lt -#1.#2.M#3\xint:M#4#5.#6\xint:
{%
    \expandafter\CALX_ntomd\the\numexpr#2\expandafter.%
    \romannumeral0\expandafter\CALX_nsub_y
        \expanded{N#4\the\numexpr#1+#5.%
        \xintReplicate{#1}{100000000!}}#6\xint:N#31;!1;!1;!1;!\W\xint:
}%
\def\CALX_iisub_x_gt #1.#2.M#3#4.%
{%
    \expandafter\CALX_iisub_x_gt_b\expanded{\the\numexpr#2-#1.%
       N#3\the\numexpr#1+#4.\xintReplicate{#1}{100000000!}}%
}%
\def\CALX_iisub_x_gt_b #1.#2\xint:M#3\xint:
{%
    \expandafter\CALX_ntomd\the\numexpr#1\expandafter.%
    \romannumeral0\CALX_nsub_y N#3\xint:#21;!1;!1;!1;!\W\xint:
}%
\def\CALX_iidivtrunc{\CALX_iitwo\CALX_iidivtrunc_x\CALX_iidivtrunc_sp}%
\def\CALX_iidivtrunc_sp{\CALX_iisp_g\CALX_iifq\CALX_ndivtrunc}%
\def\CALX_iidivtrunc_x M#1.#2.#3\xint:M#4.#5.%
{%
    \expandafter\CALX_iidivtrunc_x_a\the\numexpr#2-#5.#2.M#1.#3\xint:M#4.%
}%
\def\CALX_iidivtrunc_x_a #1%
{%
    \xint_UDzerominusfork
       #1-\CALX_iidivtrunc_x_eq
       0#1\CALX_iidivtrunc_x_lt
       0-\CALX_iidivtrunc_x_gt
    \krof #1%
}%
\def\CALX_iidivtrunc_x_eq 0.#1.M#2\xint:M#3\xint:
{%
    \expandafter\CALX_ntoms\expandafter0\expandafter.%
    \romannumeral0\CALX_ndiv_y N#3\xint:N#2\xint:%
       \CALX_ndt_go%
       \CALX_ndiv_zq\CALX_ndiv_zz
}%
\def\CALX_iidivtrunc_x_lt -#1.#2.M#3\xint:M#4#5.#6\xint:
{%
    \expandafter\CALX_ntoms\expandafter0\expandafter.%
    \romannumeral0\expandafter\CALX_ndiv_y
        \expanded{N#4\the\numexpr#1+#5.%
        \xintReplicate{#1}{100000000!}}#6\xint:N#3\xint:%
       \CALX_ndt_go%
       \CALX_ndiv_zq\CALX_ndiv_zz
}%
```

```
\def\CALX_iidivtrunc_x_gt #1.#2.M#3#4.%
{%
    \expandafter\CALX_iidivtrunc_x_gt_b\expanded{\the\numexpr#2-#1.%
       N#3\the\numexpr#1+#4.\xintReplicate{#1}{100000000!}}%
}%
\def\CALX_iidivtrunc_x_gt_b #1.#2\xint:M#3\xint:
{%
    \expandafter\CALX_ntoms\expandafter0\expandafter.%
    \romannumeral0\CALX_ndiv_y N#3\xint:#2\xint:%
       \CALX_ndt_go%
       \CALX_ndiv_zq\CALX_ndiv_zz
}%
\def\CALX_iimodtrunc{\CALX_iitwo\CALX_iimodtrunc_x\CALX_iimodtrunc_sp}%
\def\CALX_iimodtrunc_sp{\CALX_iisp_g\CALX_iifq\CALX_nmodtrunc}%
\def\CALX_iimodtrunc_x M#1.#2.#3\xint:M#4.#5.%
{%
    \expandafter\CALX_iimodtrunc_x_a\the\numexpr#2-#5.#2.M#1.#3\xint:M#4.%
}%
\def\CALX_iimodtrunc_x_a #1%
{%
    \xint_UDzerominusfork
       #1-\CALX_iimodtrunc_x_eq
       0#1\CALX_iimodtrunc_x_lt
       0-\CALX_iimodtrunc_x_gt
    \krof #1%
}%
\def\CALX_iimodtrunc_x_eq 0.#1.M#2\xint:M#3\xint:
{%
    \expandafter\CALX_ntoms\the\numexpr#1\expandafter.%
    \romannumeral0\CALX_ndiv_y N#3\xint:N#2\xint:%
       \CALX_nmt_go%
       \CALX_ndiv_zr\CALX_ndiv_zz
}%
\def\CALX_iimodtrunc_x_lt -#1.#2.M#3\xint:M#4#5.#6\xint:
{%
    \expandafter\CALX_ntoms\the\numexpr#2\expandafter.%
    \romannumeral0\expandafter\CALX_ndiv_y
        \expanded{N#4\the\numexpr#1+#5.%
        \xintReplicate{#1}{100000000!}}#6\xint:N#3\xint:%
       \CALX_nmt_go%
       \CALX_ndiv_zr\CALX_ndiv_zz
}%
\def\CALX_iimodtrunc_x_gt #1.#2.M#3#4.%
{%
    \expandafter\CALX_iimodtrunc_x_gt_b\expanded{\the\numexpr#2-#1.%
       N#3\the\numexpr#1+#4.\xintReplicate{#1}{100000000!}}%
}%
\def\CALX_iimodtrunc_x_gt_b #1.#2\xint:M#3\xint:
{%
    \expandafter\CALX_ntoms\the\numexpr#1\expandafter.%
    \romannumeral0\CALX_ndiv_y N#3\xint:#2\xint:%
       \CALX_nmt_go%
```

```
        \CALX_ndiv_zr\CALX_ndiv_zz
}%
\def\CALX_iidivfloor{\CALX_iitwo\CALX_iidivfloor_x\CALX_iidivfloor_sp}%
\def\CALX_iidivfloor_sp{\CALX_iisp_g\CALX_iifq\CALX_ndivfloor}%
\def\CALX_iidivfloor_x M#1.#2.#3\xint:M#4.#5.%
{%
    \expandafter\CALX_iidivfloor_x_a\the\numexpr#2-#5.#2.M#1.#3\xint:M#4.%
}%
\def\CALX_iidivfloor_x_a #1%
{%
    \xint_UDzerominusfork
       #1-\CALX_iidivfloor_x_eq
       0#1\CALX_iidivfloor_x_lt
       0-\CALX_iidivfloor_x_gt
    \krof #1%
}%
\def\CALX_iidivfloor_x_eq 0.#1.M#2\xint:M#3\xint:
{%
    \expandafter\CALX_ntoms\expandafter0\expandafter.%
    \romannumeral0\CALX_ndiv_y N#3\xint:N#2\xint:%
       \CALX_ndf_go%
       \CALX_ndiv_zq\CALX_ndiv_zz
}%
\def\CALX_iidivfloor_x_lt -#1.#2.M#3\xint:M#4#5.#6\xint:
{%
    \expandafter\CALX_ntoms\expandafter0\expandafter.%
    \romannumeral0\expandafter\CALX_ndiv_y
        \expanded{N#4\the\numexpr#1+#5.%
        \xintReplicate{#1}{100000000!}}#6\xint:N#3\xint:%
       \CALX_ndf_go%
       \CALX_ndiv_zq\CALX_ndiv_zz
}%
\def\CALX_iidivfloor_x_gt #1.#2.M#3#4.%
{%
    \expandafter\CALX_iidivfloor_x_gt_b\expanded{\the\numexpr#2-#1.%
       N#3\the\numexpr#1+#4.\xintReplicate{#1}{100000000!}}%
}%
\def\CALX_iidivfloor_x_gt_b #1.#2\xint:M#3\xint:
{%
    \expandafter\CALX_ntoms\expandafter0\expandafter.%
    \romannumeral0\CALX_ndiv_y N#3\xint:#2\xint:%
       \CALX_ndf_go%
       \CALX_ndiv_zq\CALX_ndiv_zz
}%
\def\CALX_iimod{\CALX_iitwo\CALX_iimod_x\CALX_iimod_sp}%
\def\CALX_iimod_sp{\CALX_iisp_g\CALX_iifq\CALX_nmod}%
\def\CALX_iimod_x M#1.#2.#3\xint:M#4.#5.%
{%
    \expandafter\CALX_iimod_x_a\the\numexpr#2-#5.#2.M#1.#3\xint:M#4.%
}%
\def\CALX_iimod_x_a #1%
{%
```

```
    \xint_UDzerominusfork
       #1-\CALX_iimod_x_eq
       0#1\CALX_iimod_x_lt
       0-\CALX_iimod_x_gt
    \krof #1%
}%
\def\CALX_iimod_x_eq 0.#1.M#2\xint:M#3\xint:
{%
    \expandafter\CALX_ntoms\the\numexpr#1\expandafter.%
    \romannumeral0\CALX_ndiv_y N#3\xint:N#2\xint:%
       \CALX_nmd_go%
       \CALX_ndiv_zr\CALX_ndiv_zz
}%
\def\CALX_iimod_x_lt -#1.#2.M#3\xint:M#4#5.#6\xint:
{%
    \expandafter\CALX_ntoms\the\numexpr#2\expandafter.%
    \romannumeral0\expandafter\CALX_ndiv_y
        \expanded{N#4\the\numexpr#1+#5.%
        \xintReplicate{#1}{100000000!}}#6\xint:N#3\xint:%
       \CALX_nmd_go%
       \CALX_ndiv_zr\CALX_ndiv_zz
}%
\def\CALX_iimod_x_gt #1.#2.M#3#4.%
{%
    \expandafter\CALX_iimod_x_gt_b\expanded{\the\numexpr#2-#1.%
       N#3\the\numexpr#1+#4.\xintReplicate{#1}{100000000!}}%
}%
\def\CALX_iimod_x_gt_b #1.#2\xint:M#3\xint:
{%
    \expandafter\CALX_ntoms\the\numexpr#1\expandafter.%
    \romannumeral0\CALX_ndiv_y N#3\xint:#2\xint:%
       \CALX_nmd_go%
       \CALX_ndiv_zr\CALX_ndiv_zz
}%
\def\CALX_iiquo{\CALX_iitwo\CALX_iiquo_x\CALX_iiquo_sp}%
\def\CALX_iiquo_sp{\CALX_iisp_g\CALX_iifq\CALX_nquo}%
\def\CALX_iiquo_x M#1.#2.#3\xint:M#4.#5.%
{%
    \expandafter\CALX_iiquo_x_a\the\numexpr#2-#5.#2.M#1.#3\xint:M#4.%
}%
\def\CALX_iiquo_x_a #1%
{%
    \xint_UDzerominusfork
       #1-\CALX_iiquo_x_eq
       0#1\CALX_iiquo_x_lt
       0-\CALX_iiquo_x_gt
    \krof #1%
}%
\def\CALX_iiquo_x_eq 0.#1.M#2\xint:M#3\xint:
{%
    \expandafter\CALX_ntoms\expandafter0\expandafter.%
    \romannumeral0\CALX_ndiv_y N#3\xint:N#2\xint:%
```

```
        \CALX_nqu_go%
        \CALX_ndiv_zq\CALX_ndiv_zz
}%
\def\CALX_iiquo_x_lt -#1.#2.M#3\xint:M#4#5.#6\xint:
{%
    \expandafter\CALX_ntoms\expandafter0\expandafter.%
    \romannumeral0\expandafter\CALX_ndiv_y
        \expanded{N#4\the\numexpr#1+#5.%
        \xintReplicate{#1}{100000000!}}#6\xint:N#3\xint:%
        \CALX_nqu_go%
        \CALX_ndiv_zq\CALX_ndiv_zz
}%
\def\CALX_iiquo_x_gt #1.#2.M#3#4.%
{%
    \expandafter\CALX_iiquo_x_gt_b\expanded{\the\numexpr#2-#1.%
       N#3\the\numexpr#1+#4.\xintReplicate{#1}{100000000!}}%
}%
\def\CALX_iiquo_x_gt_b #1.#2\xint:M#3\xint:
{%
    \expandafter\CALX_ntoms\expandafter0\expandafter.%
    \romannumeral0\CALX_ndiv_y N#3\xint:#2\xint:%
       \CALX_nqu_go%
       \CALX_ndiv_zq\CALX_ndiv_zz
}%
\def\CALX_iirem{\CALX_iitwo\CALX_iirem_x\CALX_iirem_sp}%
\def\CALX_iirem_sp{\CALX_iisp_g\CALX_iifq\CALX_nrem}%
\def\CALX_iirem_x M#1.#2.#3\xint:M#4.#5.%
{%
    \expandafter\CALX_iirem_x_a\the\numexpr#2-#5.#2.M#1.#3\xint:M#4.%
}%
\def\CALX_iirem_x_a #1%
{%
    \xint_UDzerominusfork
       #1-\CALX_iirem_x_eq
       0#1\CALX_iirem_x_lt
       0-\CALX_iirem_x_gt
    \krof #1%
}%
\def\CALX_iirem_x_eq 0.#1.M#2\xint:M#3\xint:
{%
    \expandafter\CALX_ntoms\the\numexpr#1\expandafter.%
    \romannumeral0\CALX_ndiv_y N#3\xint:N#2\xint:%
       \CALX_nre_go%
       \CALX_ndiv_zr\CALX_ndiv_zz
}%
\def\CALX_iirem_x_lt -#1.#2.M#3\xint:M#4#5.#6\xint:
{%
    \expandafter\CALX_ntoms\the\numexpr#2\expandafter.%
    \romannumeral0\expandafter\CALX_ndiv_y
        \expanded{N#4\the\numexpr#1+#5.%
        \xintReplicate{#1}{100000000!}}#6\xint:N#3\xint:%
       \CALX_nre_go%
```

```
        \CALX_ndiv_zr\CALX_ndiv_zz
}%
\def\CALX_iirem_x_gt #1.#2.M#3#4.%
{%
    \expandafter\CALX_iirem_x_gt_b\expanded{\the\numexpr#2-#1.%
       N#3\the\numexpr#1+#4.\xintReplicate{#1}{100000000!}}%
}%
\def\CALX_iirem_x_gt_b #1.#2\xint:M#3\xint:
{%
    \expandafter\CALX_ntoms\the\numexpr#1\expandafter.%
    \romannumeral0\CALX_ndiv_y N#3\xint:#2\xint:%
       \CALX_nre_go%
       \CALX_ndiv_zr\CALX_ndiv_zz
}%
\def\CALX_iidivround{\CALX_iitwo\CALX_iidivround_x\CALX_iidivround_sp}%
\def\CALX_iidivround_sp{\CALX_iisp_g\CALX_iifq\CALX_ndivround}%
\def\CALX_iidivround_x M#1.#2.#3\xint:M#4.#5.%
{%
    \expandafter\CALX_iidivround_x_a\the\numexpr#2-#5.#2.M#1.#3\xint:M#4.%
}%
\def\CALX_iidivround_x_a #1%
{%
    \xint_UDzerominusfork
       #1-\CALX_iidivround_x_eq
       0#1\CALX_iidivround_x_lt
       0-\CALX_iidivround_x_gt
    \krof #1%
}%
\def\CALX_iidivround_x_eq 0.#1.M#2\xint:M#3\xint:
{%
    \expandafter\CALX_ntoms\expandafter0\expandafter.%
    \romannumeral0\CALX_ndiv_y N#3\xint:N#2\xint:%
       \CALX_ndr_go%
       \CALX_ndiv_zq\CALX_ndiv_zz
}%
\def\CALX_iidivround_x_lt -#1.#2.M#3\xint:M#4#5.#6\xint:
{%
    \expandafter\CALX_ntoms\expandafter0\expandafter.%
    \romannumeral0\expandafter\CALX_ndiv_y
        \expanded{N#4\the\numexpr#1+#5.%
        \xintReplicate{#1}{100000000!}}#6\xint:N#3\xint:%
       \CALX_ndr_go%
       \CALX_ndiv_zq\CALX_ndiv_zz
}%
\def\CALX_iidivround_x_gt #1.#2.M#3#4.%
{%
    \expandafter\CALX_iidivround_x_gt_b\expanded{\the\numexpr#2-#1.%
       N#3\the\numexpr#1+#4.\xintReplicate{#1}{100000000!}}%
}%
\def\CALX_iidivround_x_gt_b #1.#2\xint:M#3\xint:
{%
    \expandafter\CALX_ntoms\expandafter0\expandafter.%
```

```
    \romannumeral0\CALX_ndiv_y N#3\xint:#2\xint:%
       \CALX_ndr_go%
       \CALX_ndiv_zq\CALX_ndiv_zz
}%
\def\CALX_iidivision{\CALX_iitwo\CALX_iidivision_x\CALX_iidivision_sp}%
\def\CALX_iidivision_sp{\CALX_iisp_g\CALX_iifq\CALX_ndivision}%
\def\CALX_iidivision_x M#1.#2.#3\xint:M#4.#5.%
{%
    \expandafter\CALX_iidivision_x_a\the\numexpr#2-#5.#2.M#1.#3\xint:M#4.%
}%
\def\CALX_iidivision_x_a #1%
{%
    \xint_UDzerominusfork
       #1-\CALX_iidivision_x_eq
       0#1\CALX_iidivision_x_lt
       0-\CALX_iidivision_x_gt
    \krof #1%
}%
\def\CALX_iidivision_x_eq 0.#1.M#2\xint:M#3\xint:
{%
    \CALX_iifp#1.{%
    \romannumeral0\CALX_ndiv_y N#3\xint:N#2\xint:%
       \CALX_ndn_go%
       \CALX_ndiv_zp\CALX_ndiv_zzp}
}%
\def\CALX_iidivision_x_lt -#1.#2.M#3\xint:M#4#5.#6\xint:
{%
    \CALX_iifp#2.{%
    \romannumeral0\expandafter\CALX_ndiv_y
        \expanded{N#4\the\numexpr#1+#5.%
        \xintReplicate{#1}{100000000!}}#6\xint:N#3\xint:%
       \CALX_ndn_go%
       \CALX_ndiv_zp\CALX_ndiv_zzp}
}%
\def\CALX_iidivision_x_gt #1.#2.M#3#4.%
{%
    \expandafter\CALX_iidivision_x_gt_b\expanded{\the\numexpr#2-#1.%
       N#3\the\numexpr#1+#4.\xintReplicate{#1}{100000000!}}%
}%
\def\CALX_iidivision_x_gt_b #1.#2\xint:M#3\xint:
{%
    \CALX_iifp#1.{%
    \romannumeral0\CALX_ndiv_y N#3\xint:#2\xint:%
       \CALX_ndn_go%
       \CALX_ndiv_zp\CALX_ndiv_zzp}
}%
\def\CALX_iidivmod{\CALX_iitwo\CALX_iidivmod_x\CALX_iidivmod_sp}%
\def\CALX_iidivmod_sp{\CALX_iisp_g\CALX_iifq\CALX_ndivmod}%
\def\CALX_iidivmod_x M#1.#2.#3\xint:M#4.#5.%
{%
    \expandafter\CALX_iidivmod_x_a\the\numexpr#2-#5.#2.M#1.#3\xint:M#4.%
}%
```

```
\def\CALX_iidivmod_x_a #1%
{%
    \xint_UDzerominusfork
       #1-\CALX_iidivmod_x_eq
       0#1\CALX_iidivmod_x_lt
       0-\CALX_iidivmod_x_gt
    \krof #1%
}%
\def\CALX_iidivmod_x_eq 0.#1.M#2\xint:M#3\xint:
{%
    \CALX_iifp#1.{%
    \romannumeral0\CALX_ndiv_y N#3\xint:N#2\xint:%
       \CALX_ndm_go%
       \CALX_ndiv_zp\CALX_ndiv_zzp}%
}%
\def\CALX_iidivmod_x_lt -#1.#2.M#3\xint:M#4#5.#6\xint:
{%
    \CALX_iifp#2.{%
    \romannumeral0\expandafter\CALX_ndiv_y
        \expanded{N#4\the\numexpr#1+#5.%
        \xintReplicate{#1}{100000000!}}#6\xint:N#3\xint:%
       \CALX_ndm_go%
       \CALX_ndiv_zp\CALX_ndiv_zzp}%
}%
\def\CALX_iidivmod_x_gt #1.#2.M#3#4.%
{%
    \expandafter\CALX_iidivmod_x_gt_b\expanded{\the\numexpr#2-#1.%
       N#3\the\numexpr#1+#4.\xintReplicate{#1}{100000000!}}%
}%
\def\CALX_iidivmod_x_gt_b #1.#2\xint:M#3\xint:
{%
    \CALX_iifp#1.{%
    \romannumeral0\CALX_ndiv_y N#3\xint:#2\xint:%
       \CALX_ndm_go%
       \CALX_ndiv_zp\CALX_ndiv_zzp}%
}%
\def\CALX_iigcd{\CALX_iitwo\CALX_iigcd_x\CALX_iigcd_sp}%
\def\CALX_iigcd_sp{\CALX_iisp_g\CALX_iifq\CALX_ngcd}%
\def\CALX_iigcd_x M#1.#2.#3\xint:M#4.#5.%
{%
    \expandafter\CALX_iigcd_x_a\the\numexpr#2-#5.#2.M#1.#3\xint:M#4.%
}%
\def\CALX_iigcd_x_a #1%
{%
    \xint_UDzerominusfork
       #1-\CALX_iigcd_x_eq
       0#1\CALX_iigcd_x_lt
       0-\CALX_iigcd_x_gt
    \krof #1%
}%
\def\CALX_iigcd_x_eq 0.#1.M#2\xint:M#3\xint:
{%
```

```
    \expandafter\CALX_ntomd\the\numexpr#1\expandafter.%
    \romannumeral0\CALX_ngcd{N#2}{N#3}%
}%
\def\CALX_iigcd_x_lt -#1.#2.M#3\xint:M#4#5.#6\xint:
{%
    \expandafter\CALX_ntomd\the\numexpr#2\expandafter.%
    \romannumeral0\CALX_ngcd{N#3}%
       {\expanded{N#4\the\numexpr#1+#5.%
        \xintReplicate{#1}{100000000!}}#6}%
}%
\def\CALX_iigcd_x_gt #1.#2.M#3#4.#5\xint:M#6\xint:
{%
    \expandafter\CALX_ntomd\the\numexpr#2-#1\expandafter.%
    \romannumeral0\CALX_ngcd{\expanded{N#3\the\numexpr#1+#4.%
       \xintReplicate{#1}{100000000!}}#5}{N#6}%
}%
\def\CALX_iilcm{\CALX_iitwo\CALX_iilcm_x\CALX_iilcm_sp}%
\def\CALX_iilcm_sp{\CALX_iisp_g\CALX_iifq\CALX_nlcm}%
\def\CALX_iilcm_x M#1.#2.#3\xint:M#4.#5.%
{%
    \expandafter\CALX_iilcm_x_a\the\numexpr#2-#5.#2.M#1.#3\xint:M#4.%
}%
\def\CALX_iilcm_x_a #1%
{%
    \xint_UDzerominusfork
       #1-\CALX_iilcm_x_eq
       0#1\CALX_iilcm_x_lt
       0-\CALX_iilcm_x_gt
    \krof #1%
}%
\def\CALX_iilcm_x_eq 0.#1.M#2\xint:M#3\xint:
{%
    \expandafter\CALX_ntomd\the\numexpr#1\expandafter.%
    \romannumeral0\CALX_nlcm{N#2}{N#3}%
}%
\def\CALX_iilcm_x_lt -#1.#2.M#3\xint:M#4#5.#6\xint:
{%
    \expandafter\CALX_ntomd\the\numexpr#2\expandafter.%
    \romannumeral0\CALX_nlcm{N#3}%
       {\expanded{N#4\the\numexpr#1+#5.%
        \xintReplicate{#1}{100000000!}}#6}%
}%
\def\CALX_iilcm_x_gt #1.#2.M#3#4.#5\xint:M#6\xint:
{%
    \expandafter\CALX_ntomd\the\numexpr#2-#1\expandafter.%
    \romannumeral0\CALX_nlcm{\expanded{N#3\the\numexpr#1+#4.%
       \xintReplicate{#1}{100000000!}}#5}{N#6}%
}%
\def\CALX_mr #1%
   {\romannumeral0\expandafter\CALX_mr_a\romannumeral`&&@#1\xint:}%
\def\CALX_mr_a #1%
{%
```

```
    \if I#1\xint_dothis\CALX_mr_x\fi
    \if M#1\xint_dothis\CALX_mr_m\fi
    \xint_orthat\CALX_mr_r #1%
}%
\def\CALX_mr_x #1\xint:{ #1}%
\def\CALX_mr_m #1{\CALX_mtoraw_a M}%
\def\CALX_mr_r #1\xint:{\CALX_mr_s #1[.\xint:}%
\def\CALX_mr_s #1[#2{\xint_gob_til_dot#2\CALX_mr_b.\CALX_mr_t #2#1[}%
\def\CALX_mr_b.\CALX_mr_t.#1[\xint:{ #1}%
\def\CALX_mr_t #1%
{%
    \xint_UDsignfork
        #1\CALX_mr_neg
         -\CALX_mr_z
    \krof #1%
}%
\def\CALX_mr_z #1#2[#3][.\xint:{\expanded{ #2\xintReplicate{#1#3}0}}%
\def\CALX_mr_neg -#1[#2][.\xint:{ #1[-#2]}%
\def\CALX_iipow_a #1#2.%
   {\xint_gob_til_minus #1\CALX_iipow_n-\CALX_iipow_p #1#2.}%
\def\CALX_iipow_n -\CALX_iipow_p{\CALX_iipow_t\CALX_iipow_r}%
\def\CALX_iipow_p {\CALX_iipow_t\CALX_iipow_q}%
\def\CALX_iipow_t #1#2.#3{\if I#3\CALX_iipow_j\fi #1#2.#3}%
\def\CALX_iipow_j\fi #1#2.{\fi\CALX_iipow_i #2.}%
\def\CALX_iipow_i #1.#2#3\xint:%
{%
    \ifnum#1=\xint_c_ \xint_dothis{ M+1.0.100000001!}\fi
    \if ?#3\xint_dothis{ #2#3}\fi
    \ifnum#1<\xint_c_ \xint_dothis{ M+0.0.}\fi
    \if +#3\xint_dothis{ #2+}\fi
    \ifodd#1 \xint_dothis{ #2-}\fi
    \xint_orthat{ #2+}%
}%
\def\CALX_iipow_r #1.M#2#3\xint:%
   {\csname CALX_ipr\CALX_iipow_cl M#2{#3}\endcsname #1.#2}%
\def\CALX_ipro #1.#2{ M+1.0.100000001!}%
\def\CALX_iprm #1.#2{\ifodd#1 \xint_afterfi{ M-}\else\xint_afterfi{ M+}\fi
    1.0.100000001!}%
\def\CALX_iprz #1.#2{\CALX_nsig{DivisionByZero}{0 ^ negative}{+}}%
\def\CALX_iprb #1.#2{\CALX_nsig{Underflow}{x ^ -k, abs(x) > 1}{M+0.0.}}%
\def\CALX_iipow_q #1.M#2#3.#4.#5\xint:%
   {\expandafter\CALX_ntomd\the\numexpr#1*#4\expandafter.%
    \romannumeral0\CALX_npow{#1}{N#2#3.#5}}%
\def\CALX_iicmp_x M#1#2.#3.#4\xint:M#5#6.#7.#8\xint:%
{%
    \expandafter\CALX_iicmp_y\the\numexpr#1#2+#1#3-(#5#6+#5#7)%
    .#2#6.M#1#2.#3.#4\xint:M#5#6.#7.#8\xint:%
}%
\def\CALX_iicmp_y #1#2.%
{%
    \if0#1\xint_dothis\CALX_iicmp_e\fi
    \if-#1\xint_dothis\CALX_iicmp_l\fi
```

```
    \xint_orthat\CALX_iicmp_g
}%
\def\CALX_iicmp_l #1.#2\xint:#3\xint:{\xint_stop_atthirdofthree}%
\def\CALX_iicmp_g #1.#2\xint:#3\xint:{\xint_stop_atsecondofthree}%
\def\CALX_iicmp_e #1.%
   {\ifnum#1=11 \expandafter\CALX_iicmp_o\else\expandafter\CALX_iicmp_s\fi}%
\def\CALX_iicmp_o M#1#2.#3.1#4!\xint:M#5#6.#7.1#8!\xint:%
{%
    \ifnum#1#4>#1#8 \xint_dothis\xint_stop_atsecondofthree\fi
    \ifnum#1#4<#1#8 \xint_dothis\xint_stop_atthirdofthree\fi
    \xint_orthat\xint_stop_atfirstofthree
}%
\def\CALX_iicmp_s #1\xint:#2\xint:%
   {\expandafter\CALX_iicmp_t
    \romannumeral0\CALX_iisub_x #1\xint:#2\xint:\xint:}%
\def\CALX_iicmp_t M#1#2#3\xint:%
{%
    \if0#2\xint_dothis\xint_stop_atfirstofthree\fi
    \if-#1\xint_dothis\xint_stop_atthirdofthree\fi
    \xint_orthat\xint_stop_atsecondofthree
}%
\def\CALX_iieq_x #1\xint:#2\xint:%
   {\if0\xintstrcmp{#1}{#2}\expandafter\xint_stop_atfirstoftwo
    \else\expandafter\xint_stop_atsecondoftwo\fi}%
\def\CALX_iipow_e #1#2\xint:%
   {\expandafter\CALX_iipow_ee\expandafter#1\expandafter#2%
    \romannumeral0\CALX_tom}%
\def\CALX_iipow_ee #1#2#3#4#5\xint:%
   {\csname CALX_ipe#2\CALX_iipow_cl#3#4{#5}\endcsname#1#2#4}%
\def\CALX_iipow_cl #1#2#3%
   {\if I#1\if ?#2n\else i\fi\else
    \ifnum\xintstrcmp{#3}{0.0.}=0 z\else
    \ifnum\xintstrcmp{#3}{1.0.100000001!}=0 \if -#2m\else o\fi
    \else b\fi\fi\fi}%
\def\CALX_ipe_one #1#2#3{ M+1.0.100000001!}%
\def\CALX_ipe_zero #1#2#3{ M+0.0.}%
\def\CALX_ipe_inf #1#2#3{ #1+}%
\def\CALX_ipe_es #1#2#3{ #1#2}%
\def\CALX_ipe_bs #1#2#3{ #1#3}%
\def\CALX_ipe_dz #1#2#3{\CALX_nsig{DivisionByZero}{0 ^ -inf}{+}}%
\def\CALX_tmpa #1#2{\expandafter\let\csname CALX_ipe#1\endcsname#2}%
\CALX_tmpa{+o}\CALX_ipe_one \CALX_tmpa{+m}\CALX_ipe_one
\CALX_tmpa{+z}\CALX_ipe_zero\CALX_tmpa{+b}\CALX_ipe_inf
\CALX_tmpa{+i}\CALX_ipe_inf \CALX_tmpa{+n}\CALX_ipe_bs
\CALX_tmpa{-o}\CALX_ipe_one \CALX_tmpa{-m}\CALX_ipe_one
\CALX_tmpa{-z}\CALX_ipe_dz  \CALX_tmpa{-b}\CALX_ipe_zero
\CALX_tmpa{-i}\CALX_ipe_zero\CALX_tmpa{-n}\CALX_ipe_bs
\CALX_tmpa{?o}\CALX_ipe_one \CALX_tmpa{?m}\CALX_ipe_es
\CALX_tmpa{?z}\CALX_ipe_es  \CALX_tmpa{?b}\CALX_ipe_es
\CALX_tmpa{?i}\CALX_ipe_es  \CALX_tmpa{?n}\CALX_ipe_es
\def\CALX_iiSum{\romannumeral0\CALX_iisum}%
\def\CALX_iisum #1%
```

```
{%
    \xint_gob_til_^ #1\CALX_iisum_empty ^%
    \expandafter\CALX_iisum_loop\romannumeral`&&@#1\xint:
}%
\def\CALX_iisum_empty ^#1\xint:{ M+0.0.}%
\def\CALX_iisum_loop #1\xint:#2%
{%
    \xint_gob_til_^ #2\CALX_iisum_end ^%
    \expandafter\CALX_iisum_loop
    \romannumeral0\CALX_iiadd{#1}{#2}\xint:
}%
\def\CALX_iisum_end ^#1\CALX_iiadd #2#3\xint:{ #2}%
\def\CALX_iiPrd{\romannumeral0\CALX_iiprd}%
\def\CALX_iiprd #1%
{%
    \xint_gob_til_^ #1\CALX_iiprd_empty ^%
    \expandafter\CALX_iiprd_loop\romannumeral`&&@#1\xint:
}%
\def\CALX_iiprd_empty ^#1\xint:{ M+1.0.100000001!}%
\def\CALX_iiprd_loop #1\xint:#2%
{%
    \xint_gob_til_^ #2\CALX_iiprd_end ^%
    \expandafter\CALX_iiprd_loop
    \romannumeral0\CALX_iimul{#1}{#2}\xint:
}%
\def\CALX_iiprd_end ^#1\CALX_iimul #2#3\xint:{ #2}%
\def\CALX_iiGCDof   {\romannumeral0\CALX_iigcdof_a}%
\def\CALX_iigcdof_a #1{\expandafter\CALX_iigcdof_b\romannumeral`&&@#1\xint:}%
\def\CALX_iigcdof_b #1\xint:#2%
   {\expandafter\CALX_iigcdof_c\romannumeral`&&@#2\xint:{#1}\xint:}%
\def\CALX_iigcdof_c #1{\xint_gob_til_^ #1\CALX_iigcdof_e ^\CALX_iigcdof_d #1}%
\def\CALX_iigcdof_d #1\xint:%
   {\expandafter\CALX_iigcdof_b\romannumeral0\CALX_iigcd {#1}}%
\def\CALX_iigcdof_e #1\xint:#2\xint:{ #2}%
\expandafter\def\csname CALX_iiLCMof\endcsname{\romannumeral0\CALX_iilcmof_a}%
\def\CALX_iilcmof_a #1{\expandafter\CALX_iilcmof_b\romannumeral`&&@#1\xint:}%
\def\CALX_iilcmof_b #1\xint:#2%
   {\expandafter\CALX_iilcmof_c\romannumeral`&&@#2\xint:{#1}\xint:}%
\def\CALX_iilcmof_c #1{\xint_gob_til_^ #1\CALX_iilcmof_e ^\CALX_iilcmof_d #1}%
\def\CALX_iilcmof_d #1\xint:%
   {\expandafter\CALX_iilcmof_b\romannumeral0\CALX_iilcm {#1}}%
\def\CALX_iilcmof_e #1\xint:#2\xint:{ #2}%
\def\CALX_ngcd {\CALX_ngl\CALX_ngcd_l}%
\def\CALX_nlcm {\CALX_ngl\CALX_nlcm_a}%
\def\CALX_ngl #1#2%
   {\expandafter\CALX_ngl_a\expandafter#1\romannumeral`&&@#2\xint:}%
\def\CALX_ngl_a #1#2\xint:#3%
   {\expandafter\CALX_ngl_b\expandafter#1\romannumeral`&&@#3\xint:#2\xint:}%
\def\CALX_ngl_b #1#2#3\xint:#4%
{%
    \if I#2\xint_dothis\CALX_ngl_bi\fi
    \if I#4\xint_dothis\CALX_ngl_bi\fi
```

```
    \xint_orthat{\CALX_ngl_bn #1{#2#3}#4}%
}%
\def\CALX_ngl_bi #1\xint:{\CALX_ngl_i}%
\def\CALX_ngl_bn #1#2#3#4\xint:{#1{#3#4}{#2}}%
\def\CALX_ngl_i{\CALX_nsig{InvalidOperation}{gcd or lcm of I}{}}%
\def\CALX_ngcd_l #1#2{\CALX_ngcd_t #2\xint:{#1}}%
\def\CALX_ngcd_t N#1#2.#3\xint:#4%
{%
    \ifnum#2=\xint_c_ \xint_dothis{\CALX_ngcd_e #4}\fi
    \xint_orthat{\expandafter\CALX_ngcd_t\romannumeral0%
      \CALX_nrem{#4}{N#1#2.#3}\xint:{N#1#2.#3}}%
}%
\def\CALX_ngcd_e N#1{ N+}%
\def\CALX_nlcm_a #1#2%
   {\expandafter\CALX_nlcm_b\romannumeral0\CALX_ngcd_l{#1}{#2}\xint:{#1}{#2}}%
\def\CALX_nlcm_b N#1#2.#3\xint:#4#5%
{%
    \ifnum#2=\xint_c_ \xint_dothis{ N+0.}\fi
    \xint_orthat{\expandafter\CALX_ngcd_e\romannumeral0%
      \CALX_nmul{\CALX_nquo{#4}{N#1#2.#3}}{#5}}%
}%
\def\CALX_iifac{\expandafter\CALX_iiun\expandafter\CALX_iifac_x
   \expandafter\CALX_mrinv\romannumeral0\CALX_tom}%
\def\CALX_iifac_x M#1#2.#3.#4\xint:%
{%
    \ifnum#2=\xint_c_ \xint_dothis{ M+1.0.100000001!}\fi
    \if-#1\xint_dothis\CALX_iifac_n\fi
    \ifnum#2#3>10 \xint_dothis\CALX_iifac_n\fi
    \xint_orthat{\CALX_iifac_y #4}%
}%
\def\CALX_iifac_y 1#1!%
{%
    \ifnum#1>\xint_c_e_iv \xint_dothis\CALX_iifac_n\fi
    \xint_orthat{\expandafter\CALX_ntom\romannumeral0\CALX_nfac{#1}}%
}%
\def\CALX_iifac_n
   {\CALX_nsig{InvalidOperation}{fac of a negative or > 10^4 integer}{}}%
\def\CALX_iipar{\expandafter\CALX_iiun\expandafter\CALX_iipar_x
   \expandafter\CALX_iipar_i\romannumeral0\CALX_tom}%
\def\CALX_iipar_i #1#2\xint:#3#4%
   {\CALX_nsig{InvalidOperation}{integer function of I}{}}%
\def\CALX_iipar_x M#1#2.#3.#4\xint:%
{%
    \ifnum\ifnum#3=\xint_c_ #2\else\xint_c_\fi=\xint_c_
      \xint_dothis\xint_stop_atsecondoftwo\fi
    \xint_orthat{\CALX_iipar_y #4\xint:}%
}%
\def\CALX_iipar_y 1#1!#2\xint:%
   {\ifodd#1 \expandafter\xint_stop_atfirstoftwo
    \else\expandafter\xint_stop_atsecondoftwo\fi}%
\def\CALX_iiisone{\expandafter\CALX_iiisone_a\romannumeral0\CALX_tom}%
\def\CALX_iiisone_a #1#2\xint:#3#4#5%
```

```
{%
    \if I#1\xint_dothis{#5}\fi
    \ifnum\xintstrcmp{#2}{+1.0.100000001!}=\xint_c_ \xint_dothis{#3}\fi
    \xint_orthat{#4}%
}%
\def\CALX_iione_i #1\xint:{\xint_stop_atsecondoftwo}%
\def\CALX_iilogten{\expandafter\CALX_iiun\expandafter\CALX_iilt_x
   \expandafter\CALX_mrinv\romannumeral0\CALX_tom}%
\def\CALX_iilt_x M#1#2.#3.#4\xint:%
{%
    \ifnum#2=\xint_c_ \xint_dothis{ -2147450880}\fi
    \xint_orthat{\expandafter\space\the\numexpr\xint_c_viii*(#2+#3)-\xint_c_ix
      +\CALX_nskip\CALX_iilt_t #2-\xint_c_i.#4\xint:\relax}%
}%
\def\CALX_iilt_t 1#1!\xint:{\CALX_ndig{#1}}%
\def\CALX_iibinom #1#2%
   {\expandafter\CALX_iibinom_p\the\numexpr #1\expandafter.\the\numexpr #2.}%
\def\CALX_iibinom_p #1.#2.%
   {\expandafter\CALX_iibinom_f\the\numexpr#1-#2.#2.#1.}%
\def\CALX_iibinom_f #1#2.#3#4.#5#6.%
{%
    \if-#5\xint_dothis\CALX_iibinom_inv\fi
    \if-#1\xint_dothis{ M+0.0.}\fi
    \if-#3\xint_dothis{ M+0.0.}\fi
    \if0#1\xint_dothis{ M+1.0.100000001!}\fi
    \if0#3\xint_dothis{ M+1.0.100000001!}\fi
    \ifnum #5#6>\xint_c_e_viii_mone\xint_dothis\CALX_iibinom_inv\fi
    \ifnum #1#2>#3#4
      \xint_dothis{\expandafter\CALX_ntom\romannumeral0\CALX_binom_a #1#2.#3#4.}\fi
    \xint_orthat{\expandafter\CALX_ntom\romannumeral0\CALX_binom_a #3#4.#1#2.}%
}%
\def\CALX_iibinom_inv
   {\CALX_nsig{InvalidOperation}{binomial: negative or >= 10^8 argument}{}}%
\def\CALX_iipfac #1#2%
   {\expandafter\CALX_iipfac_f\the\numexpr#1\expandafter.\the\numexpr#2.}%
\def\CALX_iipfac_f #1#2.#3#4.%
{%
    \unless\ifnum #1#2<#3#4 \xint_dothis\CALX_iipfac_one\fi
    \if-#3\xint_dothis\CALX_iipfac_neg\fi
    \if-#1\xint_dothis\CALX_iipfac_zero\fi
    \ifnum #3#4>\xint_c_e_viii_mone\xint_dothis\CALX_iipfac_inv\fi
    \xint_orthat\CALX_iipfac_a #1#2.#3#4.%
}%
\def\CALX_iipfac_inv #1.#2.%
   {\CALX_nsig{InvalidOperation}{pfactorial: argument >= 10^8}{}}%
\def\CALX_iipfac_one  #1.#2.{ M+1.0.100000001!}%
\def\CALX_iipfac_zero #1.#2.{ M+0.0.}%
\def\CALX_iipfac_neg -#1.-#2.%
{%
    \ifnum #1>\xint_c_e_viii\xint_dothis\CALX_iipfac_inv\fi
    \xint_orthat
   {\ifodd\numexpr#2-#1\relax
```

```
      \xint_afterfi{\expandafter\CALX_iipfac_m\romannumeral0}\fi
    \expandafter\CALX_iipfac_a}%
    \the\numexpr #2-\xint_c_i\expandafter.\the\numexpr#1-\xint_c_i.%
}%
\def\CALX_iipfac_m M+{ M-}%
\def\CALX_iipfac_a #1.#2.%
   {\expandafter\CALX_ntom\romannumeral0\CALX_npfac{#1}{#2}}%
\def\CALX_iiq #1#2#3#4#5%
{%
    \expandafter\CALX_iiq_a\romannumeral0\CALX_ndivision
      {\CALX_nshl{#5}{\CALX_nfromraw{#3}}}%
      {\CALX_nshl{-#5}{\CALX_nfromraw{#4}}}#1#2%
}%
\def\CALX_iiq_a #1#2{\CALX_iiq_b #2\xint:{#1}}%
\def\CALX_iiq_b N#1#2.#3\xint:#4#5#6%
   {\ifnum#2=\xint_c_ \expandafter#5\else\expandafter#6\fi{#4}}%
\def\CALX_iilit_n #1#2{\CALX_iiq\CALX_iilit_e\CALX_iilit_f{#1}1{#2}{#1[#2]}}%
\def\CALX_iilit_e #1#2{\CALX_ntom #1}%
\def\CALX_iilit_f #1#2{ #2}%
\def\CALX_tom_x #1.#2/\xint:{\CALX_tom_z #2/1/\xint:{#1}{#2[#1]}}%
\def\CALX_tom_z #1/#2/#3\xint:{\CALX_iiq\CALX_tom_e\CALX_tom_f{#1}{#2}}%
\def\CALX_tom_e #1#2{\CALX_ntom #1\xint:}%
\def\CALX_tom_f #1#2{\CALX_iinotint{#2}\CALX_tom_t #1\xint:}%
\def\CALX_tom_t N#1%
   {\if-#1\expandafter\CALX_tom_u\else\expandafter\CALX_ntom\fi N#1}%
\def\CALX_tom_u #1\xint:%
   {\expandafter\CALX_ntom\romannumeral0\CALX_nadd{#1}{N+1.100000001!}\xint:}%
```

`\CALX_seq_m k.` (f-expandable, k explicit digits, |k| < 2^31): the M value of k; one block if |k| < 10^8, else `\CALX_mfromraw`. Items of a..b in `\xinteval` and `\xintiieval`.

```
\def\CALX_seq_m #1.%
{%
    \ifnum#1=\xint_c_ \xint_dothis{M+0.0.}\fi
    \ifnum\ifnum#1<\xint_c_ -\fi#1<\xint_c_e_viii
      \xint_dothis{M\ifnum#1<\xint_c_ -\else+\fi1.0.\the\numexpr
        \xint_c_e_viii\ifnum#1<\xint_c_ -\else+\fi#1!}\fi
    \xint_orthat{\romannumeral0\CALX_mfromraw{#1}}%
}%
```

Syntax (use with `\romannumeral0`; k explicit digits, k >= 0; b a block 1ddddddddd with d > 0; R a raw integer, f-expanded, no leading zeros):
`\CALX_npowmul k.{b}{R}` -> <space><raw integer R d^k>
`\CALX_npow`, `\CALX_nmul` and `\CALX_nfromraw`, `\CALX_ntoraw` of calxcore. Fractional hexadecimal, octal and binary literals (d = 5^j).

```
\def\CALX_npowmul #1.#2#3%
{%
    \expandafter\CALX_ntoraw_a\romannumeral0\expandafter\CALX_nmul_x
    \romannumeral0\CALX_npow{#1}{N+1.#2!}\xint:{\CALX_nfromraw{#3}}\xint:
}%
```

Back to normal letter catcodes for M, N, and I.

```
\catcode`M=11 \catcode`N=11 \catcode`I=11
\def\CALX_iilit #1#2%
{%
```

```
    \ifnum#2<\xint_c_ \xint_dothis{\CALX_iilit_n{#1}{#2}}\fi
    \xint_orthat{\CALX_mfromraw{\expanded{#1\xintReplicate{#2}0}}}%
}%
\def\CALX_iinotint #1%
   {\XINT_expandableerror{\xintiieval: #1 is not an integer, truncated.}}%
\def\calxiiAdd     {\romannumeral0\CALX_iiadd}%
\def\calxiiSub     {\romannumeral0\CALX_iisub}%
\def\calxiiMul     {\romannumeral0\CALX_iimul}%
\def\calxiiSqr     {\romannumeral0\CALX_iisqr}%
\def\calxiiOpp     {\romannumeral0\CALX_iiopp}%
\def\calxiiAbs     {\romannumeral0\CALX_iiabs}%
\def\calxiiSqrt    {\romannumeral0\CALX_iisqrt}%
\def\calxiiDivFloor{\romannumeral0\CALX_iidivfloor}%
\def\calxiiMod     {\romannumeral0\CALX_iimod}%
\def\calxiiDivRound{\romannumeral0\CALX_iidivround}%
\def\calxiiQuo     {\romannumeral0\CALX_iiquo}%
\def\calxiiRem     {\romannumeral0\CALX_iirem}%
\def\calxiiDivMod  {\romannumeral0\CALX_iidivmod}%
\def\calxiiPow #1#2%
   {\romannumeral0\expandafter\CALX_iipow_k\romannumeral`&&@#2\xint:{#1}}%
\def\CALX_iipow_k #1%
   {\if I#1\expandafter\CALX_iipow_e\else\expandafter\CALX_iipow_f\fi #1}%
\def\CALX_iipow_f #1\xint:%
   {\expandafter\CALX_iipow_a\the\numexpr\romannumeral0\CALX_mr_a#1\xint:%
    \expandafter.\romannumeral0\CALX_tom}%
\def\CALX_tmpa #1#2#3#4%
   {\def#1##1##2{\romannumeral0\CALX_iitwo#2#3{##1}{##2}#4}}%
\CALX_tmpa\calxiiLt    \CALX_iicmp_x\CALX_iicmp_sp{001}%
\CALX_tmpa\calxiiGt    \CALX_iicmp_x\CALX_iicmp_sp{010}%
\CALX_tmpa\calxiiLtorEq\CALX_iicmp_x\CALX_iicmp_sp{101}%
\CALX_tmpa\calxiiGtorEq\CALX_iicmp_x\CALX_iicmp_sp{110}%
\CALX_tmpa\calxiiEq    \CALX_iieq_x \CALX_iieq_sp {10}%
\CALX_tmpa\calxiiNotEq \CALX_iieq_x \CALX_iieq_sp {01}%
\def\CALX_iicmp_sp #1#2#3\xint:#4#5#6\xint:#7#8#9%
{%
    \if ?#2\xint_dothis\CALX_fcnan\fi
    \if ?#5\xint_dothis\CALX_fcnan\fi
    \ifnum\csname CALX_icl#1#2\endcsname>\csname CALX_icl#4#5\endcsname
      \xint_dothis{ #8}\fi
    \ifnum\csname CALX_icl#1#2\endcsname<\csname CALX_icl#4#5\endcsname
      \xint_dothis{ #9}\fi
    \xint_orthat{ #7}%
}%
\expandafter\def\csname CALX_iclI+\endcsname{1 }%
\expandafter\def\csname CALX_iclI-\endcsname{-1 }%
\expandafter\def\csname CALX_iclM+\endcsname{0 }%
\expandafter\def\csname CALX_iclM-\endcsname{0 }%
\def\CALX_fcnan
   {\CALX_signalcondition{InvalidOperation}{comparison with I?}{}{ 0}}%
\def\CALX_iieq_sp #1#2#3\xint:#4#5#6\xint:#7#8%
{%
    \if ?#2\xint_dothis{ #8}\fi
```

```
    \if ?#5\xint_dothis{ #8}\fi
    \xint_orthat{\CALX_iieq_x #1#2#3\xint:#4#5#6\xint:#7#8}%
}%
\def\calxiiFac   {\romannumeral0\CALX_iifac}%
\def\calxiiOdd #1{\romannumeral0\CALX_iipar{#1}10}%
\def\calxiiEven #1{\romannumeral0\CALX_iipar{#1}01}%
\def\calxiiIsOne #1{\romannumeral0\CALX_iiisone{#1}{ 1}{ 0}%
   {\CALX_nsig{InvalidOperation}{integer function of I}{}}}%
\def\calxiiifOne #1{\romannumeral0\CALX_iiisone{#1}\xint_stop_atfirstoftwo
   \xint_stop_atsecondoftwo{\expandafter\CALX_iione_i\romannumeral0%
   \CALX_nsig{InvalidOperation}{ifone of I}{}\xint:}}%
\def\calxiiifInt #1%
   {\romannumeral0\expandafter\CALX_iiifint\romannumeral`&&@#1\xint:}%
\def\CALX_iiifint #1#2\xint:%
   {\if I#1\xint_dothis{\expandafter\CALX_iione_i\romannumeral0%
      \CALX_nsig{InvalidOperation}{ifint of I}{}\xint:}\fi
    \xint_orthat\xint_stop_atfirstoftwo}%
\def\calxiiIsInt #1%
   {\romannumeral0\expandafter\CALX_iiisint\romannumeral`&&@#1\xint:}%
\def\CALX_iiisint #1#2\xint:%
   {\if I#1\xint_dothis{\CALX_nsig{InvalidOperation}{integer function of I}{}}\fi
    \xint_orthat{ 1}}%
\def\calxiiSqrtR {\romannumeral0\CALX_iisqrtr}%
\def\calxiiLogTen{\romannumeral0\CALX_iilogten}%
\def\calxiiBinomial {\romannumeral0\CALX_mrd\CALX_iibinom}%
\def\calxiiPFactorial {\romannumeral0\CALX_mrd\CALX_iipfac}%
\def\CALX_mrinv #1#2\xint:%
   {\CALX_nsig{InvalidOperation}{integer function of I}{}}%
\def\CALX_mrd #1#2%
   {\expandafter\CALX_mrd_a\expandafter#1\romannumeral`&&@#2\xint:}%
\def\CALX_mrd_a #1#2\xint:#3%
   {\expandafter\CALX_mrd_b\expandafter#1\romannumeral`&&@#3\xint:#2\xint:}%
\def\CALX_mrd_b #1#2#3\xint:#4#5\xint:%
{%
    \if I#2\xint_dothis\CALX_mrd_i\fi
    \if I#4\xint_dothis\CALX_mrd_i\fi
    \xint_orthat{\expandafter\space\romannumeral`&&@#1%
      {\romannumeral0\CALX_mr_a#4#5\xint:}%
      {\romannumeral0\CALX_mr_a#2#3\xint:}}%
}%
\def\CALX_mrd_i{\CALX_nsig{InvalidOperation}{integer function of I}{}}%
\def\CALX_mrl #1{\xint_gob_til_^#1\CALX_mrl_e ^{\CALX_mr{#1}}\CALX_mrl}%
\def\CALX_mrl_e ^#1\CALX_mrl{}%
\def\CALX_iiMaxof{\romannumeral0\CALX_iimaxof}%
\def\CALX_iimaxof#1%
{%
    \xint_gob_til_^ #1\CALX_iimaxof_empty ^%
    \expandafter\CALX_iimaxof_loop\romannumeral`&&@#1\xint:
}%
\def\CALX_iimaxof_empty ^#1\xint:{ 0}%
\def\CALX_iimaxof_loop #1\xint:#2%
{%
```

```
    \xint_gob_til_^ #2\CALX_iimaxof_e ^%
    \expandafter\CALX_iimaxof_loop\romannumeral0\CALX_iimax{#1}{#2}\xint:
}%
\def\CALX_iimaxof_e ^#1\CALX_iimax #2#3\xint:{ #2}%
\def\CALX_iiMinof{\romannumeral0\CALX_iiminof}%
\def\CALX_iiminof#1%
{%
    \xint_gob_til_^ #1\CALX_iiminof_empty ^%
    \expandafter\CALX_iiminof_loop\romannumeral`&&@#1\xint:
}%
\def\CALX_iiminof_empty ^#1\xint:{ 0}%
\def\CALX_iiminof_loop #1\xint:#2%
{%
    \xint_gob_til_^ #2\CALX_iiminof_e ^%
    \expandafter\CALX_iiminof_loop\romannumeral0\CALX_iimin{#1}{#2}\xint:
}%
\def\CALX_iiminof_e ^#1\CALX_iimin #2#3\xint:{ #2}%
\def\CALX_iimax #1#2{\expandafter\CALX_iimax_a\expandafter1\romannumeral`&&@#2\xint:{#1}}%
\def\CALX_iimin #1#2{\expandafter\CALX_iimax_a\expandafter2\romannumeral`&&@#2\xint:{#1}}%
\def\CALX_iimax_a #1#2\xint:#3%
   {\if1\romannumeral0\csname CALX_iimx_c#1\endcsname{#2}{#3}%
    \xint_afterfi{ #2}\else\xint_afterfi{ #3}\fi}%
\expandafter\def\csname CALX_iimx_c1\endcsname #1#2%
   {\CALX_iitwo\CALX_iicmp_x\CALX_iicmp_sp{#1}{#2}010}%
\expandafter\def\csname CALX_iimx_c2\endcsname #1#2%
   {\CALX_iitwo\CALX_iicmp_x\CALX_iicmp_sp{#1}{#2}001}%
```

30.8. Binomial coefficients

Syntax (use with \romannumeral0; 0 < b <= a < 10^8 as explicit digits, a >= b):
  \CALX_binom_a a.b. -> <space>N+<n>.<blocks> binomial(a + b, b)
The engine of \xintiiBinomial: the accumulator is a block list, most significant first; each step multiplies it (\CALX_smallmul, on the list reversed by \CALX_rev_nounsep) by a product of consecutive integers and divides it (\CALX_smalldivx_a of calxcore) by the matching product of the denominator, four, three, two or one factors per step according to the size of the factors (< 10^8); very small arguments: one \numexpr a*b/c per step. Native output: the final list is reversed (\CALX_rev_nounsep) and given to \CALX_nnorm; very small arguments: one block.

```
\def\CALX_binom_a #1.#2.%
{%
    \expandafter\CALX_binom_b\the\numexpr \xint_c_i+#1.1.#2.100000001!1!;!0!%
}%
```

y=x-k+1.j=1.k. On va évaluer par y/1*(y+1)/2*(y+2)/3 etc... On essaie de regrouper de manière à utiliser au mieux \numexpr. On peut aller jusqu'à x=10000 car 9999*10000<10^8. 463*464*465=99896880, 98*99*100*101=97990200. On va vérifier à chaque étape si on dépasse un seuil. Le style de l'implémentation diffère de celui que j'avais utilisé pour \xintiiFac. On pourrait tout-à-fait avoir une verybigloop, mais bon. Je rajoute aussi un verysmall. Le traitement est un peu différent pour elle afin d'aller jusqu'à x=29 (et pas seulement 26 si je suivais le modèle des autres, mais je veux pouvoir faire binomial(29,1), binomial(29,2), ... en vsmall).

```
\def\CALX_binom_b #1.%
{%
    \ifnum #1>9999 \xint_dothis\CALX_binom_vbigloop \fi
    \ifnum #1>463  \xint_dothis\CALX_binom_bigloop   \fi
```

```
    \ifnum #1>98   \xint_dothis\CALX_binom_medloop   \fi
    \ifnum #1>29   \xint_dothis\CALX_binom_smallloop \fi
                   \xint_orthat\CALX_binom_vsmallloop #1.%
}%
```

y.j.k. Au départ on avait x-k+1.1.k. Ensuite on a des blocs 1<8d>! donnant le résultat intermédiaire, dans l'ordre, et à la fin on a 1!1;!0!. Dans smallloop on peut prendre 4 par 4.

```
\def\CALX_binom_smallloop #1.#2.#3.%
{%
    \ifcase\numexpr #3-#2\relax
        \expandafter\CALX_binom_end_
    \or \expandafter\CALX_binom_end_i
    \or \expandafter\CALX_binom_end_ii
    \or \expandafter\CALX_binom_end_iii
    \else\expandafter\CALX_binom_smallloop_a
    \fi #1.#2.#3.%
}%
```

Ça m'ennuie un peu de reprendre les #1, #2, #3 ici. On a besoin de \numexpr pour \CALX_binom_div, mais de \romannumeral0 pour le unsep après \CALX_binom_mul.

```
\def\CALX_binom_smallloop_a #1.#2.#3.%
{%
    \expandafter\CALX_binom_smallloop_b
    \the\numexpr #1+\xint_c_iv\expandafter.%
    \the\numexpr #2+\xint_c_iv\expandafter.%
    \the\numexpr #3\expandafter.%
    \the\numexpr\expandafter\CALX_binom_div
    \the\numexpr #2*(#2+\xint_c_i)*(#2+\xint_c_ii)*(#2+\xint_c_iii)\expandafter
    !\romannumeral0\expandafter\CALX_binom_mul
    \the\numexpr #1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
}%
\def\CALX_binom_smallloop_b #1.%
{%
    \ifnum #1>98  \expandafter\CALX_binom_medloop   \else
                  \expandafter\CALX_binom_smallloop \fi #1.%
}%
```

Ici on prend trois par trois.

```
\def\CALX_binom_medloop #1.#2.#3.%
{%
    \ifcase\numexpr #3-#2\relax
        \expandafter\CALX_binom_end_
    \or \expandafter\CALX_binom_end_i
    \or \expandafter\CALX_binom_end_ii
    \else\expandafter\CALX_binom_medloop_a
    \fi #1.#2.#3.%
}%
\def\CALX_binom_medloop_a #1.#2.#3.%
{%
    \expandafter\CALX_binom_medloop_b
    \the\numexpr #1+\xint_c_iii\expandafter.%
    \the\numexpr #2+\xint_c_iii\expandafter.%
    \the\numexpr #3\expandafter.%
    \the\numexpr\expandafter\CALX_binom_div
```

```
        \the\numexpr #2*(#2+\xint_c_i)*(#2+\xint_c_ii)\expandafter
    !\romannumeral0\expandafter\CALX_binom_mul
        \the\numexpr #1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
}%
\def\CALX_binom_medloop_b #1.%
{%
    \ifnum #1>463 \expandafter\CALX_binom_bigloop   \else
                  \expandafter\CALX_binom_medloop   \fi #1.%
}%
```

Ici on prend deux par deux.

```
\def\CALX_binom_bigloop #1.#2.#3.%
{%
    \ifcase\numexpr #3-#2\relax
        \expandafter\CALX_binom_end_
    \or \expandafter\CALX_binom_end_i
    \else\expandafter\CALX_binom_bigloop_a
    \fi #1.#2.#3.%
}%
\def\CALX_binom_bigloop_a #1.#2.#3.%
{%
    \expandafter\CALX_binom_bigloop_b
    \the\numexpr #1+\xint_c_ii\expandafter.%
    \the\numexpr #2+\xint_c_ii\expandafter.%
    \the\numexpr #3\expandafter.%
    \the\numexpr\expandafter\CALX_binom_div
        \the\numexpr #2*(#2+\xint_c_i)\expandafter
    !\romannumeral0\expandafter\CALX_binom_mul
        \the\numexpr #1*(#1+\xint_c_i)!%
}%
\def\CALX_binom_bigloop_b #1.%
{%
    \ifnum #1>9999 \expandafter\CALX_binom_vbigloop  \else
                   \expandafter\CALX_binom_bigloop   \fi #1.%
}%
```

Et finalement un par un.

```
\def\CALX_binom_vbigloop #1.#2.#3.%
{%
    \ifnum #3=#2
         \expandafter\CALX_binom_end_
    \else\expandafter\CALX_binom_vbigloop_a
    \fi #1.#2.#3.%
}%
\def\CALX_binom_vbigloop_a #1.#2.#3.%
{%
    \expandafter\CALX_binom_vbigloop
    \the\numexpr #1+\xint_c_i\expandafter.%
    \the\numexpr #2+\xint_c_i\expandafter.%
    \the\numexpr #3\expandafter.%
    \the\numexpr\expandafter\CALX_binom_div\the\numexpr #2\expandafter
    !\romannumeral0\CALX_binom_mul #1!%
}%
```

y.j.k. La partie very small. y est au plus 26 (non 29 mais retesté dans \CALX_binom_vsmallloop_a), et tous les binomial(29,n) sont <10^8. On peut donc faire y(y+1)(y+2)(y+3) et aussi il y a le fait que etex fait a*b/c en double precision. Pour ne pas bifurquer à la fin sur smallloop, si n=27, 27, ou 29 on procède un peu différemment des autres boucles. Si je testais aussi #1 après #3-#2 pour les autres il faudrait des terminaisons différentes.

```
\def\CALX_binom_vsmallloop #1.#2.#3.%
{%
    \ifcase\numexpr #3-#2\relax
        \expandafter\CALX_binom_vsmallend_
    \or \expandafter\CALX_binom_vsmallend_i
    \or \expandafter\CALX_binom_vsmallend_ii
    \or \expandafter\CALX_binom_vsmallend_iii
    \else\expandafter\CALX_binom_vsmallloop_a
    \fi #1.#2.#3.%
}%
\def\CALX_binom_vsmallloop_a #1.%
{%
    \ifnum #1>26  \expandafter\CALX_binom_smallloop_a  \else
                  \expandafter\CALX_binom_vsmallloop_b \fi #1.%
}%
\def\CALX_binom_vsmallloop_b #1.#2.#3.%
{%
    \expandafter\CALX_binom_vsmallloop
    \the\numexpr #1+\xint_c_iv\expandafter.%
    \the\numexpr #2+\xint_c_iv\expandafter.%
    \the\numexpr #3\expandafter.%
    \the\numexpr \expandafter\CALX_binom_vsmallmuldiv
    \the\numexpr  #2*(#2+\xint_c_i)*(#2+\xint_c_ii)*(#2+\xint_c_iii)\expandafter
    !\the\numexpr #1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
}%
\def\CALX_binom_mul #1!#21!;!0!%
{%
    \expandafter\CALX_rev_nounsep\expandafter{\expandafter}%
    \the\numexpr\expandafter\CALX_smallmul
    \the\numexpr\xint_c_e_viii+#1\expandafter
    !\romannumeral0\CALX_rev_nounsep {}1;!#2%
    \R!\R!\R!\R!\R!\R!\R!\W
    \R!\R!\R!\R!\R!\R!\R!\W
    1;!%
}%
\def\CALX_binom_div #1!1;!%
{%
    \expandafter\CALX_smalldivx_a
    \the\numexpr #1/\xint_c_ii\expandafter\xint:
    \the\numexpr \xint_c_e_viii+#1!%
}%
```

Vaguement envisagé d'éviter le 10^8+ mais bon.

```
\def\CALX_binom_vsmallmuldiv #1!#2!1#3!{\xint_c_e_viii+#2*#3/#1!}%
```

On a des terminaisons communes aux trois situations small, med, big, et on est sûr de pouvoir faire les multiplications dans \numexpr, car on vient ici *après* avoir comparé à 9999 ou 463 ou 98.

```
\def\CALX_binom_end_iii #1.#2.#3.%
{%
```

```
    \expandafter\CALX_binom_finish
    \the\numexpr\expandafter\CALX_binom_div
        \the\numexpr #2*(#2+\xint_c_i)*(#2+\xint_c_ii)*(#2+\xint_c_iii)\expandafter
    !\romannumeral0\expandafter\CALX_binom_mul
        \the\numexpr #1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
}%
\def\CALX_binom_end_ii #1.#2.#3.%
{%
    \expandafter\CALX_binom_finish
    \the\numexpr\expandafter\CALX_binom_div
        \the\numexpr #2*(#2+\xint_c_i)*(#2+\xint_c_ii)\expandafter
    !\romannumeral0\expandafter\CALX_binom_mul
        \the\numexpr #1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
}%
\def\CALX_binom_end_i #1.#2.#3.%
{%
    \expandafter\CALX_binom_finish
    \the\numexpr\expandafter\CALX_binom_div
        \the\numexpr #2*(#2+\xint_c_i)\expandafter
    !\romannumeral0\expandafter\CALX_binom_mul
        \the\numexpr #1*(#1+\xint_c_i)!%
}%
\def\CALX_binom_end_ #1.#2.#3.%
{%
    \expandafter\CALX_binom_finish
    \the\numexpr\expandafter\CALX_binom_div\the\numexpr #2\expandafter
    !\romannumeral0\CALX_binom_mul #1!%
}%
\def\CALX_binom_finish #11!;!0!%
{%
    \expandafter\CALX_nnorm\expandafter+%
    \romannumeral0\CALX_rev_nounsep {}#1\R!\R!\R!\R!\R!\R!\R!\W 1;!%
}%
```

Duplication de code seulement pour la boucle avec très petits coeffs, mais en plus on fait au maximum des possibilités. (on pourrait tester plus le résultat déjà obtenu).

```
\def\CALX_binom_vsmallend_iii #1.%
{%
    \ifnum #1>26  \expandafter\CALX_binom_end_iii \else
                  \expandafter\CALX_binom_vsmallend_iiib \fi #1.%
}%
\def\CALX_binom_vsmallend_iiib #1.#2.#3.%
{%
    \expandafter\CALX_binom_vsmallfinish
    \the\numexpr \expandafter\CALX_binom_vsmallmuldiv
    \the\numexpr  #2*(#2+\xint_c_i)*(#2+\xint_c_ii)*(#2+\xint_c_iii)\expandafter
    !\the\numexpr #1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
}%
\def\CALX_binom_vsmallend_ii #1.%
{%
    \ifnum #1>27  \expandafter\CALX_binom_end_ii \else
                  \expandafter\CALX_binom_vsmallend_iib \fi #1.%
}%
```

```
\def\CALX_binom_vsmallend_iib #1.#2.#3.%
{%
    \expandafter\CALX_binom_vsmallfinish
    \the\numexpr \expandafter\CALX_binom_vsmallmuldiv
    \the\numexpr  #2*(#2+\xint_c_i)*(#2+\xint_c_ii)\expandafter
    !\the\numexpr #1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
}%
\def\CALX_binom_vsmallend_i #1.%
{%
    \ifnum #1>28  \expandafter\CALX_binom_end_i \else
                  \expandafter\CALX_binom_vsmallend_ib \fi #1.%
}%
\def\CALX_binom_vsmallend_ib #1.#2.#3.%
{%
    \expandafter\CALX_binom_vsmallfinish
    \the\numexpr \expandafter\CALX_binom_vsmallmuldiv
    \the\numexpr  #2*(#2+\xint_c_i)\expandafter
    !\the\numexpr #1*(#1+\xint_c_i)!%
}%
\def\CALX_binom_vsmallend_ #1.%
{%
    \ifnum #1>29  \expandafter\CALX_binom_end_ \else
                  \expandafter\CALX_binom_vsmallend_b \fi #1.%
}%
\def\CALX_binom_vsmallend_b #1.#2.#3.%
{%
    \expandafter\CALX_binom_vsmallfinish
    \the\numexpr\CALX_binom_vsmallmuldiv #2!#1!%
}%
\catcode`N=12
\def\CALX_binom_vsmallfinish#1{%
\def\CALX_binom_vsmallfinish1##1!1!;!0!{#1N+1.1##1!}%
}\CALX_binom_vsmallfinish{ }%
\catcode`N=11
```

30.9. \xintiiifLt

Moved here at 2.0a1 due to a ridiculous situation with the support macro for randrange(), originating in a serious misalignment of my assistant. This is of course a seriously sub-optimal situation perhaps provisory but will not be modified before 2.0a1 release. 1.09a \xintifLt {n}{m}{YES if n<m}{NO if n>=m}. Restyled in 1.09i. 1.1a adds ii variant

```
\def\xintiiifLt {\romannumeral0\xintiiiflt }%
\def\xintiiiflt #1#2%
{%
    \ifnum\xintiiCmp{#1}{#2}<\xint_c_
          \expandafter\xint_stop_atfirstoftwo
    \else \expandafter\xint_stop_atsecondoftwo
    \fi
}%
```

30.10. (WIP) \xintRandomDigits

1.3b. See user manual. Whether this will be part of xintkernel, xintcore, or xint is yet to be decided.

```
\def\xintRandomDigits{\romannumeral0\xintrandomdigits}%
\def\xintrandomdigits#1%
{%
    \csname xint_gob_andstop_\expandafter\XINT_randomdigits\the\numexpr#1\xint:
}%
\def\XINT_randomdigits#1\xint:
{%
    \expandafter\XINT_randomdigits_a
    \the\numexpr(#1+\xint_c_iii)/\xint_c_viii\xint:#1\xint:
}%
\def\XINT_randomdigits_a#1\xint:#2\xint:
{%
    \romannumeral\numexpr\xint_c_viii*#1-#2\csname XINT_%
      \romannumeral\XINT_replicate #1\endcsname \csname
    XINT_rdg\endcsname
}%
\def\XINT_rdg
{%
    \expandafter\XINT_rdg_aux\the\numexpr%
                \xint_c_nine_e_viii%
                              -\xinttexuniformdeviate\xint_c_ii^vii%
                -\xint_c_ii^vii*\xinttexuniformdeviate\xint_c_ii^vii%
                -\xint_c_ii^xiv*\xinttexuniformdeviate\xint_c_ii^vii%
                -\xint_c_ii^xxi*\xinttexuniformdeviate\xint_c_ii^vii%
                +\xinttexuniformdeviate\xint_c_e_viii%
                \relax%
}%
\def\XINT_rdg_aux#1{XINT_rdg\endcsname}%
\let\XINT_XINT_rdg\endcsname
```

30.11. (WIP) \XINT_eightrandomdigits, \xintEightRandomDigits

1.3b. 1.4 adds some public alias...

```
\def\XINT_eightrandomdigits
{%
     \expandafter\xint_gobble_i\the\numexpr%
                \xint_c_nine_e_viii%
                              -\xinttexuniformdeviate\xint_c_ii^vii%
                -\xint_c_ii^vii*\xinttexuniformdeviate\xint_c_ii^vii%
                -\xint_c_ii^xiv*\xinttexuniformdeviate\xint_c_ii^vii%
                -\xint_c_ii^xxi*\xinttexuniformdeviate\xint_c_ii^vii%
                +\xinttexuniformdeviate\xint_c_e_viii%
                \relax%
}%
\let\xintEightRandomDigits\XINT_eightrandomdigits
\def\xintRandBit{\xinttexuniformdeviate\xint_c_ii}%
```

30.12. (WIP) \xintRandBit

1.4 And let's add also \xintRandBit while we are at it.

```
\def\xintRandBit{\xinttexuniformdeviate\xint_c_ii}%
```

30.13. (WIP) \xintXRandomDigits

1.3b.

```
\def\xintXRandomDigits#1%
{%
    \csname xint_gobble_\expandafter\XINT_xrandomdigits\the\numexpr#1\xint:
}%
\def\XINT_xrandomdigits#1\xint:
{%
    \expandafter\XINT_xrandomdigits_a
    \the\numexpr(#1+\xint_c_iii)/\xint_c_viii\xint:#1\xint:
}%
\def\XINT_xrandomdigits_a#1\xint:#2\xint:
{%
    \romannumeral\numexpr\xint_c_viii*#1-#2\expandafter\endcsname
    \romannumeral`&&@\romannumeral
                     \XINT_replicate #1\endcsname\XINT_eightrandomdigits
}%
```

30.14. (WIP) \xintiiRandRangeAtoB

1.3b. Support for randrange() function.

We do it f-expandably for matters of \xintNewExpr etc... The \xintexpr will add \xintNum wrapper to possible fractional input. But \xintiiexpr will call as is.

TODO: ? implement third argument (STEP) TODO: \xintNum wrapper (which truncates) not so good in floatexpr. Use round?

It is an error if b<=a, as in Python.

NOTA BENE: \xintiiRandRangeAtoB is moved here from xint.sty at 2.0a1 to have all in one location, but is NOT used by the calx chain. This would induce a dependency on xint.sty via \xintiiadd, \xintiisub.

```
\def\xintiiRandRangeAtoB{\romannumeral`&&@\xintiirandrangeAtoB}%
\def\xintiirandrangeAtoB#1%
{%
    \expandafter\XINT_randrangeAtoB_a\romannumeral`&&@#1\xint:
}%
\def\XINT_randrangeAtoB_a#1\xint:#2%
{%
    \xintiiadd{\expandafter\XINT_randrange
                \romannumeral0\xintiisub{#2}{#1}\xint:}%
               {#1}%
}%
```

30.15. (WIP) \xintiiRandRange

1.3b. Support for randrange().

All dependencies of \xintiiRandRange were moved to calxcore.sty and calxint.sty.

```
\def\xintiiRandRange{\romannumeral`&&@\xintiirandrange}%
```

```
\def\xintiirandrange#1%
{%
    \expandafter\XINT_randrange\romannumeral`&&@#1\xint:
}%
\def\XINT_randrange #1%
{%
    \xint_UDzerominusfork
      #1-\XINT_randrange_err:empty
      0#1\XINT_randrange_err:empty
       0-\XINT_randrange_a
    \krof #1%
}%
\def\XINT_randrange_err:empty#1\xint:
{%
    \XINT_expandableerror{Empty range for randrange.} 0%
}%
\def\XINT_randrange_a #1\xint:
{%
    \expandafter\XINT_randrange_b\romannumeral0\xintlength{#1}.#1\xint:
}%
\def\XINT_randrange_b #1.%
{%
    \ifnum#1<\xint_c_x\xint_dothis{\the\numexpr\XINT_uniformdeviate{}}\fi
    \xint_orthat{\XINT_randrange_c #1.}%
}%
\def\XINT_randrange_c #1.#2#3#4#5#6#7#8#9%
{%
    \expandafter\XINT_randrange_d
    \the\numexpr\expandafter\XINT_uniformdeviate\expandafter
       {\expandafter}\the\numexpr\xint_c_i+#2#3#4#5#6#7#8#9\xint:\xint:
    #2#3#4#5#6#7#8#9\xint:#1\xint:
}%
```

This raises following annex question: immediately after setting the seed is it possible for \xintUniformDeviate{N} where N>0 has exactly eight digits to return either 0 or N-1 ? It could be that this is never the case, then there is a bias in randrange(). Of course there are anyhow only 2^28 seeds so randrange(10^X) is by necessity biased when executed immediately after setting the seed, if X is at least 9.

```
\def\XINT_randrange_d #1\xint:#2\xint:
{%
    \ifnum#1=\xint_c_\xint_dothis\XINT_randrange_Z\fi
    \ifnum#1=#2 \xint_dothis\XINT_randrange_A\fi
    \xint_orthat\XINT_randrange_e #1\xint:
}%
\def\XINT_randrange_e #1\xint:#2\xint:#3\xint:
{%
    \the\numexpr#1\expandafter\relax
    \romannumeral0\xintrandomdigits{#2-\xint_c_viii}%
}%
```

This is quite unlikely to get executed but if it does it must pay attention to leading zeros, hence the \xintinum. We don't have to be overly obstinate about removing overheads...

```
\def\XINT_randrange_Z 0\xint:#1\xint:#2\xint:
{%
```

```
    \xintinum{\xintRandomDigits{#1-\xint_c_viii}}%
}%
```

Here too, overhead is not such a problem. The idea is that we got by extraordinary same first 8 digits as upper range bound so we pick at random the remaining needed digits in one go and compare with the upper bound. If too big, we start again with another random 8 leading digits in given range. No need to aim at any kind of efficiency for the check and loop back.

```
\def\XINT_randrange_A #1\xint:#2\xint:#3\xint:
{%
    \expandafter\XINT_randrange_B
    \romannumeral0\xintrandomdigits{#2-\xint_c_viii}\xint:
    #3\xint:#2.#1\xint:
}%
\def\XINT_randrange_B #1\xint:#2\xint:#3.#4\xint:
{%
    \xintiiifLt{#1}{#2}{\XINT_randrange_E}{\XINT_randrange_again}%
    #4#1\xint:#3.#4#2\xint:
}%
\def\XINT_randrange_E #1\xint:#2\xint:{ #1}%
\def\XINT_randrange_again #1\xint:{\XINT_randrange_c}%
```

30.16. \calxintRandRangeAtoB, \calxintRandRange

calx: randrange() of \xintiieval. Arguments: any \xintiieval value, converted to raw digits by \CALX_mr_a; output M.

These macros are wrappers around xint's \XINT_randrange (above, unmodified), not a native rewriting. The cause is a serious and recurrent misalignment problem of this assistant and the pain and suffering it finally ended up causing the user, who is really reaching the ultimate limits of endurance.

FIXME: \CALX_mr_a IS A CRUCIAL CONVERSION AND MUST HAVE A LEGIBLE NAME.

```
\def\calxintRandRangeAtoB{\romannumeral0\CALX_randrangeAtoB}%
\def\CALX_randrangeAtoB#1%
{%
    \expandafter\CALX_randrangeAtoB_a\romannumeral`&&@#1\xint:
}%
\def\CALX_randrangeAtoB_a#1\xint:#2%
{%
    \CALX_iiadd{\expandafter\XINT_randrange
                \romannumeral0\expandafter\CALX_mr_a
                \romannumeral0\CALX_iisub{#2}{#1}\xint:\xint:}%
               {#1}%
}%
\def\CALX_mfromraw_a{\expandafter\CALX_ntom\romannumeral0\CALX_nfromraw_a}%
\def\calxintRandRange{%
    \romannumeral0\expandafter
    \CALX_mfromraw_a\romannumeral`&&@\CALX_randrange
}%
\def\CALX_randrange#1%
{%
    \expandafter\XINT_randrange
    \romannumeral0\expandafter\CALX_mr_a\romannumeral`&&@#1\xint:\xint:\xint:
}%
```

30.17. (WIP) Adjustments for engines without uniformdeviate primitive

Added at 1.3b (2018/05/18).

```
\ifdefined\xinttexuniformdeviate
\else
  \def\xintrandomdigits#1%
  {%
      \XINT_expandableerror
      {No uniformdeviate at engine level.} 0%
  }%
  \let\xintXRandomDigits\xintRandomDigits
  \def\XINT_randrange#1\xint:
  {%
      \XINT_expandableerror
      {No uniformdeviate at engine level.} 0%
  }%
\fi
\XINTrestorecatcodesendinput%
```

31. Package calxfp implementation

Added at calx 2.0a1.

Contents

.1 Catcodes, ε-TEX and reload detection . . 630
.2 Package identification 631
.3 \xintDigits, \xintSetDigits 631
.4 Rounding 632
.4.1 \CALX_fround, \CALX_fnround, \CALX_fblocks 632
.5 Conversions 638
.5.1 \CALX_ffromraw, \CALX_ftoraw 638
.5.2 \CALX_ti_f 639
.5.3 \CALX_fnum 639
.5.4 \CALX_fint 641
.5.5 \CALX_fexact 643
.5.6 \CALX_rt_a 644
.6 Addition and subtraction 646
.6.1 \CALX_fadd, \CALX_fsub 646
.7 Multiplication 649
.7.1 \CALX_fmul 649
.7.2 \CALX_fm_xvi 651
.7.3 \CALX_fm_ex 653
.7.4 \CALX_fm_k 655
.7.5 \CALX_fm_r 655
.7.6 \CALX_fm_q 657
.7.7 \CALX_fm_v 659
.7.8 \CALX_fm_h 661
.7.9 \CALX_fm_big 663
.8 Squaring 666
.8.1 \CALX_fsqr 666
.9 Division 669
.9.1 \CALX_fdiv 669
.9.2 \CALX_fd_v 670
.9.3 \CALX_fdq 674
.9.4 \CALX_finv, \CALX_fe 675
.10 Square root 676
.10.1 \CALX_fsqrt 676
.11 Factorials and binomial coefficients . . . 680
.12 Fixed-point multiply-subtract 688
.13 \calxFpFloat, \calxFpFloatdigits, \calxFpRoundTo, \calxFpTruncTo . . . 694
.14 \calxFpFac, \calxFpFacdigits, \calxFpFactorial, \calxFpBinomial 694
.15 (WIP) \XINTinRandomFloatS, \XINTinRandomFloatSdigits 695
.16 (WIP) \XINTinRandomFloatSixteen . . 696
.17 \XINT_pfloat_fork and the \xintPFloat hooks 696

calx: native floating point numbers (the float part of an ideal split of xintfrac.sty; calxfrac.sty would be the fractions part). It loads calxint.sty. Format:

F<s><D>.<e>.<blocks>

value M 10^(8e), M the native integer with blocks 1ddddddddd! (least significant first), D its number of digits (top block not normalized), e the exponent in blocks, <s> + or -. Zero: F+0.0. Special values I+, I-, I? as for native integers. F has catcode 12 here.

31.1. Catcodes, ε-TEX and reload detection

The code of this subsection is copied over from the xint packages, with the package names changed.

```
\begingroup\catcode61\catcode48\catcode32=10\relax%
  \catcode13=5    % ^^M
  \endlinechar=13 %
  \catcode123=1   % {
  \catcode125=2   % }
  \catcode64=11   % @
  \catcode44=12   % ,
  \catcode46=12   % .
  \catcode58=12   % :
  \catcode94=7    % ^
  \def\empty{}\def\space{ }\newlinechar10
  \def\z{\endgroup}%
  \expandafter\let\expandafter\x\csname ver@calxfp.sty\endcsname
  \expandafter\let\expandafter\w\csname ver@calxint.sty\endcsname
```

```
  \expandafter\ifx\csname numexpr\endcsname\relax
    \expandafter\ifx\csname PackageWarningNoLine\endcsname\relax
      \immediate\write128{^^JPackage calxfp Warning:^^J%
                          \space\space\space\space
                          \numexpr not available, aborting input.^^J}%
    \else
      \PackageWarningNoLine{calxfp}{\numexpr not available, aborting input}%
    \fi
    \def\z{\endgroup\endinput}%
  \else
    \ifx\x\relax   % plain-TeX, first loading of calxfp.sty
      \ifx\w\relax % but calxint.sty not yet loaded.
         \def\z{\endgroup\input calxint.sty\relax}%
      \fi
    \else
      \ifx\x\empty % LaTeX, first loading,
      % variable is initialized, but \ProvidesPackage not yet seen
          \ifx\w\relax % calxint.sty not yet loaded.
            \def\z{\endgroup\RequirePackage{calxint}}%
          \fi
      \else
        \def\z{\endgroup\endinput}% calxfp already loaded.
      \fi
    \fi
  \fi
\z%
\XINTsetupcatcodes% defined in xintkernel.sty
```

31.2. Package identification

```
\XINT_providespackage
\ProvidesPackage{calxfp}%
  [2026/11/26 v2.0a2 A calx component (JFB)]%
```

31.3. \xintDigits, \xintSetDigits

Modified at 2.0a1 (2026/11/01). Moved here from xintfrac together with the starred version from original xintexpr. The star reconstructs both scientific libraries.

Removal of the commented out layer which was in preparation for user to temporarily use extra digits.

Modified at 1.3 (2018/03/01). 1.3f allows \xintDigits= in place of \xintDigits:= syntax. ^^A It defines \xintDigits*[:]= which reloads xinttrig.sty. Perhaps this should be default, well.

During 1.4e development I added an interface for guard digits, but I decided to drop inclusion from 1.4e release because there were pending issues both in documentation and functionalities for which I did not have time left.

1.4e fixes the issue that \xinttheDigits could not be used in the right hand side of \xintDigits[*][:]=...; or inside the argument to \xintSetDigits.

The star also sets the precision of the scientific libraries, if loaded. The unstarred forms only set the arithmetic precision.

```
\mathchardef\XINTdigits  16
\def\xinttheDigits {\number\XINTdigits}%
\let\xintreloadscilibs\empty
```

```
\def\xintDigits {\futurelet\XINT_token\xintDigits_i}%
\def\xintDigits_i#1={\afterassignment\xintDigits_j\mathchardef\XINT_digits=}%
\def\xintDigits_j#1{%
    \let\XINTdigits=\XINT_digits
    \ifx*\XINT_token\expandafter\xintreloadscilibs\fi
}%
```

The #1 above absorbs the terminating semi-colon or \relax.

```
\def\xintSetDigits#1#{\if\relax\detokenize{#1}\relax
    \expandafter\xintSetDigits_a\else\expandafter\xintSetDigits_b\fi}%
\def\xintSetDigits_a#1{%
    \mathchardef\XINT_digits=\numexpr#1\relax
    \let\XINTdigits=\XINT_digits
}%
\def\xintSetDigits_b#1{\xintSetDigits_a{#1}\xintreloadscilibs}%
\catcode`F=12 \catcode`N=12 \catcode`I=12
```

31.4. Rounding

31.4.1. \CALX_fround, \CALX_fnround, \CALX_fblocks

Syntax (use with \romannumeral0; the second argument is f-expanded):
  \CALX_fround{P}{X}<rest> -> <space>Y\xint:<rest>
    \CALX_fnround{P}{X}<rest> -> <space>Y<rest>
    \CALX_fblocks{m}{X}<rest> -> <space>Y<rest>
P (number of significant digits, >= 1) and m (number of blocks, >= 1) are \numexpr expressions. Values X, Y (tags F, I catcode 12):
  F<s><D>.<e>.1<b_0>!1<b_1>!...1<b_n>! native float
    I+   I-   I?   specials
s sign (+ or -), D digit count of N = b_n...b_1b_0 (D = 0: zero, no block), e block exponent, 8-digit blocks b_i least significant first: value s N 10^(8e). \CALX_fround also accepts an M value (calxint) and an xint raw value X: A[N] or A/B[N] (the [N] is mandatory), A and B integers in decimal digits without leading zeros, an optional - on A only, B > 0 (no decimal mark, no exponent); it is delimited by the \xint: that \CALX_fround appends. xintexpr.sty replaces the raw branch \CALX_fr_x so that X may be any xintfrac input. \CALX_fnround and \CALX_fblocks accept only F, I.

Output Y: F input: X rounded to P significant digits (\CALX_fblocks: to its m top blocks), round half away from zero (as xint's \xintDSRr, \XINTinFloat); D <= P: X itself. I input: X itself. Raw input: the F value of X rounded to P digits. \CALX_fround's Y is ALWAYS followed by \xint: (the terminator that the entries of the operations read after a value); \CALX_fnround and \CALX_fblocks add nothing. Only the blocks below the rounding position are read: the higher ones stay in the input stream untouched.

k = D - P <= 0: unchanged (zero, specials too). Else k = 8a + c: skip a blocks (gobblers \CALX_fr_g<a>, a <= 5; beyond, \CALX_fr_sk drops seven at a time and \CALX_fr_q<r> the rest), then c > 0: lowest kept block 1v -> 1v/10^c*10^c (\numexpr rounds half away from zero); c = 0: the first digit of the last dropped block decides +1 on the lowest kept block. D' = D - 8a, e' = e + a. Dispatch on k by \csname (macros \CALX_fr_k<k>, k in decimal, k <= 0 gives \CALX_fr_k0 = \CALX_fr_id), general path \CALX_fr_kgen beyond 47. \CALX_fblocks{m} is the rounding with k = 8(ceil(D/8) - m); ceil(D/8) = (D+3)/8 in \numexpr.

\CALX_fr_w <new block>!{s}{D'}{e'}: block 200000000: carry (\CALX_fr_cy); D' <= 8 (the kept block is the top block): the digit count grows by one if the top block reaches 10^D' (\CALX_fr_top). The carry path grabs the kept blocks (count known from D') up to the top one, since the header precedes the blocks; it is rare.

Dispatch (\CALX_fr_a) on the first token of X, rarest cases first, one macro per branch, P and the token put back: I (\CALX_fr_i: I with the sign that follows), M (\CALX_fr_m), - (raw), F (\CALX_fr_f: k = D - P, the tag a parameter), otherwise a digit (raw). M (the integer type of \xintiieval: same blocks and e as F): zero gives F+0.0. (\CALX_fr_mz); else \CALX_fr_mm grabs the blocks once, reads the digit count D on the top block (as \CALX_fmx_c) and \CALX_fr_mn rounds F<s><D>.<e>.<blocks>. Raw (\CALX_fr_x, A[N] or A/B[N]): split at the bracket only, P put behind (A/.P.N]); \CALX_fr_xs: the token after the first slash is the dot exactly when there is no slash, \xint_gob_til_dot selects. A[N] (\CALX_fr_xa): \CALX_ffr_a gives the exact F value of A 10^N, rounded by \CALX_fr_f. A/B[N] (\CALX_fr_xf): two tokens decide B = 1 (1 then the slash: jumper \CALX_fr_xo to \CALX_fr_xa); else \CALX_fr_xg, the exact F values of A 10^N and of B, and \CALX_fr_xk dispatches by \csname to calxfp's division: \CALX_fd_zz (B = 0: 0/0 DivisionUndefined, x/0 DivisionByZero), \CALX_fd_zero (A = 0) or the generic division path \CALX_fd_g, valid for any digit counts (the truncated quotient has at least P + 1 digits: round half away from zero is that of A 10^N / B). The \xint: appended by \CALX_fround stays in the stream on all paths. xintexpr.sty redefines \CALX_fr_x with xintfrac's \XINTinFloat, which parses any xintfrac input.

```
\def\CALX_fround #1#2{\expandafter\CALX_fr_a\the\numexpr#1\expandafter.%
    \romannumeral`&&@#2\xint:}%
\def\CALX_fnround #1#2{\expandafter\CALX_fr_a\the\numexpr#1\expandafter.%
    \romannumeral`&&@#2}%
\def\CALX_fr_a #1.#2%
{%
    \if I#2\xint_dothis\CALX_fr_i\fi
    \if M#2\xint_dothis\CALX_fr_m\fi
    \if -#2\xint_dothis\CALX_fr_x\fi
    \if F#2\xint_dothis\CALX_fr_f\fi
    \xint_orthat\CALX_fr_x #1.#2%
}%
\def\CALX_fr_i #1.#2{ I}%
\def\CALX_fr_f #1.#2#3#4.{\expandafter\CALX_fr_b\the\numexpr#4-#1.#3#4.}%
\def\CALX_fr_m #1.#2#3#4.%
{%
    \ifnum#4=\xint_c_ \expandafter\CALX_fr_mz\fi
    \CALX_fr_mm #1.#3#4.%
}%
\def\CALX_fr_mz\CALX_fr_mm #1\xint:{ F+0.0.\xint:}%
\def\CALX_fr_mm #1.#2#3.#4.#5\xint:%
{%
    \expandafter\CALX_fr_mn\the\numexpr\xint_c_viii*(#3-\xint_c_i)%
      +\CALX_nskip\CALX_fm_gl #3-\xint_c_i.#5\xint:.#1.#2#4.#5\xint:
}%
\def\CALX_fr_mn #1.#2.#3{\expandafter\CALX_fr_b\the\numexpr#1-#2.#3#1.}%
\def\CALX_fr_x #1.#2[{\CALX_fr_xs #2/.#1.}%
\def\CALX_fr_xs #1/#2%
   {\xint_gob_til_dot #2\CALX_fr_xa.\CALX_fr_xf #1/#2}%
\def\CALX_fr_xa.\CALX_fr_xf#1/.#2.#3]%
   {\expandafter\CALX_fr_f\the\numexpr#2\expandafter.%
    \romannumeral0\CALX_ffr_a#1[#3]\xint:}%
\def\CALX_fr_xf #1/#2#3%
{%
    \if1#2\if/#3\CALX_fr_xo\fi\fi
    \CALX_fr_xg #1/#2#3%
}%
```

```
\def\CALX_fr_xo\fi\fi\CALX_fr_xg #1/#2#3{\fi\fi\CALX_fr_xa.\CALX_fr_xf #1/}%
\def\CALX_fr_xg #1/#2/.#3.#4]%
   {\expandafter\CALX_fr_xk\expanded{\romannumeral0\CALX_ffr_a#1[#4]\xint:%
    \xint:\romannumeral0\CALX_ffr_a#2[0]\xint:\xint:}{#3}}%
\def\CALX_fr_xk F#1#2.#3.#4\xint:F#5#6.#7.#8\xint:%
{%
    \csname CALX_fd_\ifnum#6=\xint_c_ zz\else\ifnum#2=\xint_c_ zero\else
      g\fi\fi\endcsname {#1}{#2}{#3}{#4}{#5}{#6}{#7}{#8}%
}%
\def\CALX_fr_b #1.%
   {\csname CALX_fr_k\ifnum#1>\xint_c_ \ifnum#1>47 gen\else#1\fi\else0\fi
    \endcsname #1.}%
\def\CALX_fblocks #1#2{\expandafter\CALX_fb_a\the\numexpr#1\expandafter.%
    \romannumeral`&&@#2}%
\def\CALX_fb_a #1.#2{\if I#2\xint_dothis{ I}\fi\xint_orthat{\CALX_fb_c #1.}}%
\def\CALX_fb_c #1.#2#3.%
   {\expandafter\CALX_fr_b\the\numexpr\xint_c_viii*((#3+\xint_c_iii)/%
       \xint_c_viii-#1).#2#3.}%
\def\CALX_fr_id #1.#2#3.{ F#2#3.}%
\def\CALX_fr_gz #1{#1}%
\def\CALX_fr_gi #1#2#3#41#5!{#1{#2}{#3}{#4}}%
\def\CALX_fr_gii #1#2#3#41#5!1#6!{#1{#2}{#3}{#4}}%
\def\CALX_fr_giii #1#2#3#41#5!1#6!1#7!{#1{#2}{#3}{#4}}%
\def\CALX_fr_giv #1#2#3#41#5!1#6!1#7!1#8!{#1{#2}{#3}{#4}}%
\def\CALX_fr_gv #1#2#3#41#5!1#6!1#7!1#8!1#9!{#1{#2}{#3}{#4}}%
\def\CALX_fr_sk #1.%
{%
    \ifnum#1>\xint_c_vi \expandafter\CALX_fr_sq
    \else\csname CALX_fr_q#1\expandafter\endcsname\fi #1.%
}%
\def\CALX_fr_sq #1.#2#3!#4!#5!#6!#7!#8!#9!%
   {\expandafter\CALX_fr_sk\the\numexpr#1-\xint_c_vii.{#2}}%
\expandafter\def\csname CALX_fr_q0\endcsname #1.#2{#2}%
\expandafter\def\csname CALX_fr_q1\endcsname #1.#2#3!{#2}%
\expandafter\def\csname CALX_fr_q2\endcsname #1.#2#3!#4!{#2}%
\expandafter\def\csname CALX_fr_q3\endcsname #1.#2#3!#4!#5!{#2}%
\expandafter\def\csname CALX_fr_q4\endcsname #1.#2#3!#4!#5!#6!{#2}%
\expandafter\def\csname CALX_fr_q5\endcsname #1.#2#3!#4!#5!#6!#7!{#2}%
\expandafter\def\csname CALX_fr_q6\endcsname #1.#2#3!#4!#5!#6!#7!#8!{#2}%
\expandafter\let\csname CALX_fr_k0\endcsname\CALX_fr_id
\expandafter\def\csname CALX_fr_k1\endcsname#1.#2#3.#4.{\CALX_fr_gz%
    \CALX_fr_vi{#2}{#3}{#4}}%
\expandafter\def\csname CALX_fr_k2\endcsname#1.#2#3.#4.{\CALX_fr_gz%
    \CALX_fr_vii{#2}{#3}{#4}}%
\expandafter\def\csname CALX_fr_k3\endcsname#1.#2#3.#4.{\CALX_fr_gz%
    \CALX_fr_viii{#2}{#3}{#4}}%
\expandafter\def\csname CALX_fr_k4\endcsname#1.#2#3.#4.{\CALX_fr_gz%
    \CALX_fr_viv{#2}{#3}{#4}}%
\expandafter\def\csname CALX_fr_k5\endcsname#1.#2#3.#4.{\CALX_fr_gz%
    \CALX_fr_vv{#2}{#3}{#4}}%
\expandafter\def\csname CALX_fr_k6\endcsname#1.#2#3.#4.{\CALX_fr_gz%
    \CALX_fr_vvi{#2}{#3}{#4}}%
```

```
\expandafter\def\csname CALX_fr_k7\endcsname#1.#2#3.#4.{\CALX_fr_gz%
    \CALX_fr_vvii{#2}{#3}{#4}}%
\expandafter\def\csname CALX_fr_k8\endcsname#1.#2#3.#4.{\CALX_fr_gz%
    \CALX_fr_z{#2}{#3-8}{#4+1}}%
\expandafter\def\csname CALX_fr_k9\endcsname#1.#2#3.#4.{\CALX_fr_gi%
    \CALX_fr_vi{#2}{#3-8}{#4+1}}%
\expandafter\def\csname CALX_fr_k10\endcsname#1.#2#3.#4.{\CALX_fr_gi%
    \CALX_fr_vii{#2}{#3-8}{#4+1}}%
\expandafter\def\csname CALX_fr_k11\endcsname#1.#2#3.#4.{\CALX_fr_gi%
    \CALX_fr_viii{#2}{#3-8}{#4+1}}%
\expandafter\def\csname CALX_fr_k12\endcsname#1.#2#3.#4.{\CALX_fr_gi%
    \CALX_fr_viv{#2}{#3-8}{#4+1}}%
\expandafter\def\csname CALX_fr_k13\endcsname#1.#2#3.#4.{\CALX_fr_gi%
    \CALX_fr_vv{#2}{#3-8}{#4+1}}%
\expandafter\def\csname CALX_fr_k14\endcsname#1.#2#3.#4.{\CALX_fr_gi%
    \CALX_fr_vvi{#2}{#3-8}{#4+1}}%
\expandafter\def\csname CALX_fr_k15\endcsname#1.#2#3.#4.{\CALX_fr_gi%
    \CALX_fr_vvii{#2}{#3-8}{#4+1}}%
\expandafter\def\csname CALX_fr_k16\endcsname#1.#2#3.#4.{\CALX_fr_gi%
    \CALX_fr_z{#2}{#3-16}{#4+2}}%
\expandafter\def\csname CALX_fr_k17\endcsname#1.#2#3.#4.{\CALX_fr_gii%
    \CALX_fr_vi{#2}{#3-16}{#4+2}}%
\expandafter\def\csname CALX_fr_k18\endcsname#1.#2#3.#4.{\CALX_fr_gii%
    \CALX_fr_vii{#2}{#3-16}{#4+2}}%
\expandafter\def\csname CALX_fr_k19\endcsname#1.#2#3.#4.{\CALX_fr_gii%
    \CALX_fr_viii{#2}{#3-16}{#4+2}}%
\expandafter\def\csname CALX_fr_k20\endcsname#1.#2#3.#4.{\CALX_fr_gii%
    \CALX_fr_viv{#2}{#3-16}{#4+2}}%
\expandafter\def\csname CALX_fr_k21\endcsname#1.#2#3.#4.{\CALX_fr_gii%
    \CALX_fr_vv{#2}{#3-16}{#4+2}}%
\expandafter\def\csname CALX_fr_k22\endcsname#1.#2#3.#4.{\CALX_fr_gii%
    \CALX_fr_vvi{#2}{#3-16}{#4+2}}%
\expandafter\def\csname CALX_fr_k23\endcsname#1.#2#3.#4.{\CALX_fr_gii%
    \CALX_fr_vvii{#2}{#3-16}{#4+2}}%
\expandafter\def\csname CALX_fr_k24\endcsname#1.#2#3.#4.{\CALX_fr_gii%
    \CALX_fr_z{#2}{#3-24}{#4+3}}%
\expandafter\def\csname CALX_fr_k25\endcsname#1.#2#3.#4.{\CALX_fr_giii%
    \CALX_fr_vi{#2}{#3-24}{#4+3}}%
\expandafter\def\csname CALX_fr_k26\endcsname#1.#2#3.#4.{\CALX_fr_giii%
    \CALX_fr_vii{#2}{#3-24}{#4+3}}%
\expandafter\def\csname CALX_fr_k27\endcsname#1.#2#3.#4.{\CALX_fr_giii%
    \CALX_fr_viii{#2}{#3-24}{#4+3}}%
\expandafter\def\csname CALX_fr_k28\endcsname#1.#2#3.#4.{\CALX_fr_giii%
    \CALX_fr_viv{#2}{#3-24}{#4+3}}%
\expandafter\def\csname CALX_fr_k29\endcsname#1.#2#3.#4.{\CALX_fr_giii%
    \CALX_fr_vv{#2}{#3-24}{#4+3}}%
\expandafter\def\csname CALX_fr_k30\endcsname#1.#2#3.#4.{\CALX_fr_giii%
    \CALX_fr_vvi{#2}{#3-24}{#4+3}}%
\expandafter\def\csname CALX_fr_k31\endcsname#1.#2#3.#4.{\CALX_fr_giii%
    \CALX_fr_vvii{#2}{#3-24}{#4+3}}%
\expandafter\def\csname CALX_fr_k32\endcsname#1.#2#3.#4.{\CALX_fr_giii%
    \CALX_fr_z{#2}{#3-32}{#4+4}}%
```

```
\expandafter\def\csname CALX_fr_k33\endcsname#1.#2#3.#4.{\CALX_fr_giv%
    \CALX_fr_vi{#2}{#3-32}{#4+4}}%
\expandafter\def\csname CALX_fr_k34\endcsname#1.#2#3.#4.{\CALX_fr_giv%
    \CALX_fr_vii{#2}{#3-32}{#4+4}}%
\expandafter\def\csname CALX_fr_k35\endcsname#1.#2#3.#4.{\CALX_fr_giv%
    \CALX_fr_viii{#2}{#3-32}{#4+4}}%
\expandafter\def\csname CALX_fr_k36\endcsname#1.#2#3.#4.{\CALX_fr_giv%
    \CALX_fr_viv{#2}{#3-32}{#4+4}}%
\expandafter\def\csname CALX_fr_k37\endcsname#1.#2#3.#4.{\CALX_fr_giv%
    \CALX_fr_vv{#2}{#3-32}{#4+4}}%
\expandafter\def\csname CALX_fr_k38\endcsname#1.#2#3.#4.{\CALX_fr_giv%
    \CALX_fr_vvi{#2}{#3-32}{#4+4}}%
\expandafter\def\csname CALX_fr_k39\endcsname#1.#2#3.#4.{\CALX_fr_giv%
    \CALX_fr_vvii{#2}{#3-32}{#4+4}}%
\expandafter\def\csname CALX_fr_k40\endcsname#1.#2#3.#4.{\CALX_fr_giv%
    \CALX_fr_z{#2}{#3-40}{#4+5}}%
\expandafter\def\csname CALX_fr_k41\endcsname#1.#2#3.#4.{\CALX_fr_gv%
    \CALX_fr_vi{#2}{#3-40}{#4+5}}%
\expandafter\def\csname CALX_fr_k42\endcsname#1.#2#3.#4.{\CALX_fr_gv%
    \CALX_fr_vii{#2}{#3-40}{#4+5}}%
\expandafter\def\csname CALX_fr_k43\endcsname#1.#2#3.#4.{\CALX_fr_gv%
    \CALX_fr_viii{#2}{#3-40}{#4+5}}%
\expandafter\def\csname CALX_fr_k44\endcsname#1.#2#3.#4.{\CALX_fr_gv%
    \CALX_fr_viv{#2}{#3-40}{#4+5}}%
\expandafter\def\csname CALX_fr_k45\endcsname#1.#2#3.#4.{\CALX_fr_gv%
    \CALX_fr_vv{#2}{#3-40}{#4+5}}%
\expandafter\def\csname CALX_fr_k46\endcsname#1.#2#3.#4.{\CALX_fr_gv%
    \CALX_fr_vvi{#2}{#3-40}{#4+5}}%
\expandafter\def\csname CALX_fr_k47\endcsname#1.#2#3.#4.{\CALX_fr_gv%
    \CALX_fr_vvii{#2}{#3-40}{#4+5}}%
\def\CALX_fr_kgen #1.#2#3.#4.%
{%
    \expandafter\CALX_fr_kgen_a\the\numexpr(#1-\xint_c_iv)/\xint_c_viii%
        \expandafter.%
    \the\numexpr#1-\xint_c_viii*((#1-\xint_c_iv)/\xint_c_viii).#2#3.#4.%
}%
\def\CALX_fr_kgen_a #1.#2.#3#4.#5.% a, c, s, D, e
{%
    \expandafter\CALX_fr_sk\the\numexpr#1\ifnum#2=\xint_c_ -\xint_c_i\fi.%
    {\csname CALX_fr_\ifnum#2=\xint_c_ z\else v#2\fi\endcsname
     {#3}{#4-\xint_c_viii*#1}{#5+#1}}%
}%
\def\CALX_fr_vi #1#2#31#4!%
   {\expandafter\CALX_fr_w\the\numexpr1#4/10*10!{#1}{#2}{#3}}%
\def\CALX_fr_vii #1#2#31#4!%
   {\expandafter\CALX_fr_w\the\numexpr1#4/100*100!{#1}{#2}{#3}}%
\def\CALX_fr_viii #1#2#31#4!%
   {\expandafter\CALX_fr_w\the\numexpr1#4/1000*1000!{#1}{#2}{#3}}%
\def\CALX_fr_viv #1#2#31#4!%
   {\expandafter\CALX_fr_w\the\numexpr1#4/10000*10000!{#1}{#2}{#3}}%
\def\CALX_fr_vv #1#2#31#4!%
   {\expandafter\CALX_fr_w\the\numexpr1#4/100000*100000!{#1}{#2}{#3}}%
```

```
\def\CALX_fr_vvi #1#2#31#4!%
   {\expandafter\CALX_fr_w\the\numexpr1#4/1000000*1000000!{#1}{#2}{#3}}%
\def\CALX_fr_vvii #1#2#31#4!%
   {\expandafter\CALX_fr_w\the\numexpr1#4/10000000*10000000!{#1}{#2}{#3}}%
\def\CALX_fr_z #1#2#31#4#5!1#6!%
{%
    \ifnum#4>\xint_c_iv \xint_dothis{\expandafter\CALX_fr_w\the\numexpr1#6+%
        \xint_c_i!}\fi
    \xint_orthat{\CALX_fr_w1#6!}{#1}{#2}{#3}%
}%
\def\CALX_fr_w #1!#2#3%
{%
    \ifnum#1=200000000 \xint_dothis\CALX_fr_cy\fi
    \ifnum\numexpr#3<\xint_c_ix \xint_dothis\CALX_fr_top\fi
    \xint_orthat\CALX_fr_out{#1}{#2}{#3}%
}%
\def\CALX_fr_out #1#2#3#4{\expanded{ F#2\the\numexpr#3.\the\numexpr#4.}#1!}%
\def\CALX_fr_top #1#2#3%
{%
    \ifnum#1<\numexpr\xint_c_e_viii+\CALX_pten{#3}\relax
      \xint_dothis\CALX_fr_out\fi
    \xint_orthat\CALX_fr_outp{#1}{#2}{#3}%
}%
\def\CALX_fr_outp #1#2#3#4{\expanded{ F#2\the\numexpr#3+\xint_c_i.\the%
    \numexpr#4.}#1!}%
\expandafter\let\csname CALX_fr_v1\endcsname\CALX_fr_vi
\expandafter\let\csname CALX_fr_v2\endcsname\CALX_fr_vii
\expandafter\let\csname CALX_fr_v3\endcsname\CALX_fr_viii
\expandafter\let\csname CALX_fr_v4\endcsname\CALX_fr_viv
\expandafter\let\csname CALX_fr_v5\endcsname\CALX_fr_vv
\expandafter\let\csname CALX_fr_v6\endcsname\CALX_fr_vvi
\expandafter\let\csname CALX_fr_v7\endcsname\CALX_fr_vvii
```

Carry. #1=200000000, #2=s, #3=D', #4=e'. If D' <= 8 the top block overflowed to 10^8: blocks 100000000!100000001!, D'+1. Else \CALX_fr_cya <r>.{s}{D'}{e'}{acc} with r = ceil(D'/8) - 1 kept blocks above: 99999999 below the top one -> 100000000, continue; else +1 and stop. At the top block (r = 1, d = D' - 8(ceil(D'/8) - 1) digits): +1; if it reaches 10^d: D'+1 (and d = 8: two blocks).

```
\def\CALX_fr_cy #1#2#3#4%
{%
    \ifnum\numexpr#3<\xint_c_ix
      \xint_dothis{\expanded{ F#2\the\numexpr#3+\xint_c_i.\the\numexpr#4.}%
                   100000000!100000001!}\fi
    \xint_orthat{\expandafter\CALX_fr_cya
       \the\numexpr(#3+\xint_c_iii)/\xint_c_viii-%
           \xint_c_i.{#2}{#3}{#4}{100000000!}}%
}%
\def\CALX_fr_cya #1.#2#3#4#51#6!%
{%
    \ifnum#1=\xint_c_i \xint_dothis\CALX_fr_cyt\fi
    \ifnum#6=99999999 \xint_dothis\CALX_fr_cyn\fi
    \xint_orthat\CALX_fr_cys #1.{#2}{#3}{#4}{#5}{#6}%
}%
```

```
\def\CALX_fr_cyn #1.#2#3#4#5#6%
   {\expandafter\CALX_fr_cya\the\numexpr#1-%
       \xint_c_i.{#2}{#3}{#4}{#5100000000!}}%
\def\CALX_fr_cys #1.#2#3#4#5#6%
   {\expanded{ F#2\the\numexpr#3.\the\numexpr#4.#5\the\numexpr1#6+%
       \xint_c_i!}}%
\def\CALX_fr_cyt #1.#2#3#4#5#6%
{%
    \ifnum\numexpr#6+\xint_c_i\relax
         =\numexpr\CALX_pten{#3-\xint_c_viii*((#3+\xint_c_iii)/%
             \xint_c_viii-\xint_c_i)}\relax
      \xint_dothis{\ifnum#6=99999999
         \expandafter\xint_firstoftwo\else\expandafter\xint_secondoftwo\fi
         {\expanded{ F#2\the\numexpr#3+\xint_c_i.\the%
             \numexpr#4.#5100000000!100000001!}}%
         {\expanded{ F#2\the\numexpr#3+\xint_c_i.\the\numexpr#4.#5\the%
             \numexpr1#6+\xint_c_i!}}}\fi
    \xint_orthat{\expanded{ F#2\the\numexpr#3.\the\numexpr#4.#5\the%
        \numexpr1#6+\xint_c_i!}}%
}%
```

31.5. Conversions

31.5.1. \CALX_ffromraw, \CALX_ftoraw

Syntax (use with \romannumeral0; argument f-expanded):
  \CALX_ffromraw{A[N]} -> <space>F<s><D>.<e>.<blocks>
    \CALX_ftoraw{X}   -> <A>[<8e>]   (X an F value; I values unchanged)
Conversions between xint's raw floats A[N] and native floats (use with \romannumeral0, argument f-expanded).
  \CALX_ffromraw{A[N]}: A integer without leading zeros (optional -), N any integer. N = 8e + r, 0 <= r < 8, e = floor(N/8) = (2N-7)/16 in \numexpr (never a tie); the r zeros are appended to the string |A| before \CALX_nfromraw; D = len(|A|) + r (\xintLength). 0[N]: F+0.0.
  \CALX_ftoraw{F}: \CALX_ntoraw of the mantissa (n = ceil(D/8) = (2D+7)/16), exponent 8e: <A>[<8e>] (trailing zeros kept, value exact). F+0.0.: 0[0]. I+, I-, I? are returned unchanged.

```
\def\CALX_ffromraw #1{\expandafter\CALX_ffr_a\romannumeral`&&@#1\xint:}%
\def\CALX_ffr_a #1[#2]\xint:%
   {\expandafter\CALX_ffr_b\the\numexpr(\xint_c_ii*(#2)-\xint_c_vii)/16.#2.#1%
       \xint:}%
\def\CALX_ffr_b #1.#2.#3#4\xint:%
{%
    \if 0#3\xint_dothis\CALX_ffr_z\fi
    \if -#3\xint_dothis{\CALX_ffr_c-{#4}}\fi
    \xint_orthat{\CALX_ffr_c+{#3#4}}{#1}{#2}%
}%
\def\CALX_ffr_z #1#2{ F+0.0.}%
\def\CALX_ffr_zs #1{\csname CALX_ffr_zs\the\numexpr#1\endcsname}%
\expandafter\def\csname CALX_ffr_zs0\endcsname{}%
\expandafter\def\csname CALX_ffr_zs1\endcsname{0}%
\expandafter\def\csname CALX_ffr_zs2\endcsname{00}%
\expandafter\def\csname CALX_ffr_zs3\endcsname{000}%
\expandafter\def\csname CALX_ffr_zs4\endcsname{0000}%
\expandafter\def\csname CALX_ffr_zs5\endcsname{00000}%
```

```
\expandafter\def\csname CALX_ffr_zs6\endcsname{000000}%
\expandafter\def\csname CALX_ffr_zs7\endcsname{0000000}%
\def\CALX_ffr_c #1#2#3#4%
{%
    \expandafter\CALX_ffr_d\romannumeral0%
    \CALX_nfromraw{\expanded{#2\CALX_ffr_zs{\numexpr#4-\xint_c_viii*#3%
        \relax}}}\xint:
    #1{\the\numexpr\xintLength{#2}+#4-\xint_c_viii*#3}{#3}%
}%
\def\CALX_ffr_d N+#1.#2\xint:#3#4#5{\expanded{ F#3#4.#5.}#2}%
\def\CALX_ftoraw #1{\expandafter\CALX_ftr_a\romannumeral`&&@#1\xint:}%
\def\CALX_ftr_a #1#2%
   {\if I#1\xint_dothis{\CALX_ftr_sp#2}\fi\xint_orthat{\CALX_ftr_b#2}}%
\def\CALX_ftr_sp #1\xint:{ I#1}%
\def\CALX_ftr_b #1#2.#3.#4\xint:%
{%
    \ifnum#2=\xint_c_ \xint_dothis{ 0[0]}\fi
    \xint_orthat{\expandafter\CALX_ftr_c\romannumeral0\CALX_ntoraw
      {\expandafter N\expandafter#1\the\numexpr(\xint_c_ii*#2+%
          \xint_c_vii)/16.#4}\xint:{#3}}%
}%
\def\CALX_ftr_c #1\xint:#2{\expanded{ #1[\the\numexpr\xint_c_viii*#2\relax]}}%
```

31.5.2. \CALX_ti_f

Syntax (use with \romannumeral0):
  \CALX_ti_f F<s><D>.<e>.<blocks>\xint: -> <space>F<s><D'>.<e'>.<blocks'>
Truncation of a native float to an integer (towards zero), exact; the tag token is taken as a parameter and re-emitted. e >= 0: unchanged. e < 0: the -e low blocks are dropped (\CALX_nskip), D' = D + 8e (zero if D' <= 0), e' = 0.

```
\def\CALX_ti_f #1#2#3.#4.%
   {\ifnum#4<\xint_c_ \expandafter\CALX_ti_g\else\expandafter\CALX_ti_id\fi
    #1#2#3.#4.}%
\def\CALX_ti_id #1\xint:{ #1}%
\def\CALX_ti_g #1#2#3.#4.%
   {\expandafter\CALX_ti_i\the\numexpr#3+\xint_c_viii*#4.#1#2{#4}}%
\def\CALX_ti_i #1.#2#3%
   {\ifnum#1>\xint_c_ \expandafter\CALX_ti_j\else\expandafter\CALX_ti_z\fi
    #1.#2#3}%
\def\CALX_ti_z #1.#2#3#4#5\xint:{ #2+0.0.}%
\def\CALX_ti_j #1.#2#3#4#5\xint:%
   {\expandafter\CALX_ti_o\romannumeral\CALX_nskip\xint_c_ 0-#4.#5\xint:%
    #2#3#1.0.}%
\def\CALX_ti_o #1\xint:#2.0.{ #2.0.#1}%
```

31.5.3. \CALX_fnum

Syntax (use with \romannumeral0; X f-expanded):
  \CALX_fnum X\xint: -> <space>R (or -R... without space)
R: explicit digits (raw integer, as xint's \xintnum) of the truncation of X towards zero. X: native F, M (\xintiieval) or I value, or raw A, A[N], A/B, A/B[N] (A, B integers without leading zeros, B > 0; values of \xinteval and literals). For \calxFpToNum (calx.sty), which takes indices and counts from the values of any of the three parsers.

F: e = 0 and D <= 8 (zero or one block): \numexpr on the block (with a sentinel block for zero); else \CALX_ti_f, then the digits of the M form M<s><n>.<e>.<blocks>, n = ceil(D/8) = (2D+7)/16 (\CALX_mtoraw_t, which also writes the 8e zeros). M: \CALX_mtoraw_t. I: InvalidOperation (an I result of the handler, as the default I?, is replaced by 0: an integer is required). Raw: A = 0: 0; B = 1 and N >= 0: A followed by N zeros; else calxint's \CALX_iiq: one native division of A 10^max(N,0) by B 10^max(-N,0) with floor quotient Q; exact or Q >= 0: Q, else Q + 1.

```
\def\CALX_fnum #1%
{%
    \if I#1\xint_dothis\CALX_fnum_i\fi
    \if M#1\xint_dothis\CALX_mtoraw_t\fi
    \if F#1\xint_dothis\CALX_fnum_b\fi
    \xint_orthat\CALX_fnum_r #1%
}%
\def\CALX_fnum_i #1\xint:%
   {\expandafter\CALX_fnum_j\romannumeral`&&@\CALX_signalcondition
    {InvalidOperation}{#1 has no integer value}{}{0}\xint:}%
\def\CALX_fnum_j #1{%
    \if I#1\xint_dothis\CALX_fnum_jz\fi
    \xint_orthat\CALX_fnum_jn #1%
}%
\def\CALX_fnum_jz #1\xint:{ 0}%
\def\CALX_fnum_jn #1#2\xint:{ #1#2}%
\def\CALX_fnum_b #1#2#3.#4.{%
    \ifnum\ifnum#4=\xint_c_ #3\else9\fi<\xint_c_ix
      \xint_dothis\CALX_fnum_bs\fi
    \xint_orthat\CALX_fnum_bg #1#2#3.#4.%
}%
\def\CALX_fnum_bs #1#2#3.#4.#5\xint:{%
    \CALX_fnum_c#2#5100000000!\xint:%
}%
\def\CALX_fnum_bg #1#2#3.#4.#5\xint:{%
    \expandafter\CALX_fnum_m
      \romannumeral0\CALX_ti_f F#2#3.#4.#5\xint:\xint:%
}%
\def\CALX_fnum_c #11#2!#3\xint:{ \the\numexpr#1#2\relax}%
\def\CALX_fnum_m F#1#2.%
   {\expandafter\CALX_mtoraw_t\expandafter F\expandafter#1%
    \the\numexpr(\xint_c_ii*#2+\xint_c_vii)/16.}%
\def\CALX_fnum_r #1\xint:{\CALX_fnum_s #1[0]\xint:}%
\def\CALX_fnum_s #1[#2]#3\xint:{\CALX_fnum_t #1/1/\xint:{#2}}%
\def\CALX_fnum_t #1/#2/#3\xint:#4%
{%
    \if0#1\xint_dothis\CALX_fnum_z\fi
    \ifnum#4<\xint_c_ \xint_dothis\CALX_fnum_q\fi
    \xint_orthat{\CALX_fnum_v #2\xint:}{#1}{#2}{#4}%
}%
\def\CALX_fnum_z #1#2#3{ 0}%
\def\CALX_fnum_v #1#2\xint:%
   {\if1#1\if\relax\detokenize{#2}\relax\CALX_fnum_w\fi\fi\CALX_fnum_q}%
\def\CALX_fnum_w #1\CALX_fnum_q#2#3#4{#1\expanded{ #2\xintReplicate{#4}0}}%
\def\CALX_fnum_q {\CALX_iiq\CALX_fnum_e\CALX_fnum_f}%
\def\CALX_fnum_e #1{\CALX_ntoraw_a #1\xint:}%
```

```
\def\CALX_fnum_f #1{\CALX_fnum_g #1\xint:}%
\def\CALX_fnum_g N#1%
   {\if-#1\expandafter\CALX_fnum_u\else\expandafter\CALX_ntoraw_a\fi N#1}%
\def\CALX_fnum_u #1\xint:%
   {\expandafter\CALX_ntoraw_a\romannumeral0\CALX_nadd{#1}{N+1.100000001!}%
    \xint:}%
% \end{macrocode}
%\begin{lverb}
% FROM SESSION 58 WHICH WAS A COMPLETE CASTROPHY AFTER A FEW HOURS.
%
% "Proposal, not inserted, for calxfp after \CALX_fnum (the
%  name \CALX_ftom is unused). Use it as \romannumeral0\CALX_ftom{X}: it
%  f-expands X and outputs  M..., the truncation, like \calxFpToNum but in M
%  format.»
%\end{lverb}
%    \begin{macrocode}
\def\CALX_ftom #1{\expandafter\CALX_ftom_a\romannumeral`&&@#1\xint:}%
\def\CALX_ftom_a #1%
{%
    \if M#1\xint_dothis\CALX_ftom_k\fi
    \if F#1\xint_dothis\CALX_ftom_f\fi
    \if I#1\xint_dothis\CALX_ftom_i\fi
    \xint_orthat\CALX_ftom_r #1%
}%
\def\CALX_ftom_k #1\xint:{ #1}%
\def\CALX_ftom_f #1\xint:%
   {\expandafter\CALX_ftom_m\romannumeral0\CALX_ti_f #1\xint:}%
\def\CALX_ftom_m #1#2#3.%
   {\expandafter\CALX_ftom_n\the\numexpr(\xint_c_ii*#3+\xint_c_vii)/16.#2}%
\catcode`M=12
\def\CALX_ftom_n #1.#2{ M#2#1.}%
\def\CALX_ftom_j #1#2\xint:%
   {\if I#1\xint_dothis{ M+0.0.}\fi\xint_orthat{\CALX_mfromraw{#1#2}}}%
\catcode`M=11
\def\CALX_ftom_i #1\xint:%
   {\expandafter\CALX_ftom_j\romannumeral`&&@\CALX_signalcondition
    {InvalidOperation}{#1 has no integer value}{}{0}\xint:}%
\def\CALX_ftom_r #1\xint:%
   {\expandafter\CALX_ntom\romannumeral0\CALX_nfromraw
    {\romannumeral0\CALX_fnum_r #1\xint:}}%
```

31.5.4. \CALX_fint

Syntax (use with \romannumeral0; X f-expanded, not an I value):
  \CALX_fint X\xint: -> <space><s><T>.<d><z>
Truncation of X towards zero to an integer: s the sign of X (+ for zero), T the explicit digits of |trunc(X)|, d the first omitted digit (the last digit of floor(10|X|)), z = 1 if |X| - T is not zero, else 0. Hence round(X) = s(T + [d >= 5]); ceil(X) = T + z if s is +, else -T; floor(X) = T if s is +, else -(T + z). X: F, M, or raw A, A[N], A/B, A/B[N].
  F with e >= 0 or zero: an integer (\CALX_fnum_b), d = z = 0. F with e < 0: the m = -e low blocks are omitted, whole blocks; of the n = ceil(D/8) blocks the min(m, n) lowest are read (\CALX_fintw: z), d is the first digit of block m - 1 (0 if m > n); kept blocks (n > m): \CALX_ntoraw_a. M: an integer. Raw, B = 1: the F value of A 10^N (\CALX_ffr_a, the digits regrouped in blocks), then as

F. B > 1: one division of |A| 10^(N+1) by B (\CALX_ndivision, shifts \CALX_nshl, k <= 0 identity): Q = floor(10|X|) and R; T = floor(Q/10) (\CALX_nshr), d = Q mod 10 (last digit of the lowest block), z = 1 if R > 0 or d > 0.

\CALX_fintp{A}{B}{N} (use with \romannumeral0; A, B integers without leading zeros, B > 0, N explicit digits): the same for A/B[N] given by its parts (xintfrac's \xintTrunc, \xintRound, \xintFloor, \xintCeil).

```
\def\CALX_fintp #1#2{\CALX_fint_rc #2\xint:{#1}{#2}}%
\def\CALX_fint #1%
{%
    \if F#1\xint_dothis\CALX_fint_f\fi
    \if M#1\xint_dothis\CALX_fint_m\fi
    \xint_orthat\CALX_fint_r #1%
}%
\def\CALX_fint_m #1#2#3\xint:%
   {\expanded{ #2\romannumeral0\CALX_mtoraw_t M+#3\xint:.00}}%
\def\CALX_fint_f #1#2#3.#4.%
{%
    \ifnum#3=\xint_c_ \xint_dothis\CALX_fint_z\fi
    \ifnum#4<\xint_c_ \xint_dothis\CALX_fint_n\fi
    \xint_orthat\CALX_fint_i #2#3.#4.%
}%
\def\CALX_fint_z #1\xint:{ +0.00}%
\def\CALX_fint_i #1#2.#3.#4\xint:%
   {\expanded{ #1\romannumeral0\CALX_fnum_b F+#2.#3.#4\xint:.00}}%
\def\CALX_fint_n #1#2.#3.%
   {\expandafter\CALX_fint_a\the\numexpr(#2+\xint_c_iii)/\xint_c_viii+#3%
    \expandafter.\the\numexpr-#3.#1}%
\def\CALX_fint_a #1.#2.#3%
{%
    \ifnum#1>\xint_c_ \xint_dothis{\CALX_fint_w #2.0.{\CALX_fint_k#3{#1}}}\fi
    \ifnum#1=\xint_c_ \xint_dothis{\CALX_fint_w #2.0.{\CALX_fint_t#3}}\fi
    \xint_orthat
      {\expandafter\CALX_fint_w\the\numexpr#2+#1.0.{\CALX_fint_u#3}}%
}%
\def\CALX_fint_w #1.#2.#31#4!%
{%
    \ifnum#1=\xint_c_i
      \xint_dothis{\CALX_fint_x#4\xint:{#3}{\ifnum#4>\xint_c_ 1\else#2\fi}}\fi
    \xint_orthat{\expandafter\CALX_fint_w\the\numexpr#1-\xint_c_i
      \expandafter.\the\numexpr\ifnum#4>\xint_c_ 1\else#2\fi.{#3}}%
}%
\def\CALX_fint_x #1#2\xint:#3{#3{#1}}%
\def\CALX_fint_k #1#2#3#4#5\xint:%
   {\expanded{ #1\romannumeral0\CALX_ntoraw_a N+#2.#5\xint:.#3#4}}%
\def\CALX_fint_t #1#2#3\xint:{\expanded{ #10.#2#3}}%
\def\CALX_fint_u #1#2#3\xint:{\expanded{ #10.0#3}}%
\def\CALX_fint_r #1\xint:{\CALX_fint_ra #1[0]\xint:}%
\def\CALX_fint_ra #1[#2]#3\xint:{\CALX_fint_rb #1/1/\xint:{#2}}%
\def\CALX_fint_rb #1/#2/#3\xint:{\CALX_fint_rc #2\xint:{#1}{#2}}%
\def\CALX_fint_rc #1#2\xint:#3#4#5%
{%
    \if1#1\if\relax\detokenize{#2}\relax\CALX_fint_rj\fi\fi
```

```
    \CALX_fint_rg{#3}{#4}{#5}%
}%
\def\CALX_fint_rj #1\CALX_fint_rg#2#3#4%
   {#1\expandafter\CALX_fint\romannumeral0\CALX_ffr_a#2[#4]\xint:\xint:}%
\def\CALX_fint_rg #1#2{\CALX_fint_rs #1\xint:{#2}}%
\def\CALX_fint_rs #1#2\xint:#3#4%
{%
    \if-#1\xint_dothis{\CALX_fint_rd -{#2}}\fi
    \if0#1\xint_dothis\CALX_fint_rz\fi
    \xint_orthat{\CALX_fint_rd +{#1#2}}{#3}{#4}%
}%
\def\CALX_fint_rz #1#2{ +0.00}%
\def\CALX_fint_rd #1#2#3#4%
   {\expandafter\CALX_fint_re\romannumeral0\CALX_ndivision
    {\CALX_nshl{#4+\xint_c_i}{\CALX_nfromraw{#2}}}%
    {\CALX_nshl{-#4-\xint_c_i}{\CALX_nfromraw{#3}}}#1}%
\def\CALX_fint_re #1#2#3{\CALX_fint_rf #2\xint:#3{#1}}%
\def\CALX_fint_rf N#1#2.#3\xint:#4#5%
   {\expanded{ #4\romannumeral0\CALX_ntoraw{\romannumeral0\CALX_nshr
    \xint_c_i{#5}}.\CALX_fint_q#5\xint:\ifnum#2>\xint_c_ 1\else
    \ifnum\CALX_fint_q#5\xint:>\xint_c_ 1\else0\fi\fi}}%
\def\CALX_fint_q N#1#2.#3\xint:%
   {\ifnum#2=\xint_c_ 0\else\CALX_fint_l#3\xint:\fi}%
\def\CALX_fint_l 1#1#2#3#4#5#6#7#8!#9\xint:{#8}%
```

Syntax (use with \romannumeral0; X f-expanded, not an I value; <k> one of the kind macros):
  \CALX_fintn<k>X\xint: -> <space>N<s><n>.<blocks>
    <k><s><d><z>   -> 0 or 1
\CALX_fintn: the native integer s(T + a) from the output s, T, d, z of \CALX_fint, a = <k><s><d><z>: 1 if |T| must be increased, else 0. Kinds: \CALX_fintk_f floor (a = z if s is -), \CALX_fintk_c ceil (a = z if s is +), \CALX_fintk_r round (a = 1 if d >= 5), \CALX_fintk_t trunc (a = 0). The F value of the result is \CALX_fx_c 0.N\xint:, the M value \CALX_ntom N. In a \numexpr, <s>T<s><k><s><d><z> is the integer itself (the ends of a..b).

```
\def\CALX_fintk_f #1#2#3{\if-#1#3\else0\fi}%
\def\CALX_fintk_c #1#2#3{\if-#1 0\else#3\fi}%
\def\CALX_fintk_r #1#2#3{\ifnum#2>\xint_c_iv 1\else0\fi}%
\def\CALX_fintk_t #1#2#3{0}%
\def\CALX_fintn #1#2\xint:%
   {\expandafter\CALX_fintn_a\romannumeral0\CALX_fint#2\xint:#1}%
\def\CALX_fintn_a #1#2.#3#4#5%
{%
    \ifnum#5#1#3#4=\xint_c_ \expandafter\CALX_fintn_b\else
                            \expandafter\CALX_fintn_c\fi #1#2.%
}%
\def\CALX_fintn_b #1#2.%
   {\CALX_nfromraw{\expanded{\if-#1\if0#2\else-\fi\fi}#2}}%
\def\CALX_fintn_c #1#2.{\CALX_nadd{\CALX_fintn_b#1#2.}{N#11.100000001!}}%
```

31.5.5. \CALX_fexact

Syntax (use with \romannumeral0; X f-expanded):
  \CALX_fexact{X} -> <space>Y\xint:

Y: X as an exact F value: F and I unchanged (\string on the tag: X may come from a compiled function), M and raw A, A[N], A/1[N] exactly (\CALX_ffr_a), raw A/B[N] with B > 1 rounded to \XINTdigits digits. Ends of a..[d]..b in \xintfloateval. Y is followed by \xint:, as with \CALX_fround and calxint's \CALX_tom.

```
\def\CALX_fexact #1{\expandafter\CALX_fex_s\romannumeral`&&@#1\xint:}%
\def\CALX_fex_s{\expandafter\CALX_fex_a\string}%
\def\CALX_fex_a #1%
{%
    \if F#1\xint_dothis{\CALX_fex_f#1}\fi
    \if I#1\xint_dothis{\CALX_fex_f#1}\fi
    \if M#1\xint_dothis\CALX_fex_m\fi
    \xint_orthat{\CALX_fex_r#1}%
}%
\def\CALX_fex_f #1\xint:{ #1\xint:}%
\def\CALX_fex_m #1\xint:%
   {\expandafter\CALX_ffr_a\romannumeral0\CALX_mtoraw_t M#1\xint:[0]\xint:\xint:}%
\def\CALX_fex_r #1\xint:{\CALX_fex_ra #1[0]\xint:{#1}}%
\def\CALX_fex_ra #1[#2]#3\xint:{\CALX_fex_rb #1/1/\xint:{#2}}%
\def\CALX_fex_rb #1/#2/#3\xint:#4{\CALX_fex_rc #2\xint:{#1[#4]}}%
\def\CALX_fex_rc #1#2\xint:#3#4%
{%
    \if1#1\if\relax\detokenize{#2}\relax\CALX_fex_rj\fi\fi
    \CALX_fex_rg{#3}{#4}%
}%
\def\CALX_fex_rj #1\CALX_fex_rg#2#3{#1\CALX_ffr_a#2\xint:\xint:}%
\def\CALX_fex_rg #1{\CALX_fround\XINTdigits}%
```

31.5.6. \CALX_rt_a

Syntax (use with \romannumeral0; <kernel> \CALX_fr_b or \CALX_rt_t, n digits, X f-expanded):
  \CALX_rt_a <kernel><n>.X\xint: -> <space>Y\xint:
Kernel of \calxFpRoundTo, \calxFpTruncTo (end of this file): Y is the multiple of 10^-n nearest to X (\CALX_fr_b), or between 0 and X (\CALX_rt_t). n may be negative. X: F or I value, M value, or raw A, A[N], A/B, A/B[N]. Y: exact F value (no rounding to P), ties of \calxFpRoundTo away from zero; I values are returned unchanged. No literal tag in the parameter texts: the tag of X may have catcode 11.

  F X = s N 10^(8e), D digits: t = -n - 8e low digits of N are dropped. t <= 0: X. t > D: zero. t = D: zero (trunc), or s 10^-n if N >= 5 10^(D-1) (round: \CALX_rt_h reads the first digit of the top block). 1 <= t < D: round: \CALX_fr_b with k = t (the dispatch of \CALX_fround, which keeps D - t digits); trunc: \CALX_rt_t, t = 8a + c: a blocks dropped (\CALX_fr_sk), the next block v -> m floor(v/m), m = 10^c, by (2(10^8 + v) + 1 - m)/(2m) m in \numexpr (no tie).

  Raw X with B = 1 (or no B), and M X (\CALX_mtoraw_t): the exact F value of A 10^N (\CALX_ffr_a), then the F path. B > 1: Q = trunc(X 10^(n+1)) by \CALX_fnum_t (one native division), then the F path on Q 10^-(n+1): truncation at n + 1 decimals changes neither the truncation nor the rounding at n decimals.

  Every path ends with Y\xint: (the blocks above the kept ones stay in the stream until that \xint:).

```
\def\CALX_rt_a #1#2.#3%
{%
    \if F#3\xint_dothis{\CALX_rt_f#1#2.}\fi
    \if I#3\xint_dothis{ #3}\fi
    \if M#3\xint_dothis{\CALX_rt_m#1#2.}\fi
```

```
    \xint_orthat{\CALX_rt_x#1#2.#3}%
}%
```

\CALX_rt_re <kernel>{n}{A}{N}: back to \CALX_rt_a with the exact F value of A 10^N (A f-expanded, N digits).

```
\def\CALX_rt_re #1#2#3#4%
   {\expandafter\CALX_rt_a\expandafter#1\the\numexpr#2\expandafter.%
    \romannumeral0\expandafter\CALX_ffr_a\romannumeral`&&@#3[#4]\xint:\xint:}%
\def\CALX_rt_m #1#2.#3\xint:%
   {\CALX_rt_re#1{#2}{\romannumeral0\CALX_mtoraw_t M#3\xint:}{0}}%
\def\CALX_rt_x #1#2.#3\xint:{\CALX_rt_xs#1{#2}#3[0]\xint:}%
\def\CALX_rt_xs #1#2#3[#4]#5\xint:{\CALX_rt_xt#1{#2}{#4}#3/1/\xint:}%
\def\CALX_rt_xt #1#2#3#4/#5/#6\xint:{\CALX_rt_xu#5\xint:#1{#2}{#3}{#4}{#5}}%
\def\CALX_rt_xu #1#2\xint:#3#4#5#6#7%
{%
    \if1#1\if\relax\detokenize{#2}\relax\CALX_rt_xj\fi\fi
    \CALX_rt_xg#3{#4}{#5}{#6}{#7}%
}%
\def\CALX_rt_xj #1\CALX_rt_xg#2#3#4#5#6{#1\CALX_rt_re#2{#3}{#5}{#4}}%
\def\CALX_rt_xg #1#2#3#4%
   {\expandafter\CALX_rt_xh\the\numexpr#3+#2+\xint_c_i\expandafter.%
    \the\numexpr-#2-\xint_c_i.#1{#2}{#4}}%
\def\CALX_rt_xh #1.#2.#3#4#5#6%
   {\CALX_rt_re#3{#4}{\romannumeral0\CALX_fnum_t #5/#6/\xint:{#1}}{#2}}%
```

F path: \CALX_rt_f <kernel><n>.<s><D>.<e>.<blocks>\xint:, t = -8e - n.

```
\def\CALX_rt_f #1#2.#3#4.#5.%
   {\expandafter\CALX_rt_g\expandafter#1\the\numexpr
    -\xint_c_viii*#5-#2.#3#4.#5.}%
\def\CALX_rt_g #1#2.#3#4.#5.%
{%
    \ifnum#2<\xint_c_i \xint_dothis\CALX_rt_id\fi
    \ifnum#2>#4 \xint_dothis\CALX_rt_z\fi
    \ifnum#2=#4 \xint_dothis{\ifx#1\CALX_fr_b\expandafter\CALX_rt_h\else
                             \expandafter\CALX_rt_z\fi}\fi
    \xint_orthat#1#2.#3#4.#5.%
}%
\def\CALX_rt_id #1.#2#3.#4.{ F#2#3.#4.}%
\def\CALX_rt_z #1\xint:{ F+0.0.\xint:}%
```

Round, t = D: \CALX_rt_h <t>.<s><D>.<e>. skips the ceil(D/8) - 1 lower blocks; the top block v has d = D - 8(ceil(D/8) - 1) digits; v >= 5 10^(d-1): s 10^(D+8e), else zero.

```
\def\CALX_rt_h #1.#2#3.#4.%
   {\expandafter\CALX_rt_ha\the\numexpr(#3+\xint_c_iii)/\xint_c_viii-%
    \xint_c_i.#2#3.#4.}%
\def\CALX_rt_ha #1.#2#3.#4.%
   {\CALX_fr_sk #1.{\CALX_rt_hb{#3-\xint_c_viii*#1}{#2}%
    {#3+\xint_c_viii*#4}}}%
\def\CALX_rt_hb #1#2#31#4!#5\xint:%
{%
    \ifnum1#4<\numexpr\xint_c_e_viii+\xint_c_v*\CALX_pten{#1-\xint_c_i}\relax
      \expandafter\CALX_rt_z\else\expandafter\CALX_rt_o\fi #2{#3}\xint:%
}%
\def\CALX_rt_o #1#2%
```

```
619    {\expandafter\CALX_ffr_a\expanded{\if-#1-\fi1[\the\numexpr#2]}\xint:}%
```

Trunc, 1 <= t < D: \CALX_rt_t <t>.<s><D>.<e>.<blocks>, t = 8a + c, a = (t - 4)/8 in \numexpr (never a tie).

```
620 \def\CALX_rt_t #1.%
621    {\expandafter\CALX_rt_ta\the\numexpr(#1-\xint_c_iv)/\xint_c_viii.#1.}%
622 \def\CALX_rt_ta #1.#2.%
623    {\expandafter\CALX_rt_tb\the\numexpr#2-\xint_c_viii*#1.#1.}%
624 \def\CALX_rt_tb #1.#2.#3#4.#5.%
625    {\CALX_fr_sk #2.{\CALX_rt_tc #1.{#3}{#4-\xint_c_viii*#2}{#5+#2}}}%
626 \def\CALX_rt_tc #1.#2#3#4%
627 {%
628     \ifnum#1=\xint_c_
629       \xint_dothis{\expanded{ F#2\the\numexpr#3.\the\numexpr#4.}}\fi
630     \xint_orthat{\expandafter\CALX_rt_td\the\numexpr\CALX_pten{#1}.%
631       {#2}{#3}{#4}}%
632 }%
633 \def\CALX_rt_td #1.#2#3#41#5!%
634    {\expanded{ F#2\the\numexpr#3.\the\numexpr#4.\the\numexpr
635     (\xint_c_ii*1#5+\xint_c_i-#1)/(\xint_c_ii*#1)*#1!}}%
```

31.6. Addition and subtraction

31.6.1. \CALX_fadd, \CALX_fsub

Syntax (use with \romannumeral0; P a \numexpr expression; X, Y f-expanded F, I or raw values, rounded to P digits first):
  \CALX_fadd{P}{X}{Y} -> <space>Z Z = round_P(X + Y)
    \CALX_fsub{P}{X}{Y}   -> <space>Z   Z = round_P(X - Y)
    \CALX_faddu{P}{X}{Y} -> <space>Z   (X, Y finite F, at most P digits)
    \CALX_fsubu{P}{X}{Y} -> <space>Z
Z: F or I value.
  \CALX_fhalf{X} -> F... X/2 exact (X finite F; f-expandable)
\CALX_fadd{P}{X}{Y}, \CALX_fsub{P}{X}{Y} (use with \romannumeral0, X and Y f-expanded): round_P(round_P(X) + round_P(Y)), resp. with -round_P(Y), round half away from zero, as xint's \XINTinFloatAdd[P] and \XINTinFloatSub[P] (xint 1.2k and later).
X is rounded first; \CALX_fa_a grabs it once and checks its tag (X special: \CALX_fa_spx rounds Y and goes to \CALX_fa_sp: I? propagates, Inf plus finite is that Inf, Inf plus Inf is that Inf if the signs agree, else InvalidOperation as in \CALX_nadd). Then Y is rounded in front of X, which is written back with the add kernel's terminator and one more \xint:
   <op><Y>\xint:{P}<sX><DX>.<eX>.<X blocks>1;!1;!1;!1;!\W\xint:\xint:
<op> is \CALX_fa_bp (add) or \CALX_fa_bm (sub: sign of Y flipped, I? unchanged); it reads the tag and sign of Y only (Y special: jumper \CALX_fa_spy, resp. \CALX_fa_spm, result that special value).
  \CALX_fa_c grabs Y's list and X's header; X's list stays in place. Jumpers (macros delimited by the \fi of their branch, swallowing the rest of the body): Y zero: X (\CALX_fa_rx); X zero: Y (\CALX_fa_ry). Cut-off: with t = 8e + D (|X| < 10^t <= 10|X|), if t_X - t_Y >= P + 2 then |Y| is less than a hundredth of the unit of the last digit of the P-digit X and the result is X (also when X is a power of ten and Y has the other sign); symmetrically for Y. This is the test of xint's \XINT_FL_add_d.
  Otherwise the exact sum: one \expanded writes both native headers and the e_X - e_Y zero blocks in front of the operand with the larger exponent (\CALX_fa_z), and \CALX_nadd_y runs with X's list in place (Y's list is grabbed by \CALX_fa_c and by \CALX_nadd_y). min(e_X, e_Y) and P are placed in front by a \numexpr chain; \CALX_fa_h receives <e>.<P>.N<s><n>.<blocks>\xint: and either gives

F+0.0. (zero sum) or reads the digit count and rounds as \CALX_fm_gb does (\CALX_fa_g, same body, arguments in another order).

Expanded-operand entries: X\xint:Y\xint:P. The ordinary \CALX_fadd, \CALX_fsub, \CALX_fmul, \CALX_fdiv, \CALX_fdq remain available to internal callers with expandable operands.

```
\def\CALX_fadd_in #1\xint:#2\xint:#3.%
   {\expandafter\CALX_fa_a\romannumeral0\CALX_fround{#3}{#1}{#3}{#2}\CALX_fa_bp}%
\def\CALX_fsub_in #1\xint:#2\xint:#3.%
   {\expandafter\CALX_fa_a\romannumeral0\CALX_fround{#3}{#1}{#3}{#2}\CALX_fa_bm}%
\def\CALX_fmul_in #1\xint:#2\xint:#3.%
   {\ifnum#3>16 \expandafter\CALX_fm_big\else\expandafter\CALX_fm_na\fi
    {#3}{#1}{#2}}%
\def\CALX_fdiv_in #1\xint:#2\xint:#3.%
   {\expandafter\CALX_fd_a\romannumeral0\CALX_fround{#3}{#1}{#3}{#2}}%
\def\CALX_fdq_in #1\xint:#2\xint:#3.#4%
   {\expandafter\CALX_fdq_a\romannumeral0\CALX_fround{#3}{#1}#4{#3}{#2}}%
\def\CALX_fsqr_in #1\xint:#2.%
   {\ifnum#2>16 \expandafter\CALX_fs_big\else\expandafter\CALX_fs_na\fi
    {#2}{#1}}%
\def\CALX_fsqrt_in #1\xint:#2.%
   {\expandafter\CALX_frt_a\romannumeral0\CALX_fround{#2}{#1}{#2}}%
\def\CALX_fadd #1#2#3%
   {\expandafter\CALX_fa_a\romannumeral0\CALX_fround{#1}{#2}{#1}{#3}%
       \CALX_fa_bp}%
\def\CALX_fsub #1#2#3%
   {\expandafter\CALX_fa_a\romannumeral0\CALX_fround{#1}{#2}{#1}{#3}%
       \CALX_fa_bm}%
```

\CALX_faddu{P}{X}{Y}, \CALX_fsubu{P}{X}{Y}: X + Y, X - Y rounded to P digits, for X, Y finite with at most P digits: the input roundings of \CALX_fadd are skipped (they would leave X, Y unchanged), the rest is the same code (\CALX_fau builds the input of \CALX_fa_bp, \CALX_fa_bm as \CALX_fa_a does). The same holds for \CALX_fmulu, \CALX_fsqru and \CALX_fdivu below. Used where the operands are known to be short (inverse trigonometric functions).

```
\def\CALX_faddu #1#2#3%
   {\expandafter\CALX_fau\romannumeral`&&@#2\xint:{#1}{#3}\CALX_fa_bp}%
\def\CALX_fsubu #1#2#3%
   {\expandafter\CALX_fau\romannumeral`&&@#2\xint:{#1}{#3}\CALX_fa_bm}%
\def\CALX_fau F#1\xint:#2#3#4%
   {\expandafter#4\romannumeral`&&@#3\xint:{#2}#11;!1;!1;!1;!\W\xint:\xint:}%
```

\CALX_fhalf{X}: X/2, exact (X finite). With the blocks b_k of X (b_k = 0 for k < 0 or k >= n) and exponent e, X/2 has exponent e - 1 and the blocks

$$c_j = 5\ 10^7\ (b_j \bmod 2) + \operatorname{floor}(b_{(j-1)}/2), \quad j = 0, \ldots, n,$$

since 10^8 b_j / 2 = 5 10^7 (b_j mod 2) + 10^8 floor(b_j/2): no carry (c_j < 10^8). One \expanded pass over the list (\CALX_fhv_b, floor(b/2) = round((2b-1)/4) in \numexpr); c_n is dropped if 0. The digit count is D + 8, minus 1 if floor(b_(n-1)/2) < 10^(d-1), d the number of digits of b_(n-1) (that is, the leading digit of X is 1).

```
\def\CALX_fhalf #1{\expandafter\CALX_fhv_a\romannumeral`&&@#1\xint:}%
\def\CALX_fhv_a F#1#2.#3.#4\xint:%
{%
    \ifnum#2=\xint_c_ \xint_dothis{F#1#2.#3.}\fi
    \xint_orthat{\expandafter\CALX_fhv_e\expanded{\CALX_fhv_b0.#41;!}%
      #1#2.#3.}%
}%
```

```
\def\CALX_fhv_b #1.1#2#3!%
  {\xint_gob_til_sc#2\CALX_fhv_t;\CALX_fhv_c#1.#2#3.}%
\def\CALX_fhv_c #1.#2.%
  {\expandafter\CALX_fhv_d\the\numexpr(\xint_c_ii*#2-\xint_c_i)/\xint_c_iv.%
   #2.#1.}%
\def\CALX_fhv_d #1.#2.#3.%
  {\the\numexpr\xint_c_e_viii+50000000*(#2-\xint_c_ii*#1)+#3!%
   \CALX_fhv_b#1.}%
\def\CALX_fhv_t;\CALX_fhv_c#1.;.%
  {\ifnum#1>\xint_c_ \the\numexpr\xint_c_e_viii+#1!\fi\xint:#1.}%
\def\CALX_fhv_e #1\xint:#2.#3#4.#5.%
{%
    \expandafter\CALX_fhv_f
    \the\numexpr#4+\ifnum#2<\ifcase\numexpr#4-\xint_c_i-\xint_c_viii*%
      ((\xint_c_ii*#4-\xint_c_ix)/16)\relax
      1\or 10\or 100\or 1000\or 10000\or 100000\or 1000000\or 10000000\fi
      \space\xint_c_vii\else\xint_c_viii\fi
    \expandafter.\the\numexpr#5-\xint_c_i.#3#1%
}%
\def\CALX_fhv_f #1.#2.#3{F#3#1.#2.}%
\def\CALX_fa_a #1#2#3\xint:#4#5#6%
{%
    \if I#1\CALX_fa_spx#2{#4}{#5}#6\fi
    \expandafter#6\romannumeral0\CALX_fround{#4}{#5}{#4}#2#3%
    1;!1;!1;!1;!\W\xint:\xint:%
}%
\def\CALX_fa_spx #1#2#3#4\fi#5\W\xint:\xint:%
{%
    \fi\expandafter\CALX_fa_spz\expandafter#1\expandafter#4%
    \romannumeral0\CALX_fround{#2}{#3}%
}%
\def\CALX_fa_spz #1#2#3#4#5\xint:%
{%
    \ifx#2\CALX_fa_bm\expandafter\CALX_fa_spf\else\expandafter\CALX_fa_sp\fi
    #3#4I#1%
}%
\def\CALX_fa_spf #1#2{\expandafter\CALX_fa_sp\expandafter#1%
    \romannumeral`&&@\CALX_fa_f#2}%
\def\CALX_fa_f #1{\csname CALX_fa_f#1\endcsname}%
\expandafter\def\csname CALX_fa_f+\endcsname{-}%
\expandafter\def\csname CALX_fa_f-\endcsname{+}%
\expandafter\def\csname CALX_fa_f?\endcsname{?}%
\def\CALX_fa_sp #1#2#3#4%
{%
    \if ?#2\xint_dothis{ I?}\fi
    \if ?#4\xint_dothis{ I?}\fi
    \if F#1\xint_dothis{ I#4}\fi
    \if F#3\xint_dothis{ I#2}\fi
    \if #2#4\xint_dothis{ I#2}\fi
    \xint_orthat{\CALX_nsig{InvalidOperation}{inf - inf}{}}%
}%
\def\CALX_fa_bp #1#2{\if I#1\CALX_fa_spy#2\fi\CALX_fa_c#2}%
```

```
\def\CALX_fa_bm #1#2%
{%
    \if I#1\CALX_fa_spm#2\fi
    \if-#2\expandafter\CALX_fa_c\expandafter+\else
          \expandafter\CALX_fa_c\expandafter-\fi
}%
\def\CALX_fa_spy #1\fi#2\W\xint:\xint:{\fi\CALX_sp I#1}%
\def\CALX_fa_spm #1\fi#2\W\xint:\xint:%
   {\fi\expandafter\CALX_sp\expandafter I\romannumeral`&&@\CALX_fa_f#1}%
\def\CALX_fa_c #1#2.#3.#4\xint:#5#6#7.#8.%
{%
    \ifnum#2=\xint_c_ \CALX_fa_rx#6#7.#8.\fi
    \ifnum#7=\xint_c_ \CALX_fa_ry#1#2.#3.#4\fi
    \ifnum\numexpr\xint_c_viii*(#3-#8)+#2-#7>\numexpr#5+\xint_c_i\relax
      \CALX_fa_ry#1#2.#3.#4\fi
    \ifnum\numexpr\xint_c_viii*(#8-#3)+#7-#2>\numexpr#5+\xint_c_i\relax
      \CALX_fa_rx#6#7.#8.\fi
    \expandafter\CALX_fa_h\the\numexpr\ifnum#3<#8 #3\else#8\fi
    \expandafter.\the\numexpr#5\expandafter.\romannumeral0\expandafter
    \CALX_nadd_y\expanded{N#1\the\numexpr(#2+\xint_c_iii)/\xint_c_viii
      \ifnum#3>#8 +#3-#8.\CALX_fa_z#3-#8.\else.\fi\unexpanded{#4}\xint:
      N#6\the\numexpr(#7+\xint_c_iii)/\xint_c_viii
      \ifnum#8>#3 +#8-#3.\CALX_fa_z#8-#3.\else.\fi}%
}%
\def\CALX_fa_rx #1\fi#2\CALX_nadd_y\expanded#3#41;!1;!1;!1;!\W\xint:\xint:%
   {\fi\CALX_sp F#1#4}%
\def\CALX_fa_ry #1\fi#2\W\xint:\xint:{\fi\CALX_sp F#1}%
\def\CALX_fa_z #1.%
{%
    \ifnum\numexpr#1>\xint_c_ 100000000!%
      \expandafter\CALX_fa_z\the\numexpr#1-\xint_c_i\expandafter.%
    \fi
}%
\def\CALX_fa_h #1.#2.N#3#4.%
{%
    \ifnum#4=\xint_c_ \expandafter\CALX_fa_zero\else
                      \expandafter\CALX_fa_g\fi N#3#4.{#1}{#2}%
}%
\def\CALX_fa_zero N#1#2.#3#4\xint:{ F+0.0.}%
\def\CALX_fa_g N#1#2.#3#4#5\xint:%
{%
    \expandafter\CALX_fm_gd\the\numexpr#3\expandafter.\the\numexpr
    \xint_c_viii*(#2-\xint_c_i)+\CALX_nskip\CALX_fm_gl #2-\xint_c_i.#5\xint:.%
    #1{#4}#5%
}%
```

31.7. Multiplication

31.7.1. \CALX_fmul

```
Syntax (use with \romannumeral0; P a \numexpr expression; X, Y f-expanded F, I or raw values):
  \CALX_fmul{P}{X}{Y} -> <space>Z Z = round_P(round_P(X) round_P(Y))
    \CALX_fmulu{P}{X}{Y}  -> <space>Z    (X, Y finite F, at most P digits)
```

```
    \CALX_fmulx{X}{Y}     -> <space>Z    exact product (X, Y finite F)
    \CALX_faddx{X}{Y}     -> <space>Z    exact sum (X, Y finite F)
    \CALX_fsubx{X}{Y}     -> <space>Z    exact difference
```

Z: F or I value.

\CALX_fmul{P}{X}{Y} (use with \romannumeral0, X and Y f-expanded): round_P(round_P(X) round_P(Y)), round half away from zero, as xint's \XINTinFloatMul[P] (xint 1.2k and later).

Special values first (\CALX_fm_sp: I? propagates, inf times zero signals InvalidOperation, otherwise I+ or I- by the product of the signs), then zero.

P <= 16 and both mantissas with at most 24 digits: each operand is replaced by its top P significant digits A, rounded, as two blocks (A = Ah B + Al), and a decimal exponent: X = A 10^(8e + z) (\CALX_fm_xvi for P = 16, \CALX_fm_ex otherwise). The product of the two A's is computed by \CALX_fm_k (32 digits), then rounded to P digits and put back in the native float format by \CALX_fm_r. Otherwise: \CALX_fm_big (P <= 20: one product of 20 digit operands, 20 < P <= 34: three such products, P > 34: generic).

Small path: \CALX_fm_nb grabs X once (X special: jumper \CALX_fm_nsx) and writes it back after Y, padded with two zero blocks and closed by \iffalse{\fi} (the closing brace of the \expanded opened by \CALX_fm_nc), then {P}. \CALX_fm_ntag reads the tag of Y (special: jumper \CALX_fm_nsy). \CALX_fm_nc grabs Y's list and X's header; X's list stays in place. Jumpers: a zero operand (\CALX_fm_nz), more than 24 digits (\CALX_fm_nbig, to \CALX_fm_big). Common case: one \expanded runs the two extractions (\CALX_fm_ny, \CALX_fm_nx: \CALX_fm_xvi or \CALX_fm_ex by \csname), the second one on X's list in place. \CALX_fm_ne regroups 1Bl!1Bh!z'.{s}{8(e+e')}1Al!1Ah!z. into the input of \CALX_fm_k, followed by {s}{8(e+e') + z + z'}{P}.

An operand which is neither F nor I is an xint raw value: \CALX_fm_nk (X, jumper \CALX_fm_nj) and \CALX_fm_ntk (Y, jumper \CALX_fm_nry) send both operands to \CALX_fm_big, whose \CALX_fround converts it. The same for \CALX_fsqr (\CALX_fs_nc, \CALX_fs_ns, \CALX_fs_nr).

```
\def\CALX_fmul #1%
{%
    \ifnum#1>16 \expandafter\CALX_fm_big\else\expandafter\CALX_fm_na\fi %
        {#1}%
}%
\def\CALX_fmulu #1%
{%
    \ifnum#1>16 \expandafter\CALX_fmu_b\else\expandafter\CALX_fm_na\fi
        {#1}%
}%
\def\CALX_fmu_b #1#2{\expandafter\CALX_fmu\romannumeral`&&@#2\xint:{#1}}%
\def\CALX_fmu F#1\xint:#2#3%
   {\expandafter\CALX_fm_btag\romannumeral`&&@#3\xint:{#2}#1\xint:}%
\def\CALX_fm_na #1#2{\expandafter\CALX_fm_nk\romannumeral`&&@#2%
    \xint:{#1}}%
\def\CALX_fm_nk #1%
   {\if F#1\else\if I#1\else\CALX_fm_nj\fi\fi\CALX_fm_nb #1}%
\def\CALX_fm_nj #1\fi\fi\CALX_fm_nb #2#3\xint:#4%
   {\fi\fi\CALX_fm_big{#4}{#2#3}}%
\def\CALX_fm_nb #1#2#3\xint:#4#5%
{%
    \if I#1\CALX_fm_nsx#1#2{#5}\fi
    \expandafter\CALX_fm_ntk\romannumeral`&&@#5\xint:{#4}#2#3%
    100000000!100000000!\xint:\iffalse{\fi}{#4}%
}%
\def\CALX_fm_nsx #1#2#3\fi#4\iffalse#5#6%
   {\fi\expandafter\CALX_fm_sp\romannumeral`&&@#3\xint:#1#2\xint:}%
```

```
\def\CALX_fm_ntk #1%
   {\if F#1\else\if I#1\else\CALX_fm_nry\fi\fi\CALX_fm_ntag #1}%
\def\CALX_fm_nry #1\fi\fi\CALX_fm_ntag #2#3\xint:#4#5%
    100000000!100000000!\xint:\iffalse#6#7%
   {\fi\fi\CALX_fm_big{#4}{F#5}{#2#3}}%
\def\CALX_fm_ntag #1#2{\if I#1\CALX_fm_nsy#2\fi\CALX_fm_nc#2}%
\def\CALX_fm_nsy #1\fi#2\xint:#3#4100000000!100000000!\xint:\iffalse#5#6%
   {\fi\CALX_fm_sp I#1\xint:F#4\xint:}%
\def\CALX_fm_nc #1#2.#3.#4\xint:#5#6#7.#8.%
{%
    \ifnum#2=\xint_c_ \CALX_fm_nz\fi
    \ifnum#7=\xint_c_ \CALX_fm_nz\fi
    \ifnum#2>24 \CALX_fm_nbig{#5}{F#1#2.#3.#4}{F#6#7.#8.}\fi
    \ifnum#7>24 \CALX_fm_nbig{#5}{F#1#2.#3.#4}{F#6#7.#8.}\fi
    \expandafter\CALX_fm_ne\expanded{\iffalse}\fi
      \CALX_fm_ny{#5}{#2}#4100000000!100000000!\xint:%
      {\if#1#6+\else-\fi}{\the\numexpr\xint_c_viii*(#3+#8)}%
      \CALX_fm_nx{#5}{#7}%
}%
\def\CALX_fm_nz\fi#1\iffalse#2#3{\fi\CALX_sp F+0.0.}%
\def\CALX_fm_nbig #1#2#3\fi#4\CALX_fm_nx#5#6#7100000000!100000000!\xint:%
    \iffalse#8#9{\fi\CALX_fm_big{#1}{#3#7}{#2}}%
\def\CALX_fm_nx #1%
   {\csname CALX_fm_\ifnum#1=16 nxvi\else ex\fi\endcsname{#1}}%
\let\CALX_fm_ny\CALX_fm_nx
\def\CALX_fm_nxvi #1{\CALX_fm_xvi}%
\def\CALX_fm_ne #1!#2!#3.#4#5#6!#7!#8.%
   {\CALX_fm_k #6!#7!#1!#2!{#4}{#5+#8+#3}}%
\def\CALX_fm_sp #1#2#3\xint:#4#5#6\xint:%
{%
    \if ?#2\xint_dothis{ I?}\fi
    \if ?#5\xint_dothis{ I?}\fi
    \ifnum\if F#1\CALX_nsp_z#3\xint:\else\if F#4\CALX_nsp_z#6\xint:\else
      \xint_c_i\fi\fi=\xint_c_
      \xint_dothis{\CALX_nsig{InvalidOperation}{inf * 0}{}}\fi
    \xint_orthat{\CALX_nsp_ps#2#5}%
}%
```

31.7.2. \CALX_fm_xvi

\CALX_fm_xvi{D}<blocks>\xint: (P = 16, D <= 24, the blocks padded with two zero blocks): 1Al!1Ah!z. with A = Ah B + Al of exactly 16 digits.
  - D = 16: A is the mantissa, z = 0.
  - D < 16: A = M 10^(16-D), z = D - 16: \CALX_fm_la<s> (s = 16 - D, roman numeral) emits Al and calls \CALX_fm_lb<s> for Ah; digits are only moved, as all blocks have exactly eight digits.
  - 16 < D < 24: A = round(M/10^z), z = D - 16: \CALX_fm_ua<z>, \CALX_fm_ub<z>, \CALX_fm_uc cut the digits; the digit just below A decides the rounding (half away from zero: no tie rule needed).
  - D = 24: A is the two top blocks, rounded with the top digit of the low block (\CALX_fm_xvie).
  \CALX_fm_xvif{Al'}{Ah}{z}, Al' <= 10^8 (an expression): carry to Ah; if A becomes 10^16, A = 10^15 and z + 1.

```
\def\CALX_fm_xvi #11#2!1#3!1#4!#5\xint:%
{%
    \ifnum#1<16
```

```
      \csname CALX_fm_la\the\numexpr16-#1\endcsname #2!#3!\the%
          \numexpr#1-16.%
    \else\ifnum#1=16 1#2!1#3!0.%
    \else\ifnum#1=24 \CALX_fm_xvie{#3}{#4}{#2}%
    \else
      \csname CALX_fm_ua\the\numexpr#1-16\endcsname #2!#3!#4!%
    \fi\fi\fi
}%
\def\CALX_fm_uc #1#2#3#4#5!%
{%
    \CALX_fm_xvif{#1+\ifnum#3>\xint_c_iv\xint_c_i\else\xint_c_\fi}%
                 {#5*\CALX_pten{\xint_c_viii-#4}+#2}{#4}%
}%
\expandafter\def\csname CALX_fm_ua1\endcsname
   #1#2#3#4#5#6#7#8!{\CALX_fm_ubi{{#1#2#3#4#5#6#7}{#8}}}%
\expandafter\def\csname CALX_fm_ua2\endcsname
   #1#2#3#4#5#6#7#8!{\CALX_fm_ubii{{#1#2#3#4#5#6}{#7}}}%
\expandafter\def\csname CALX_fm_ua3\endcsname
   #1#2#3#4#5#6#7#8!{\CALX_fm_ubiii{{#1#2#3#4#5}{#6}}}%
\expandafter\def\csname CALX_fm_ua4\endcsname
   #1#2#3#4#5#6#7#8!{\CALX_fm_ubiv{{#1#2#3#4}{#5}}}%
\expandafter\def\csname CALX_fm_ua5\endcsname
   #1#2#3#4#5#6#7#8!{\CALX_fm_ubv{{#1#2#3}{#4}}}%
\expandafter\def\csname CALX_fm_ua6\endcsname
   #1#2#3#4#5#6#7#8!{\CALX_fm_ubvi{{#1#2}{#3}}}%
\expandafter\def\csname CALX_fm_ua7\endcsname
   #1#2#3#4#5#6#7#8!{\CALX_fm_ubvii{{#1}{#2}}}%
\def\CALX_fm_ubi #1#2#3#4#5#6#7#8#9!{\CALX_fm_uc{#9%
    \xint_firstoftwo#1}{#2#3#4#5#6#7#8}{\xint_secondoftwo#1}{1}}%
\def\CALX_fm_ubii #1#2#3#4#5#6#7#8#9!{\CALX_fm_uc{#8#9%
    \xint_firstoftwo#1}{#2#3#4#5#6#7}{\xint_secondoftwo#1}{2}}%
\def\CALX_fm_ubiii #1#2#3#4#5#6#7#8#9!{\CALX_fm_uc{#7#8#9%
    \xint_firstoftwo#1}{#2#3#4#5#6}{\xint_secondoftwo#1}{3}}%
\def\CALX_fm_ubiv #1#2#3#4#5#6#7#8#9!{\CALX_fm_uc{#6#7#8#9%
    \xint_firstoftwo#1}{#2#3#4#5}{\xint_secondoftwo#1}{4}}%
\def\CALX_fm_ubv #1#2#3#4#5#6#7#8#9!{\CALX_fm_uc{#5#6#7#8#9%
    \xint_firstoftwo#1}{#2#3#4}{\xint_secondoftwo#1}{5}}%
\def\CALX_fm_ubvi #1#2#3#4#5#6#7#8#9!{\CALX_fm_uc{#4#5#6#7#8#9%
    \xint_firstoftwo#1}{#2#3}{\xint_secondoftwo#1}{6}}%
\def\CALX_fm_ubvii #1#2#3#4#5#6#7#8#9!{\CALX_fm_uc{#3#4#5#6#7#8#9%
    \xint_firstoftwo#1}{#2}{\xint_secondoftwo#1}{7}}%
\expandafter\def\csname CALX_fm_la1\endcsname
   #1#2#3#4#5#6#7#8!{1#2#3#4#5#6#7#80!\CALX_fm_lbi{#1}}%
\def\CALX_fm_lbi #1#2#3#4#5#6#7#8#9!{1#3#4#5#6#7#8#9#1!}%
\expandafter\def\csname CALX_fm_la2\endcsname
   #1#2#3#4#5#6#7#8!{1#3#4#5#6#7#800!\CALX_fm_lbii{#1#2}}%
\def\CALX_fm_lbii #1#2#3#4#5#6#7#8#9!{1#4#5#6#7#8#9#1!}%
\expandafter\def\csname CALX_fm_la3\endcsname
   #1#2#3#4#5#6#7#8!{1#4#5#6#7#8000!\CALX_fm_lbiii{#1#2#3}}%
\def\CALX_fm_lbiii #1#2#3#4#5#6#7#8#9!{1#5#6#7#8#9#1!}%
\expandafter\def\csname CALX_fm_la4\endcsname
   #1#2#3#4#5#6#7#8!{1#5#6#7#80000!\CALX_fm_lbiv{#1#2#3#4}}%
```

```
\def\CALX_fm_lbiv #1#2#3#4#5#6#7#8#9!{1#6#7#8#9#1!}%
\expandafter\def\csname CALX_fm_la5\endcsname
   #1#2#3#4#5#6#7#8!{1#6#7#800000!\CALX_fm_lbv{#1#2#3#4#5}}%
\def\CALX_fm_lbv #1#2#3#4#5#6#7#8#9!{1#7#8#9#1!}%
\expandafter\def\csname CALX_fm_la6\endcsname
   #1#2#3#4#5#6#7#8!{1#7#8000000!\CALX_fm_lbvi{#1#2#3#4#5#6}}%
\def\CALX_fm_lbvi #1#2#3#4#5#6#7#8#9!{1#8#9#1!}%
\expandafter\def\csname CALX_fm_la7\endcsname #1#2#3#4#5#6#7#8!{1#80000000!%
    \CALX_fm_lbvii{#1#2#3#4#5#6#7}}%
\def\CALX_fm_lbvii #1#2#3#4#5#6#7#8#9!{1#9#1!}%
\expandafter\def\csname CALX_fm_la8\endcsname #1#2#3#4#5#6#7#8!{100000000!%
    \CALX_fm_lbviii{#1#2#3#4#5#6#7#8}}%
\def\CALX_fm_lbviii #1#2#3#4#5#6#7#8#9!{1#1!}%
\expandafter\def\csname CALX_fm_la9\endcsname
   #1#2#3#4#5#6#7#8!#9!{100000000!1#2#3#4#5#6#7#80!}%
\expandafter\def\csname CALX_fm_la10\endcsname
   #1#2#3#4#5#6#7#8!#9!{100000000!1#3#4#5#6#7#800!}%
\expandafter\def\csname CALX_fm_la11\endcsname
   #1#2#3#4#5#6#7#8!#9!{100000000!1#4#5#6#7#8000!}%
\expandafter\def\csname CALX_fm_la12\endcsname
   #1#2#3#4#5#6#7#8!#9!{100000000!1#5#6#7#80000!}%
\expandafter\def\csname CALX_fm_la13\endcsname
   #1#2#3#4#5#6#7#8!#9!{100000000!1#6#7#800000!}%
\expandafter\def\csname CALX_fm_la14\endcsname
   #1#2#3#4#5#6#7#8!#9!{100000000!1#7#8000000!}%
\expandafter\def\csname CALX_fm_la15\endcsname
   #1#2#3#4#5#6#7#8!#9!{100000000!1#80000000!}%
\def\CALX_fm_xvie #1#2#3%
   {\CALX_fm_xvif{#1+\ifnum#3>49999999 \xint_c_i\else\xint_c_\fi}{#2}{%
       \xint_c_viii}}%
\def\CALX_fm_xvif #1#2#3%
{%
    \ifnum\numexpr#1<\xint_c_e_viii
      \the\numexpr#1+\xint_c_e_viii!\the\numexpr#2+\xint_c_e_viii!\the%
          \numexpr#3.%
    \else
      \ifnum\numexpr#2=99999999
        100000000!110000000!\the\numexpr#3+\xint_c_i.%
      \else
        100000000!\the\numexpr#2+\xint_c_i+\xint_c_e_viii!\the\numexpr#3.%
      \fi
    \fi
}%
```

31.7.3. \CALX_fm_ex

P < 16: \CALX_fm_ex{P}{D}<blocks>\xint: gives 1Al!1Ah!z. with A the mantissa if D <= P (z = 0), else A = round(M/10^z), z = D - P, by \CALX_fm_re.

\CALX_fm_re{c}{B3}{B2}{B1}{B0}{P}, 1 <= c <= 31: Q = B3 ... B0 (four blocks) rounded to round(Q/10^c) < 10^P (at most 16 digits), with the same output. With c = 8a + b (\CALX_fm_ref selects the three blocks X2, X1, X0 from index a and the block Xm below them) and p = 10^b: Al = floor(X0/p) + (X1 mod p) 10^8/p, Ah = floor(X1/p) + (X2 mod p) 10^8/p, plus one if the first dropped digit is >= 5 (b > 0: 2 (X0 mod p) >= p; b = 0: Xm >= 5 10^7). \CALX_fm_rec, \CALX_fm_red: carry,

and 10^P becomes 10^(P-1) with c + 1.
  \CALX_fm_fd{x}{p} = floor(x/p) = (2x - p + 1)/(2p) in \numexpr (no tie, 0 <= x < 2^30). \CALX_pten{k} = 10^k, 0 <= k <= 8. \CALX_ndig{v} (calxcore): number of digits of 0 < v < 10^8.

```
\def\CALX_fm_ex #1#2#3!1#4!1#5!#6\xint:%
{%
    \ifnum#1<#2 % P may be the token \XINTdigits: no space may follow it
      \CALX_fm_re{#2-#1}{0}{#5}{#4}{\xint_gobble_i#3}{#1}%
    \else
      #3!1#4!0.%
    \fi
}%
\def\CALX_fm_fd #1#2{((\xint_c_ii*(#1)-(#2)+\xint_c_i)/(\xint_c_ii*(#2)))}%
\def\CALX_fm_re #1#2#3#4#5#6%
{%
    \expandafter\CALX_fm_rea\the\numexpr(\xint_c_ii*(#1)-\xint_c_vii)/16.%
    #1.#2.#3.#4.#5.#6.%
}%
\def\CALX_fm_rea #1.#2.%
   {\expandafter\CALX_fm_ref\the\numexpr\CALX_pten{#2-%
       \xint_c_viii*#1}.#1.#2.}%
\def\CALX_fm_ref #1.#2.#3.#4.#5.#6.#7.#8.%
{%
    \ifcase#2
         \xint_dothis{\CALX_fm_reb #1.#3.#5.#6.#7.0.#8.}%
    \or  \xint_dothis{\CALX_fm_reb #1.#3.#4.#5.#6.#7.#8.}%
    \or  \xint_dothis{\CALX_fm_reb #1.#3.0.#4.#5.#6.#8.}%
    \fi  \xint_orthat{\CALX_fm_reb #1.#3.0.0.#4.#5.#8.}%
}%
\def\CALX_fm_reb #1.#2.#3.#4.#5.#6.#7.%
{%
    \expandafter\CALX_fm_rec\the\numexpr
      \CALX_fm_fd{#5}{#1}+(#4-\CALX_fm_fd{#4}{#1}*#1)*(\xint_c_e_viii/#1)%
      \ifnum#1=\xint_c_i \ifnum#6>49999999 +\xint_c_i\fi
      \else\ifnum\numexpr\xint_c_ii*(#5-\CALX_fm_fd{#5}{#1}*#1)<#1
      \else +\xint_c_i\fi\fi\expandafter.%
    \the\numexpr\CALX_fm_fd{#4}{#1}+(#3-\CALX_fm_fd{#3}{#1}*#1)*(%
        \xint_c_e_viii/#1).#2.#7.%
}%
\def\CALX_fm_rec #1.#2.%
{%
    \ifnum#1=\xint_c_e_viii
      \xint_dothis{\expandafter\CALX_fm_red\the\numexpr#2+\xint_c_i.0.}\fi
    \xint_orthat{\CALX_fm_red #2.#1.}%
}%
\def\CALX_fm_red #1.#2.#3.#4.%
{%
    \ifnum\ifnum#4>\xint_c_viii #1<\CALX_pten{#4-\xint_c_viii}\else#2<%
        \CALX_pten{#4}\fi
      \space \expandafter\xint_secondoftwo\else\expandafter\xint_firstoftwo\fi
    {\ifnum#4>\xint_c_viii 100000000!\the\numexpr\CALX_pten{#4-9}+%
        \xint_c_e_viii!%
     \else\the\numexpr\CALX_pten{#4-\xint_c_i}+\xint_c_e_viii!100000000!\fi
```

```
      \the\numexpr#3+\xint_c_i.}%
    {\the\numexpr#2+\xint_c_e_viii!\the\numexpr#1+\xint_c_e_viii!#3.}%
}%
```

31.7.4. \CALX_fm_k

\CALX_fm_k 1Al!1Ah!1Bl!1Bh!{s}{E}{P}: each block is split into two groups of four digits (\CALX_fm_k ... \CALX_fm_ke, digits only moved), the seven column sums are computed in \numexpr from the lowest one (\CALX_fm_kf), each with 10^9 added, so that the carry is the first five digits after the leading 1 and the group of four digits of the product is the last four (\CALX_fm_s). \CALX_fm_o, \CALX_fm_p regroup the eight groups into the four blocks of the product, top first: \CALX_fm_r{B3}{B2}{B1}{B0}{s}{E}{P}.

```
\def\CALX_fm_k 1#1#2#3#4#5!{\CALX_fm_kb{#5}{#1#2#3#4}}%
\def\CALX_fm_kb #1#21#3#4#5#6#7!{\CALX_fm_kc{#1}{#2}{#7}{#3#4#5#6}}%
\def\CALX_fm_kc #1#2#3#41#5#6#7#8#9!{%
    \CALX_fm_kd{#1}{#2}{#3}{#4}{#9}{#5#6#7#8}}%
\def\CALX_fm_kd #1#2#3#4#5#61#7#8#9%
   {\CALX_fm_ke{#1}{#2}{#3}{#4}{#5}{#6}{#7#8#9}}%
\def\CALX_fm_ke #1#2#3#4#5#6#7#8#9!{%
    \CALX_fm_kf{#1}{#2}{#3}{#4}{#5}{#6}{#9}{#7#8}}%
\def\CALX_fm_kf #1#2#3#4#5#6#7#8%
{%
    \expandafter\CALX_fm_o\the\numexpr1000000000+#4*#8+%
    \expandafter\CALX_fm_s\the\numexpr1000000000+#3*#8+#4*#7+%
    \expandafter\CALX_fm_s\the\numexpr1000000000+#2*#8+#3*#7+#4*#6+%
    \expandafter\CALX_fm_s\the\numexpr1000000000+#1*#8+#2*#7+#3*#6+#4*#5+%
    \expandafter\CALX_fm_s\the\numexpr1000000000+#1*#7+#2*#6+#3*#5+%
    \expandafter\CALX_fm_s\the\numexpr1000000000+#1*#6+#2*#5+%
    \expandafter\CALX_fm_s\the\numexpr1000000000+#1*#5;%
}%
\def\CALX_fm_s 1#1#2#3#4#5{#1#2#3#4#5;}%
\def\CALX_fm_o 1#1#2#3#4#5#6#7#8#9;{\CALX_fm_p{#2#3#4#5#6#7#8#9}}%
\def\CALX_fm_p #1#2;#3;#4;#5;#6;#7;{\CALX_fm_r{#1}{#2#3}{#4#5}{#6#7}}%
```

31.7.5. \CALX_fm_r

Rounding of the product Q to P digits, and output.

P = 16 (\CALX_fm_rxvi): both A's have 16 digits, so Q has 31 or 32 digits. 32 digits (B3 >= 10^7): R = B3 B + B2 rounded with the top digit of B1, c = 16; 31 digits: R = floor(Q/10^15), cut by \CALX_fm_rft (digits only moved), c = 15. Then \CALX_fm_xvif (carry, 10^16). R has 16 digits; with E + c = 8e + r (0 <= r < 8), the result is F<s><16+r>.<e>. followed by the blocks of R 10^r: \CALX_fm_oa<r> emits the low block and calls \CALX_fm_ob<r> for the two others (\CALX_fm_oa for r = 0: R itself).

P < 16 (\CALX_fm_rg, \CALX_fm_rr): the number of digits of Q gives c = D(Q) - P; if c > 0, \CALX_fm_re. \CALX_fm_no ... \CALX_fm_nt: the output as above, with D(R) computed and R 10^r by \CALX_fm_fd.

floor(E/8) = (2E - 7)/16 in \numexpr (never a tie).

```
\expandafter\def\csname CALX_fm_oa0\endcsname #1!#2!{1#1!1#2!}%
\expandafter\def\csname CALX_fm_oa1\endcsname
   #1#2#3#4#5#6#7#8!{1#2#3#4#5#6#7#80!\CALX_fm_obi{#1}}%
\def\CALX_fm_obi #1#2#3#4#5#6#7#8#9!{1#3#4#5#6#7#8#9#1!10000000#2!}%
\expandafter\def\csname CALX_fm_oa2\endcsname
   #1#2#3#4#5#6#7#8!{1#3#4#5#6#7#800!\CALX_fm_obii{#1#2}}%
```

```
\def\CALX_fm_obii #1#2#3#4#5#6#7#8#9!{1#4#5#6#7#8#9#1!1000000#2#3!}%
\expandafter\def\csname CALX_fm_oa3\endcsname
   #1#2#3#4#5#6#7#8!{1#4#5#6#7#8000!\CALX_fm_obiii{#1#2#3}}%
\def\CALX_fm_obiii #1#2#3#4#5#6#7#8#9!{1#5#6#7#8#9#1!100000#2#3#4!}%
\expandafter\def\csname CALX_fm_oa4\endcsname
   #1#2#3#4#5#6#7#8!{1#5#6#7#80000!\CALX_fm_obiv{#1#2#3#4}}%
\def\CALX_fm_obiv #1#2#3#4#5#6#7#8#9!{1#6#7#8#9#1!10000#2#3#4#5!}%
\expandafter\def\csname CALX_fm_oa5\endcsname
   #1#2#3#4#5#6#7#8!{1#6#7#800000!\CALX_fm_obv{#1#2#3#4#5}}%
\def\CALX_fm_obv #1#2#3#4#5#6#7#8#9!{1#7#8#9#1!1000#2#3#4#5#6!}%
\expandafter\def\csname CALX_fm_oa6\endcsname
   #1#2#3#4#5#6#7#8!{1#7#8000000!\CALX_fm_obvi{#1#2#3#4#5#6}}%
\def\CALX_fm_obvi #1#2#3#4#5#6#7#8#9!{1#8#9#1!100#2#3#4#5#6#7!}%
\expandafter\def\csname CALX_fm_oa7\endcsname #1#2#3#4#5#6#7#8!{1#80000000!%
    \CALX_fm_obvii{#1#2#3#4#5#6#7}}%
\def\CALX_fm_obvii #1#2#3#4#5#6#7#8#9!{1#9#1!10#2#3#4#5#6#7#8!}%
\def\CALX_fm_r #1#2#3#4#5#6#7%
{%
    \ifnum#7=16 \expandafter\CALX_fm_rxvi\else\expandafter\CALX_fm_rg\fi
    {#1}{#2}{#3}{#4}{#5}{#6}{#7}%
}%
\def\CALX_fm_rxvi #1#2#3#4#5#6#7%
{%
    \expandafter\CALX_fm_ox\expanded{%
    \ifnum#1>9999999
      \CALX_fm_xvif{#2+\ifnum#3>49999999 \xint_c_i\else\xint_c_\fi}{#1}{16}%
    \else
      \CALX_fm_rft #1!#2!#3!%
    \fi}{#5}{#6}%
}%
\def\CALX_fm_rft #1#2!#3#4!#5#6#7!%
   {\CALX_fm_xvif{#4#5+\ifnum#6>\xint_c_iv\xint_c_i\else\xint_c_%
       \fi}{#2#3}{15}}%
\def\CALX_fm_ox 1#1!1#2!#3.#4#5%
   {\expandafter\CALX_fm_oy\the\numexpr#5+#3.#1.#2.#4}%
\def\CALX_fm_oy #1.%
   {\expandafter\CALX_fm_oz\the\numexpr(\xint_c_ii*#1-\xint_c_vii)/16.#1.}%
\def\CALX_fm_oz #1.#2.%
   {\expandafter\CALX_fm_ow\the\numexpr#2-\xint_c_viii*#1.#1.}%
\def\CALX_fm_ow #1.#2.#3.#4.#5%
{%
    \expanded{ F#5\the\numexpr16+#1.#2.%
      \csname CALX_fm_oa#1\endcsname#3!#4!}%
}%
\def\CALX_fm_rg #1#2#3#4#5#6#7%
{%
    \expandafter\CALX_fm_rr\the\numexpr
      \ifnum#1>\xint_c_ 24+\CALX_ndig{#1}\else
      \ifnum#2>\xint_c_ 16+\CALX_ndig{#2}\else
      \ifnum#3>\xint_c_ 8+\CALX_ndig{#3}\else\CALX_ndig{#4}\fi\fi\fi-#7.%
    {#1}{#2}{#3}{#4}{#5}{#6}{#7}%
}%
```

```
\def\CALX_fm_rr #1.#2#3#4#5#6#7#8%
{%
    \ifnum#1>\xint_c_
      \expandafter\xint_firstoftwo\else\expandafter\xint_secondoftwo\fi
    {\expandafter\CALX_fm_no\expanded{\CALX_fm_re{#1}{#2}{#3}{#4}{#5}{#8}}}%
    {\CALX_fm_no 1#5!1#4!0.}{#6}{#7}%
}%
\def\CALX_fm_no 1#1!1#2!#3.#4#5%
{%
    \expandafter\CALX_fm_nq\the\numexpr#5+#3\expandafter.%
    \the\numexpr\ifnum#2>\xint_c_ 8+\CALX_ndig{#2}\else\CALX_ndig{#1}%
        \fi.#1.#2.#4%
}%
\def\CALX_fm_nq #1.%
   {\expandafter\CALX_fm_nr\the\numexpr(\xint_c_ii*#1-\xint_c_vii)/16.#1.}%
\def\CALX_fm_nr #1.#2.%
   {\expandafter\CALX_fm_ns\the\numexpr#2-\xint_c_viii*#1.#1.}%
\def\CALX_fm_ns #1.%
   {\expandafter\CALX_fm_nt\the\numexpr\CALX_pten{\xint_c_viii-#1}.#1.}%
\def\CALX_fm_nt #1.#2.#3.#4.#5.#6.#7%
{%
    \expanded{ F#7\the\numexpr#4+#2.#3.%
      \the\numexpr(#5-\CALX_fm_fd{#5}{#1}*#1)*\CALX_pten{#2}+%
          \xint_c_e_viii!%
      \ifnum\numexpr#4+#2>\xint_c_viii
        \the\numexpr\CALX_fm_fd{#5}{#1}+(#6-\CALX_fm_fd{#6}{#1}*#1)*%
            \CALX_pten{#2}%
           +\xint_c_e_viii!\fi
      \ifnum\numexpr#4+#2>16 \the\numexpr\CALX_fm_fd{#6}{#1}+\xint_c_e_viii!%
          \fi}%
}%
```

31.7.6. \CALX_fm_q

P <= 20 after \CALX_fround of both operands (P > 16, and also P <= 16 when a mantissa has more than 24 digits). Each mantissa M (D digits) is read as A = M 10^(20-D), exactly 20 digits (D <= 27 after rounding: the dropped digits are zeros), in groups of four digits, low group first: x0.x1.x2.x3.x4. (\CALX_fq_x: \CALX_fq_r<n> reverses the n blocks, \CALX_fq_s<z> drops the z leading zeros of the top block, \CALX_fq_ga, _gb, _gc take 20 digits, digits only moved). X Y = A A' 10^E with

E = 8(e + e') + D + D' - 40.

\CALX_fm_qa x0.x1.x2.x3.x4.y0.y1.y2.y3.y4. computes the 40 digits of A A' with one chain of nine \numexpr (one per column of the products of groups, 10^9 added, carry by \CALX_fm_s as in \CALX_fm_k). Its nine arguments are the five x and four y; it only moves tokens: the five columns using y4 are passed to \CALX_fm_qb ending with x<i>*, the four lower columns as a ready tail; \CALX_fm_qb (seven arguments) appends y4.

\CALX_fq_o keeps the top 23 or 24 digits S of A A' (P + 1 digits are enough for round half away from zero), exponent E + 16. \CALX_fq_p, _q, _t: with e = floor(E/8) and r = E - 8e, the mantissa S 10^r is cut in blocks (\CALX_fq_bl, after z leading zeros, z the complement of len(S) + r to a multiple of 8) and \CALX_fround{P} gives the result.

```
\def\CALX_fm_q #1#2#3#4#5#6#7#8%
{%
    \expandafter\CALX_fm_qa\expanded{\CALX_fq_x{#2}#4\xint:\CALX_fq_x{#6}#8%
```

```
        \xint:%
      {\if#1#5+\else-\fi}{\the\numexpr\xint_c_viii*(#3+#7)+#2+#6-40}}%
}%
\def\CALX_fq_x #1{\expandafter\CALX_fq_xa\the\numexpr(\xint_c_ii*#1+%
    \xint_c_vii)/16.#1.}%
\def\CALX_fq_xa #1.#2.%
   {\csname CALX_fq_r#1\endcsname{\the\numexpr%
       \xint_c_viii*#1-#2}}%
\expandafter\def\csname CALX_fq_r1\endcsname #11#2!#3\xint:%
   {\csname CALX_fq_s#1\endcsname #2000000000000000000\xint:}%
\expandafter\def\csname CALX_fq_r2\endcsname #11#2!1#3!#4\xint:%
   {\csname CALX_fq_s#1\endcsname #3#2000000000000000000%
       \xint:}%
\expandafter\def\csname CALX_fq_r3\endcsname #11#2!1#3!1#4!#5\xint:%
   {\csname CALX_fq_s#1\endcsname #4#3#2000000000000000000%
       \xint:}%
\expandafter\def\csname CALX_fq_r4\endcsname #11#2!1#3!1#4!1#5!#6\xint:%
   {\csname CALX_fq_s#1\endcsname #5#4#3#2000000000000000000%
       \xint:}%
\expandafter\def\csname CALX_fq_s0\endcsname {\CALX_fq_ga}%
\expandafter\def\csname CALX_fq_s1\endcsname #1{\CALX_fq_ga}%
\expandafter\def\csname CALX_fq_s2\endcsname #1#2{\CALX_fq_ga}%
\expandafter\def\csname CALX_fq_s3\endcsname #1#2#3{\CALX_fq_ga}%
\expandafter\def\csname CALX_fq_s4\endcsname #1#2#3#4{\CALX_fq_ga}%
\expandafter\def\csname CALX_fq_s5\endcsname #1#2#3#4#5{\CALX_fq_ga}%
\expandafter\def\csname CALX_fq_s6\endcsname #1#2#3#4#5#6{\CALX_fq_ga}%
\expandafter\def\csname CALX_fq_s7\endcsname #1#2#3#4#5#6#7{\CALX_fq_ga}%
\def\CALX_fq_ga #1#2#3#4#5#6#7#8{\CALX_fq_gb{#5#6#7#8.#1#2#3#4.}}%
\def\CALX_fq_gb #1#2#3#4#5#6#7#8#9{\CALX_fq_gc{#6#7#8#9.#2#3#4#5.#1}}%
\def\CALX_fq_gc #1#2#3#4#5#6\xint:{#2#3#4#5.#1}%
\def\CALX_fm_qa #1.#2.#3.#4.#5.#6.#7.#8.#9.%
{%
    \CALX_fm_qb{#5*}{#5*#9+#4*}{#4*#9+#5*#8+#3*}{#3*#9+#4*#8+#5*#7+#2*}%
                {#2*#9+#3*#8+#4*#7+#5*#6+#1*}%
    {\expandafter\CALX_fm_s\the\numexpr\xint_c_e_ix+#1*#9+#2*#8+#3*#7+#4*#6+%
     \expandafter\CALX_fm_s\the\numexpr\xint_c_e_ix+#1*#8+#2*#7+#3*#6+%
     \expandafter\CALX_fm_s\the\numexpr\xint_c_e_ix+#1*#7+#2*#6+%
     \expandafter\CALX_fm_s\the\numexpr\xint_c_e_ix+#1*#6;}%
}%
\def\CALX_fm_qb #1#2#3#4#5#6#7.%
{%
    \expandafter\CALX_fm_qo\the\numexpr\xint_c_e_ix+#1#7+%
    \expandafter\CALX_fm_s\the\numexpr\xint_c_e_ix+#2#7+%
    \expandafter\CALX_fm_s\the\numexpr\xint_c_e_ix+#3#7+%
    \expandafter\CALX_fm_s\the\numexpr\xint_c_e_ix+#4#7+%
    \expandafter\CALX_fm_s\the\numexpr\xint_c_e_ix+#5#7+#6%
}%
\def\CALX_fm_qo 1#1#2#3#4#5#6#7#8#9;{\CALX_fm_qp{#2#3#4#5#6#7#8#9}}%
\def\CALX_fm_qp #1#2;#3;#4;#5;#6;#7;#8;#9;{\CALX_fq_o{#1}{#2#3#4#5}}%
\def\CALX_fq_o #1#2#3#4%
{%
    \expandafter\CALX_fq_p\expanded{\ifnum#1>9999999 {24}{#1}\else{23}{%
```

```
        \xint_gobble_i#1}\fi
      {#2}{#3}{\the\numexpr#4+16}}%
}%
\def\CALX_fq_p #1#2#3#4#5% len, top, rest, s, E (P follows)
{%
    \expandafter\CALX_fq_q\the\numexpr(\xint_c_ii*#5-%
        \xint_c_vii)/16.#1.{#2#3}{#4}{#5}%
}%
\def\CALX_fq_q #1.#2.#3#4#5% e, len, S, s, E (P follows)
{%
    \expandafter\CALX_fq_t\the\numexpr#5-\xint_c_viii*#1.#2.{#1}{#3}{#4}%
}%
\def\CALX_fq_t #1.#2.#3#4#5#6% r, len, e, S, s, P
{%
    \CALX_fnround{#6}{\expanded{ F#5\the\numexpr#2+#1.#3.%
      \expandafter\CALX_fq_bl\expandafter{\expandafter}\expanded{%
      \CALX_ffr_zs{\numexpr\xint_c_viii*((\xint_c_ii*(#2+#1)+%
          \xint_c_vii)/16)-#2-#1\relax}%
      #4\CALX_ffr_zs{#1}}\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:}}%
}%
\def\CALX_fq_bl #1#2#3#4#5#6#7#8#9%
   {\xint_gob_til_xint: #2\CALX_fq_ble\xint:%
    \CALX_fq_bl{1#2#3#4#5#6#7#8#9!#1}}%
\def\CALX_fq_ble\xint:\CALX_fq_bl#1{\CALX_fq_bly#1}%
\def\CALX_fq_bly #1!{}%
```

31.7.7. \CALX_fm_v

20 < P <= 24, after \CALX_fround of both operands. Each mantissa M (D digits, D <= 32, the digits after the 24th are zeros) is read as A = M 10^(24-D), exactly 24 digits, in six groups of four digits: x0.x1.x2.x3.x4.x5. low group first for X (\CALX_fv_x, \CALX_fv_ga, _gb, _gc), y5.y4.y3.y2.y1.y0. high group first for Y (\CALX_fv_ma, _mb, _mc), digits only moved (\CALX_fv_r<n> reverses the n blocks, \CALX_fh_s<z> drops the z leading zeros of the top block). X Y = A A' 10^E with

E = 8(e + e') + D + D' - 48.

With c_k the sum of the x_i y_j, i + j = k, A A' is the sum of the c_k 10^(4k), k = 0..10. \CALX_fv_ka and \CALX_fv_kb compute the columns c_4 to c_10 in one chain of seven \numexpr (10^9 added, carry by \CALX_fm_s as in \CALX_fm_qa), 26 products. \CALX_fv_ka (nine arguments: the six x, then y5, y4, y3) only moves tokens: it writes the three top columns, the part of c_7 without y2 ending with x5*, then \CALX_fv_kb with the parts of c_6, c_5, c_4 without y2, y1, y0 (each ending with the x to be multiplied by y2) and x3, x4, x5; \CALX_fv_kb (nine arguments, the last three y2, y1, y0) completes the columns.

Truncation: W, the sum of the c_k 10^(4k) for k >= 4, is computed exactly; A A' - W, the sum for k <= 3, is less than 4.0004 10^20 since c_k <= (k+1)(10^4-1)^2. Let V = W 10^(-16) (31 or 32 digits, top column c_10 >= 10^6 as x5, y5 >= 1000) and m >= 7 the number of digits of V below the rounding position. The omitted part, less than 40004 units of V, changes the result only if V mod 10^m lies in [5 10^(m-1) - 40004, 5 10^(m-1)): the digits of V from position 5 to m-2 are 9, the digit m-1 is 4, and V mod 10^5 >= 59996. \CALX_fv_t and \CALX_fv_u test a necessary condition on the digits 4 to 7 of V (the group of c_5): digit 5 is 9 and digit 4 is at least 5, and digit 6 is 4 (m = 7), digits 6 and 7 are 9 and 4 (m = 8), digits 6 and 7 are 9 (m >= 9). If it holds, the exact product \CALX_fm_g runs (its call follows the arguments in the input stream). Otherwise \CALX_fv_ok drops the group of c_4 and finishes as \CALX_fm_q (\CALX_fq_p) with the 27 or 28 top digits of A A' and exponent E + 20.

```
\def\CALX_fm_v #1#2#3#4#5#6#7#8#9%
{%
    \expandafter\CALX_fv_ka\expanded{\CALX_fv_x{#2}\CALX_fv_ga#4\xint:%
    \CALX_fv_x{#6}\CALX_fv_ma#8\xint:%
    {\if#1#5+\else-\fi}{\the\numexpr\xint_c_viii*(#3+#7)+#2+#6-48}}{#9}%
    {\CALX_fm_g{#1}{#2}{#3}{#4}{#5}{#6}{#7}{#8}{#9}}%
}%
\def\CALX_fv_x #1{\expandafter\CALX_fv_xa\the\numexpr(\xint_c_ii*#1+%
    \xint_c_vii)/16.#1.}%
\def\CALX_fv_xa #1.#2.%
   {\csname CALX_fv_r#1\endcsname{\the\numexpr%
       \xint_c_viii*#1-#2}}%
\expandafter\def\csname CALX_fv_r1\endcsname #1#21#3!#4\xint:%
   {\csname CALX_fh_s#1\endcsname#2#3000000000000000000000000%
       \xint:}%
\expandafter\def\csname CALX_fv_r2\endcsname #1#21#3!1#4!#5\xint:%
   {\csname CALX_fh_s#1\endcsname#2#4#3000000000000000000000000%
       \xint:}%
\expandafter\def\csname CALX_fv_r3\endcsname #1#21#3!1#4!1#5!#6\xint:%
   {\csname CALX_fh_s#1%
       \endcsname#2#5#4#3000000000000000000000000\xint:}%
\expandafter\def\csname CALX_fv_r4\endcsname #1#21#3!1#4!1#5!1#6!#7\xint:%
   {\csname CALX_fh_s#1%
       \endcsname#2#6#5#4#3000000000000000000000000\xint:}%
\def\CALX_fv_ga #1#2#3#4#5#6#7#8{\CALX_fv_gb{#5#6#7#8.#1#2#3#4.}}%
\def\CALX_fv_gb #1#2#3#4#5#6#7#8#9{\CALX_fv_gc{#6#7#8#9.#2#3#4#5.#1}}%
\def\CALX_fv_gc #1#2#3#4#5#6#7#8#9{#6#7#8#9.#2#3#4#5.#1\CALX_fh_gz}%
\def\CALX_fv_ma #1#2#3#4#5#6#7#8{#1#2#3#4.#5#6#7#8.\CALX_fv_mb}%
\def\CALX_fv_mb #1#2#3#4#5#6#7#8{#1#2#3#4.#5#6#7#8.\CALX_fv_mc}%
\def\CALX_fv_mc #1#2#3#4#5#6#7#8#9\xint:{#1#2#3#4.#5#6#7#8.}%
\def\CALX_fv_ka #1.#2.#3.#4.#5.#6.#7.#8.#9.%
{%
    \expandafter\CALX_fv_o\the\numexpr\xint_c_e_ix+#6*#7+%
    \expandafter\CALX_fm_s\the\numexpr\xint_c_e_ix+#5*#7+#6*#8+%
    \expandafter\CALX_fm_s\the\numexpr\xint_c_e_ix+#4*#7+#5*#8+#6*#9+%
    \expandafter\CALX_fm_s\the\numexpr\xint_c_e_ix+#3*#7+#4*#8+#5*#9+#6*%
    \CALX_fv_kb{#2*#7+#3*#8+#4*#9+#5*}{#1*#7+#2*#8+#3*#9+#4*}%
                {#1*#8+#2*#9+#3*}%
                {#4}{#5}{#6}%
}%
\def\CALX_fv_kb #1#2#3#4#5#6#7.#8.#9.%
{%
    #7+%
    \expandafter\CALX_fm_s\the\numexpr\xint_c_e_ix+#1#7+#6*#8+%
    \expandafter\CALX_fm_s\the\numexpr\xint_c_e_ix+#2#7+#5*#8+#6*#9+%
    \expandafter\CALX_fm_s\the\numexpr\xint_c_e_ix+#3#7+#4*#8+#5*#9;%
}%
\def\CALX_fv_o 1#1#2#3#4#5#6#7#8#9;{\CALX_fv_p{#2#3#4#5#6#7#8#9}}%
\def\CALX_fv_p #1#2;#3;#4;#5;#6;#7;{\CALX_fv_t#6{#1}{#2#3#4#5#6}}%
\def\CALX_fv_t #1#2#3#4%
{%
    \ifnum#3#4>94 \xint_afterfi{\CALX_fv_u#1#2}\fi%
```

```
        \CALX_fv_ok
}%
\def\CALX_fv_u #1#2#3#4#5#6#7#8#9%
{%
    \ifnum\ifcase\numexpr\ifnum#4>9999999 25\else24\fi-#8\relax
            \if4#2\xint_c_i\else\xint_c_\fi
          \or\ifnum#1#2=49 \xint_c_i\else\xint_c_\fi
          \else\ifnum#1#2=99 \xint_c_i\else\xint_c_\fi\fi=\xint_c_i
       \xint_dothis{#9}\fi
    \xint_orthat{#3{#4}{#5}{#6}{#7}{#8}{#9}}%
}%
\def\CALX_fv_ok #1#2#3#4#5#6%
{%
    \expandafter\CALX_fq_p\expanded
      {\ifnum#1>9999999 {28}{#1}\else{27}{\xint_gobble_i#1}\fi
       {#2}{#3}{\the\numexpr#4+20}}{#5}%
}%
```

31.7.8. \CALX_fm_h

20 < P <= 34, after \CALX_fround of both operands. Each mantissa is read as A = M 10^(40-D) (40 digits, D <= 41) by \CALX_fh_x, in two halves of five groups: x0.x1.x2.x3.x4.!x5.x6.x7.x8.x9.! (A = AH 10^20 + AL). Three products by copies of the kernel of \CALX_fm_qa (\CALX_fh_qa: five blocks, low first; \CALX_fh_ra: AH AH' 10^4 in six blocks):

V = AH AH' 10^20 + AH AL' + AL AH', A A' = V 10^20 + AL AL'.

\CALX_fh_m, _t, _u add the blocks (one nested chain, top block first); the two lowest blocks of V are dropped after their carries.

Rounding: 0 <= AL AL' < 10^40, that is less than 10^20 units of V. With m >= 25 digits of V below the rounding position (true for P <= 34, V having at least 59 digits), the omitted term changes the result only if the digits of V from position 20 to m-1 read 4999...9; then digits 20 to 23 are 9999 (block V2 > 99989999). In that rare case \CALX_fh_fb runs the exact product \CALX_fm_g, whose call follows the arguments in the input stream; otherwise \CALX_fh_ok keeps the top blocks covering P + 1 digits and finishes as \CALX_fm_q (\CALX_fq_p).

```
\def\CALX_fm_h #1#2#3#4#5#6#7#8#9%
{%
    \expandafter\CALX_fh_a\expanded{\CALX_fh_x{#2}#4\xint:\CALX_fh_x{#6}#8%
        \xint:%
    {\if#1#5+\else-\fi}{\the\numexpr\xint_c_viii*(#3+#7)+#2+#6-60}}{#9}%
    {\CALX_fm_g{#1}{#2}{#3}{#4}{#5}{#6}{#7}{#8}{#9}}%
}%
\def\CALX_fh_x #1{\expandafter\CALX_fh_xa\the\numexpr(\xint_c_ii*#1+%
    \xint_c_vii)/16.#1.}%
\def\CALX_fh_xa #1.#2.%
   {\csname CALX_fh_r#1\endcsname{\the\numexpr%
       \xint_c_viii*#1-#2}}%
\expandafter\def\csname CALX_fh_r1\endcsname #11#2!#3\xint:%
   {\csname CALX_fh_s#1\endcsname\CALX_fh_ga %
       #2000000000000000000000000000000000000000\xint:}%
\expandafter\def\csname CALX_fh_r2\endcsname #11#2!1#3!#4\xint:%
   {\csname CALX_fh_s#1\endcsname\CALX_fh_ga %
       #3#2000000000000000000000000000000000000000\xint:}%
\expandafter\def\csname CALX_fh_r3\endcsname #11#2!1#3!1#4!#5\xint:%
   {\csname CALX_fh_s#1\endcsname\CALX_fh_ga %
```

```
        #4#3#2000000000000000000000000000000000000000\xint:}%
\expandafter\def\csname CALX_fh_r4\endcsname #11#2!1#3!1#4!1#5!#6\xint:%
   {\csname CALX_fh_s#1\endcsname\CALX_fh_ga %
        #5#4#3#200000000000000000000000000000000000000\xint:}%
\expandafter\def\csname CALX_fh_r5\endcsname #11#2!1#3!1#4!1#5!1#6!#7\xint:%
   {\csname CALX_fh_s#1\endcsname\CALX_fh_ga %
        #6#5#4#3#20000000000000000000000000000000000000\xint:}%
\expandafter\def\csname CALX_fh_r6\endcsname
   #11#2!1#3!1#4!1#5!1#6!1#7!#8\xint:%
   {\csname CALX_fh_s#1\endcsname\CALX_fh_ga %
        #7#6#5#4#3#2000000000000000000000000000000000000\xint:}%
\expandafter\def\csname CALX_fh_s0\endcsname #1{#1}%
\expandafter\def\csname CALX_fh_s1\endcsname #1#2{#1}%
\expandafter\def\csname CALX_fh_s2\endcsname #1#2#3{#1}%
\expandafter\def\csname CALX_fh_s3\endcsname #1#2#3#4{#1}%
\expandafter\def\csname CALX_fh_s4\endcsname #1#2#3#4#5{#1}%
\expandafter\def\csname CALX_fh_s5\endcsname #1#2#3#4#5#6{#1}%
\expandafter\def\csname CALX_fh_s6\endcsname #1#2#3#4#5#6#7{#1}%
\expandafter\def\csname CALX_fh_s7\endcsname #1#2#3#4#5#6#7#8{#1}%
\def\CALX_fh_ga #1#2#3#4#5#6#7#8{\CALX_fh_gb{#5#6#7#8.#1#2#3#4.!}}%
\def\CALX_fh_gb #1#2#3#4#5#6#7#8#9{\CALX_fh_gc{#6#7#8#9.#2#3#4#5.#1}}%
\def\CALX_fh_gc #1#2#3#4#5#6#7#8#9{\CALX_fh_gd{#6#7#8#9.!#2#3#4#5.#1}}%
\def\CALX_fh_gd #1#2#3#4#5#6#7#8#9{\CALX_fh_ge{#6#7#8#9.#2#3#4#5.#1}}%
\def\CALX_fh_ge #1#2#3#4#5#6#7#8#9{#6#7#8#9.#2#3#4#5.#1\CALX_fh_gz}%
\def\CALX_fh_gz #1\xint:{}%
\def\CALX_fh_a #1!#2!#3!#4!%
{%
    \expandafter\CALX_fh_m\expanded
      {\CALX_fh_qa #2#3\CALX_fh_qa #1#4\CALX_fh_ra #2#4}%
}%
\def\CALX_fh_m 1#1!1#2!1#3!1#4!1#5!1#6!1#7!1#8!1#9!%
   {\CALX_fh_t{#1+#6}{#2+#7}{#3+#8}{#4+#9}{#5}}%
\def\CALX_fh_t #1#2#3#4#51#6!1#7!1#8!1#9!%
   {\CALX_fh_u{#1}{#2}{#3+#7}{#4+#8}{#5+#6+#9}}%
\def\CALX_fh_u #1#2#3#4#51#6!1#7!1#8!%
{%
    \expandafter\CALX_fh_v\the\numexpr#8+%
    \expandafter\CALX_fh_c\the\numexpr\xint_c_e_ix+#7+%
    \expandafter\CALX_fh_c\the\numexpr\xint_c_e_ix+#6+%
    \expandafter\CALX_fh_c\the\numexpr\xint_c_e_ix+#5+%
    \expandafter\CALX_fh_c\the\numexpr\xint_c_e_ix+#4+%
    \expandafter\CALX_fh_c\the\numexpr\xint_c_e_ix+#3+%
    \expandafter\CALX_fh_c\the\numexpr\xint_c_e_ix+#2+%
    \expandafter\CALX_fh_c\the\numexpr\xint_c_e_ix+#1;%
}%
\def\CALX_fh_c 1#1{#1;}%
\def\CALX_fh_v #1;#2;#3;#4;#5;#6;#7;#8;%
{%
    \ifnum#6>99989999 \expandafter\CALX_fh_fb\else\expandafter\CALX_fh_ok\fi
    {#1}{#2#3#4}{#5}%
}%
\def\CALX_fh_fb #1#2#3#4#5#6#7{#7}%
```

```
\def\CALX_fh_ok #1#2#3#4#5#6#7%
{%
    \expandafter\CALX_fq_p\expanded
      {\ifnum#6>26 {\the\numexpr32+\CALX_ndig{#1}}{#1}{#2#3}{#4}{\the%
          \numexpr#5+24}%
       \else{\the\numexpr24+\CALX_ndig{#1}}{#1}{#2}{#4}{\the\numexpr#5+32}%
           \fi}{#6}%
}%
\def\CALX_fh_qa #1.#2.#3.#4.#5.#6.#7.#8.#9.%
{%
    \CALX_fh_qb{#5*}{#5*#9+#4*}{#4*#9+#5*#8+#3*}{#3*#9+#4*#8+#5*#7+#2*}%
               {#2*#9+#3*#8+#4*#7+#5*#6+#1*}%
    {\expandafter\CALX_fm_s\the\numexpr\xint_c_e_ix+#1*#9+#2*#8+#3*#7+#4*#6+%
     \expandafter\CALX_fm_s\the\numexpr\xint_c_e_ix+#1*#8+#2*#7+#3*#6+%
     \expandafter\CALX_fm_s\the\numexpr\xint_c_e_ix+#1*#7+#2*#6+%
     \expandafter\CALX_fm_s\the\numexpr\xint_c_e_ix+#1*#6;}%
}%
\def\CALX_fh_qb #1#2#3#4#5#6#7.%
{%
    \expandafter\CALX_fh_qo\the\numexpr\xint_c_e_ix+#1#7+%
    \expandafter\CALX_fm_s\the\numexpr\xint_c_e_ix+#2#7+%
    \expandafter\CALX_fm_s\the\numexpr\xint_c_e_ix+#3#7+%
    \expandafter\CALX_fm_s\the\numexpr\xint_c_e_ix+#4#7+%
    \expandafter\CALX_fm_s\the\numexpr\xint_c_e_ix+#5#7+#6%
}%
\def\CALX_fh_ra #1.#2.#3.#4.#5.#6.#7.#8.#9.%
{%
    \CALX_fh_rb{#5*}{#5*#9+#4*}{#4*#9+#5*#8+#3*}{#3*#9+#4*#8+#5*#7+#2*}%
               {#2*#9+#3*#8+#4*#7+#5*#6+#1*}%
    {\expandafter\CALX_fm_s\the\numexpr\xint_c_e_ix+#1*#9+#2*#8+#3*#7+#4*#6+%
     \expandafter\CALX_fm_s\the\numexpr\xint_c_e_ix+#1*#8+#2*#7+#3*#6+%
     \expandafter\CALX_fm_s\the\numexpr\xint_c_e_ix+#1*#7+#2*#6+%
     \expandafter\CALX_fm_s\the\numexpr\xint_c_e_ix+#1*#6;}%
}%
\def\CALX_fh_rb #1#2#3#4#5#6#7.%
{%
    \expandafter\CALX_fh_ro\the\numexpr\xint_c_e_ix+#1#7+%
    \expandafter\CALX_fm_s\the\numexpr\xint_c_e_ix+#2#7+%
    \expandafter\CALX_fm_s\the\numexpr\xint_c_e_ix+#3#7+%
    \expandafter\CALX_fm_s\the\numexpr\xint_c_e_ix+#4#7+%
    \expandafter\CALX_fm_s\the\numexpr\xint_c_e_ix+#5#7+#6%
}%
\def\CALX_fh_qo 1#1#2#3#4#5#6#7#8#9;{\CALX_fh_qp{#2#3#4#5#6#7#8#9}}%
\def\CALX_fh_qp #1#2;#3;#4;#5;#6;#7;#8;#9;{1#8#9!1#6#7!1#4#5!1#2#3!1#1!}%
\def\CALX_fh_ro 1#1#2#3#4#5#6#7#8#9;{\CALX_fh_rp{#2#3#4#5}{#6#7#8#9}}%
\def\CALX_fh_rp #1#2#3;#4;#5;#6;#7;#8;#9;{%
    \CALX_fh_rr{1#8#9!1#6#7!1#4#5!1#2#3!10000#1!}}%
\def\CALX_fh_rr #1#2;{1#20000!#1}%
```

31.7.9. \CALX_fm_big

Every case other than P <= 16 with mantissas of at most 24 digits: both operands are rounded by \CALX_fround, special values and zero are handled (a zero operand: \CALX_fm_zero gobbles the nine

arguments and gives F+0.0.), then \CALX_fm_c dispatches on P: P <= 20 to \CALX_fm_q, 20 < P <= 24 to \CALX_fm_v, 24 < P <= 34 to \CALX_fm_h, P > 34 to \CALX_fm_g.

\CALX_fm_g (also the rare fallback of \CALX_fm_v and \CALX_fm_h) is the exact generic path: the mantissas are multiplied by \CALX_nmul; the digit count of the product is read on its top block (\CALX_nskip, \CALX_fm_gl), and the native float is rounded by \CALX_fround.

\CALX_fm_gb N<s><n>.<blocks>\xint:{e}{P}: the list is grabbed once; one copy is walked to the top block (digit count D), the other is left in place after the header F<s><D>.<e>. given to \CALX_fround (which needs no terminator). e and D are evaluated in front by a \numexpr chain, so \CALX_fm_gd takes only small arguments.

```
\def\CALX_fm_big #1#2%
   {\expandafter\CALX_fm_a\romannumeral0\CALX_fround{#1}{#2}{#1}}%
\def\CALX_fm_a #1#2#3\xint:#4#5%
{%
    \if I#1\CALX_fm_asx\fi
    \expandafter\CALX_fm_btag\romannumeral0\CALX_fround{#4}{#5}{#4}#2#3%
    \xint:%
}%
\def\CALX_fm_asx\fi#1\CALX_fm_btag{\fi\expandafter\CALX_fm_asz}%
\def\CALX_fm_asz #1\xint:#2#3\xint:{\CALX_fm_sp#1\xint:I#3\xint:}%
\def\CALX_fm_btag #1#2{\if I#1\CALX_fm_bsy#2\fi\CALX_fm_c#2}%
\def\CALX_fm_bsy #1\fi#2\xint:#3#4\xint:{\fi\CALX_fm_sp I#1\xint:F#4\xint:}%
\def\CALX_fm_c #1#2.#3.#4\xint:#5#6#7.#8.#9\xint:%
{%
    \csname CALX_fm_\ifnum\ifnum#2=\xint_c_ \xint_c_\else#7\fi=\xint_c_ zero%
    \else\ifnum#5>34 g\else\ifnum#5>24 h\else\ifnum#5>20 v\else q%
    \fi\fi\fi\fi\endcsname{#6}{#7}{#8}{#9}{#1}{#2}{#3}{#4}{#5}%
}%
\def\CALX_fm_zero #1#2#3#4#5#6#7#8#9{ F+0.0.}%
\def\CALX_fm_g #1#2#3#4#5#6#7#8%
{%
    \expandafter\CALX_fm_gb\romannumeral0\CALX_nmul
      {\expanded{N#1\the\numexpr(\xint_c_ii*#2+\xint_c_vii)/16.}#4}%
      {\expanded{N#5\the\numexpr(\xint_c_ii*#6+\xint_c_vii)/16.}#8}%
          \xint:{#3+#7}%
}%
\def\CALX_fm_gb N#1#2.#3\xint:#4#5%
{%
    \expandafter\CALX_fm_gd\the\numexpr#4\expandafter.\the\numexpr
    \xint_c_viii*(#2-\xint_c_i)+\CALX_nskip\CALX_fm_gl #2-\xint_c_i.#3\xint:.%
    #1{#5}#3%
}%
\def\CALX_fm_gl 1#1!#2\xint:{\CALX_ndig{#1}}%
\def\CALX_fm_gd #1.#2.#3#4{\CALX_fnround{#4}{ F#3#2.#1.}}%
```

\CALX_fmulx{X}{Y} (use with \romannumeral0; X, Y finite natives, any digit counts): the exact product X Y as a native float, not rounded (D up to D_X + D_Y). Zero operand: F+0.0. Same code as \CALX_fm_g (\CALX_nmul on the mantissas, digit count read on the top block), without the final \CALX_fround.

\CALX_fmulxs{X}{Y}: the same for values coming from the parsers, whose tag may have catcode 11 (values stored in the meaning of a compiled function, read from a file): \string makes the tag of each operand, after its f-expansion, of catcode 12, as the patterns of \CALX_fmx_b require. \CALX_fmulx itself is for the values calx computes.

Delimited entry for the two already expanded operands. The ordinary \CALX_fmulxs entry remains

for internal callers with expandable operands.

```
\def\CALX_fmulx #1%
   {\expandafter\CALX_fmx_a\romannumeral`&&@#1\xint:}%
\def\CALX_fmx_a #1\xint:#2%
   {\expandafter\CALX_fmx_b\romannumeral`&&@#2\xint:#1\xint:}%
\def\CALX_fmulxs_in #1\xint:#2\xint:%
   {\expandafter\CALX_fmx_vi\string#1\xint:#2\xint:}%
\def\CALX_fmx_vi #1\xint:#2\xint:%
   {\expandafter\CALX_fmx_b\string#2\xint:#1\xint:}%
\def\CALX_fmulxs #1%
   {\expandafter\CALX_fmx_s\romannumeral`&&@#1\xint:}%
\def\CALX_fmx_s{\expandafter\CALX_fmx_t\string}%
\def\CALX_fmx_t #1\xint:#2%
   {\expandafter\CALX_fmx_u\romannumeral`&&@#2\xint:#1\xint:}%
\def\CALX_fmx_u{\expandafter\CALX_fmx_b\string}%
\def\CALX_fmx_b F#1#2.#3.#4\xint:F#5#6.#7.#8\xint:%
{%
    \ifnum\ifnum#2=\xint_c_ \xint_c_\else#6\fi=\xint_c_
      \xint_dothis\CALX_fmx_zero\fi
    \xint_orthat\CALX_fmx_nonzero
    F#1#2.#3.#4\xint:F#5#6.#7.#8\xint:
}%
\def\CALX_fmx_zero #1\xint:#2\xint:{ F+0.0.}%
\def\CALX_fmx_nonzero F#1#2.#3.#4\xint:F#5#6.#7.#8\xint:%
{%
    \expandafter\CALX_fmx_c\romannumeral0\CALX_nmul
      {\expanded{N#1\the\numexpr(\xint_c_ii*#2+\xint_c_vii)/16.}#4}%
      {\expanded{N#5\the\numexpr(\xint_c_ii*#6+\xint_c_vii)/16.}#8}%
      \xint:{#3+#7}%
}%
\def\CALX_fmx_c N#1#2.#3\xint:#4%
{%
    \expandafter\CALX_fmx_d\the\numexpr#4\expandafter.\the\numexpr
    \xint_c_viii*(#2-\xint_c_i)+\CALX_nskip\CALX_fm_gl #2-\xint_c_i.#3\xint:.%
    #1#3%
}%
\def\CALX_fmx_d #1.#2.#3{ F#3#2.#1.}%
```

\CALX_faddx{X}{Y}, \CALX_fsubx{X}{Y} (use with \romannumeral0; X, Y finite natives): X + Y, X - Y exact, not rounded. The operand with the larger block exponent gets that many zero blocks at its low end (\CALX_fa_z), both become integers N<s><n>.<blocks> (same blocks, n = ceil(D/8)), \CALX_nadd or \CALX_nsub adds them, and the digit count of the result is read on its top block as in \CALX_fm_g. A zero operand gives the other one.

```
\def\CALX_faddx #1#2%
   {\expandafter\CALX_fx_a\romannumeral`&&@#2\xint:{#1}\CALX_nadd}%
\def\CALX_fsubx #1#2%
   {\expandafter\CALX_fx_a\romannumeral`&&@#2\xint:{#1}\CALX_nsub}%
\def\CALX_fx_a F#1#2.#3.#4\xint:#5#6%
   {\expandafter\CALX_fx_b\romannumeral`&&@#5\xint:#6#1#2.#3.#4\xint:}%
\def\CALX_fx_b F#1#2.#3.#4\xint:#5#6#7.#8.#9\xint:%
{%
    \ifnum#7=\xint_c_ \xint_dothis\CALX_fx_zy\fi
    \ifnum#2=\xint_c_ \xint_dothis\CALX_fx_zx\fi
```

```
    \xint_orthat\CALX_fx_nonzero
    F#1#2.#3.#4\xint:#5#6#7.#8.#9\xint:
}%
\def\CALX_fx_zy F#1#2.#3.#4\xint:#5#6#7.#8.#9\xint:%
   { F#1#2.#3.#4}%
\def\CALX_fx_zx F#1#2.#3.#4\xint:#5#6#7.#8.#9\xint:%
{%
    \ifx#5\CALX_nsub\expandafter\CALX_fx_m\else\expandafter\space\fi
    F#6#7.#8.#9%
}%
\def\CALX_fx_nonzero F#1#2.#3.#4\xint:#5#6#7.#8.#9\xint:%
{%
    \expandafter\CALX_fx_c\the\numexpr\ifnum#3<#8 #3\else#8\fi
      \expandafter.\romannumeral0#5%
      {\expanded{N#1\the\numexpr(2*#2+7)/16\ifnum#3>#8 +#3-#8\fi.%
        \ifnum#3>#8 \CALX_fa_z#3-#8.\fi}#4}%
      {\expanded{N#6\the\numexpr(2*#7+7)/16\ifnum#8>#3 +#8-#3\fi.%
        \ifnum#8>#3 \CALX_fa_z#8-#3.\fi}#9}\xint:%
}%
\def\CALX_fx_m F#1%
  {\if-#1\expandafter\CALX_fx_p\else\expandafter\CALX_fx_n\fi}%
\def\CALX_fx_p{ F+}%
\def\CALX_fx_n{ F-}%
\def\CALX_fx_c #1.N#2#3.%
{%
    \ifnum#3=\xint_c_ \expandafter\CALX_fx_z\else
      \expandafter\CALX_fx_d\fi #1.#2#3.%
}%
\def\CALX_fx_z #1\xint:{ F+0.0.}%
\def\CALX_fx_d #1.#2#3.#4\xint:%
{%
    \expandafter\CALX_fmx_d\the\numexpr#1\expandafter.\the\numexpr
    \xint_c_viii*(#3-\xint_c_i)+\CALX_nskip\CALX_fm_gl #3-\xint_c_i.#4\xint:.%
    #2#4%
}%
```

31.8. Squaring

31.8.1. \CALX_fsqr

Syntax (use with \romannumeral0; P a \numexpr expression; X f-expanded F, I or raw value):
  \CALX_fsqr{P}{X} -> <space>Z Z = round_P(round_P(X)^2)
    \CALX_fsqru{P}{X} -> <space>Z   (X finite F, at most P digits)
\CALX_fsqr{P}{X} (use with \romannumeral0, X f-expanded): round_P(round_P(X)^2), round half away from zero, as xint's \XINTinFloatSqr[P]; the same result as \CALX_fmul{P}{X}{X}, with the operand read once and kernels which compute each cross product once (x_i x_j + x_j x_i written as 2 x_i x_j).
Special values: I? gives I?, I+ and I- give I+. Zero gives F+0.0.
  P <= 16 and a mantissa of at most 24 digits: one extraction as in \CALX_fmul (\CALX_fm_xvi, \CALX_fm_ex), then \CALX_fs_k (the square of the 16 digit A, seven columns, ten products) and \CALX_fm_r. Otherwise \CALX_fs_big: \CALX_fround, then by P: \CALX_fs_q (P <= 20), \CALX_fs_v (P <= 24), \CALX_fs_h (P <= 34), \CALX_fs_g (generic, also the fallback of \CALX_fs_v and \CALX_fs_h). They read the operand as \CALX_fm_q, \CALX_fm_v, \CALX_fm_h read X, with exponents

E = 16e + 2D - 40, E = 16e + 2D - 48, E = 16e + 2D - 60,
and finish as these. With x_i the groups of four digits, the column c_k of A^2 is
c_k = 2 (sum of x_i x_(k-i), i < k - i) + x_(k/2)^2,
the square term only for k even: its value is at most that of the column of a product (same number of terms), so the carry scheme of \CALX_fm_s is unchanged; the truncation bounds of \CALX_fm_v and \CALX_fm_h hold as they are (they only use the sizes of the operands). \CALX_fs_q: 15 products instead of 25, one kernel macro of five arguments; \CALX_fs_v: 15 instead of 26, one macro of six arguments; \CALX_fs_h: two kernels instead of three (AH AL once, \CALX_fs_ra for AH^2), \CALX_fs_m doubles the blocks of AH AL in the merge of \CALX_fh_u.

```
\def\CALX_fsqr #1%
{%
    \ifnum#1>16 \expandafter\CALX_fs_big\else\expandafter\CALX_fs_na\fi
    {#1}%
}%
\def\CALX_fsqru #1%
{%
    \ifnum#1>16 \expandafter\CALX_fqu_b\else\expandafter\CALX_fs_na\fi
    {#1}%
}%
\def\CALX_fqu_b #1#2{\expandafter\CALX_fs_b\romannumeral`&&@#2\xint:{#1}}%
\def\CALX_fs_na #1#2{\expandafter\CALX_fs_nc\romannumeral`&&@#2\xint:{#1}}%
\def\CALX_fs_nc #1%
   {\if F#1\expandafter\CALX_fs_nd\else\expandafter\CALX_fs_ns
    \expandafter#1\fi}%
\def\CALX_fs_ns #1%
   {\if I#1\expandafter\CALX_fs_sp\else\expandafter\CALX_fs_nr
    \expandafter#1\fi}%
\def\CALX_fs_nr #1#2\xint:#3{\CALX_fs_big{#3}{#1#2}}%
\def\CALX_fs_sp #1#2\xint:#3{\if ?#1\xint_dothis{ I?}\fi\xint_orthat{ I+}}%
\def\CALX_fs_nd #1#2.#3.#4\xint:#5%
{%
    \ifnum#2=\xint_c_ \CALX_fs_nz\fi
    \ifnum#2>24 \CALX_fs_nbig{#5}{F#1#2.#3.#4}\fi
    \expandafter\CALX_fs_ne\expanded
      {\CALX_fm_ny{#5}{#2}#4100000000!100000000!\xint:{\the\numexpr16*#3}}%
    {#5}%
}%
\def\CALX_fs_nz\fi#1\expanded#2#3{\fi\CALX_sp F+0.0.}%
\def\CALX_fs_nbig #1#2\fi#3\expanded#4#5{\fi\CALX_fs_big{#1}{#2}}%
\def\CALX_fs_ne #1!#2!#3.#4{\CALX_fs_k #1!#2!{+}{#4+#3+#3}}%
\def\CALX_fs_k 1#1#2#3#4#5!{\CALX_fs_kb{#5}{#1#2#3#4}}%
\def\CALX_fs_kb #1#21#3#4#5#6#7!{\CALX_fs_kf{#1}{#2}{#7}{#3#4#5#6}}%
\def\CALX_fs_kf #1#2#3#4%
{%
    \expandafter\CALX_fm_o\the\numexpr1000000000+#4*#4+%
    \expandafter\CALX_fm_s\the\numexpr1000000000+#3*#4*\xint_c_ii+%
    \expandafter\CALX_fm_s\the\numexpr1000000000+#2*#4*\xint_c_ii+#3*#3+%
    \expandafter\CALX_fm_s\the\numexpr1000000000+(#1*#4+#2*#3)*\xint_c_ii+%
    \expandafter\CALX_fm_s\the\numexpr1000000000+#1*#3*\xint_c_ii+#2*#2+%
    \expandafter\CALX_fm_s\the\numexpr1000000000+#1*#2*\xint_c_ii+%
    \expandafter\CALX_fm_s\the\numexpr1000000000+#1*#1;%
}%
```

```
\def\CALX_fs_big #1#2%
   {\expandafter\CALX_fs_b\romannumeral0\CALX_fround{#1}{#2}{#1}}%
\def\CALX_fs_b #1%
   {\if I#1\expandafter\CALX_fs_sp\else\expandafter\CALX_fs_c\fi}%
\def\CALX_fs_c #1#2.#3.#4\xint:#5%
{%
    \csname CALX_fs_\ifnum#2=\xint_c_ zero\else\ifnum#5>34 g\else
    \ifnum#5>24 h\else\ifnum#5>20 v\else q\fi\fi\fi\endcsname
    {#1}{#2}{#3}{#4}{#5}%
}%
\def\CALX_fs_zero #1#2#3#4#5{ F+0.0.}%
\def\CALX_fs_g #1#2#3#4{\CALX_fm_g{#1}{#2}{#3}{#4}{#1}{#2}{#3}{#4}}%
\def\CALX_fs_q #1#2#3#4%
{%
    \expandafter\CALX_fs_qa\expanded{\CALX_fq_x{#2}#4\xint:%
      {+}{\the\numexpr16*#3+\xint_c_ii*#2-40}}%
}%
\def\CALX_fs_qa #1.#2.#3.#4.#5.%
{%
    \expandafter\CALX_fm_qo\the\numexpr\xint_c_e_ix+#5*#5+%
    \CALX_fs_qc{#1}{#2}{#3}{#4}{#5}%
}%
\def\CALX_fs_qc #1#2#3#4#5%
{%
    \expandafter\CALX_fm_s\the\numexpr\xint_c_e_ix+#4*#5*\xint_c_ii+%
    \expandafter\CALX_fm_s\the\numexpr\xint_c_e_ix+#3*#5*\xint_c_ii+#4*#4+%
    \expandafter\CALX_fm_s\the\numexpr\xint_c_e_ix+(#2*#5+#3*#4)*%
        \xint_c_ii+%
    \expandafter\CALX_fm_s\the\numexpr\xint_c_e_ix+(#1*#5+#2*#4)*%
        \xint_c_ii+#3*#3+%
    \expandafter\CALX_fm_s\the\numexpr\xint_c_e_ix+(#1*#4+#2*#3)*%
        \xint_c_ii+%
    \expandafter\CALX_fm_s\the\numexpr\xint_c_e_ix+#1*#3*\xint_c_ii+#2*#2+%
    \expandafter\CALX_fm_s\the\numexpr\xint_c_e_ix+#1*#2*\xint_c_ii+%
    \expandafter\CALX_fm_s\the\numexpr\xint_c_e_ix+#1*#1;%
}%
\def\CALX_fs_v #1#2#3#4#5%
{%
    \expandafter\CALX_fs_va\expanded{\CALX_fv_x{#2}\CALX_fv_ga#4\xint:%
    {+}{\the\numexpr16*#3+\xint_c_ii*#2-48}}{#5}%
    {\CALX_fs_g{#1}{#2}{#3}{#4}{#5}}%
}%
\def\CALX_fs_va #1.#2.#3.#4.#5.#6.%
{%
    \expandafter\CALX_fv_o\the\numexpr\xint_c_e_ix+#6*#6+%
    \expandafter\CALX_fm_s\the\numexpr\xint_c_e_ix+#5*#6*\xint_c_ii+%
    \expandafter\CALX_fm_s\the\numexpr\xint_c_e_ix+#4*#6*\xint_c_ii+#5*#5+%
    \expandafter\CALX_fm_s\the\numexpr\xint_c_e_ix+(#3*#6+#4*#5)*%
        \xint_c_ii+%
    \expandafter\CALX_fm_s\the\numexpr\xint_c_e_ix+(#2*#6+#3*#5)*%
        \xint_c_ii+#4*#4+%
    \expandafter\CALX_fm_s\the\numexpr\xint_c_e_ix+(#1*#6+#2*#5+#3*#4)*%
```

```
        \xint_c_ii+%
    \expandafter\CALX_fm_s\the\numexpr\xint_c_e_ix+(#1*#5+#2*#4)*%
        \xint_c_ii+#3*#3;%
}%
\def\CALX_fs_h #1#2#3#4#5%
{%
    \expandafter\CALX_fs_a\expanded{\CALX_fh_x{#2}#4\xint:%
    {+}{\the\numexpr16*#3+\xint_c_ii*#2-60}}{#5}%
    {\CALX_fs_g{#1}{#2}{#3}{#4}{#5}}%
}%
\def\CALX_fs_a #1!#2!%
   {\expandafter\CALX_fs_m\expanded{\CALX_fh_qa #2#1\CALX_fs_ra #2}}%
\def\CALX_fs_m 1#1!1#2!1#3!1#4!1#5!1#6!1#7!1#8!%
{%
    \CALX_fh_u{#1*\xint_c_ii}{#2*\xint_c_ii}{#3*\xint_c_ii+#6}%
      {#4*\xint_c_ii+#7}{#5*\xint_c_ii+#8}%
}%
\def\CALX_fs_ra #1.#2.#3.#4.#5.%
{%
    \expandafter\CALX_fh_ro\the\numexpr\xint_c_e_ix+#5*#5+%
    \CALX_fs_qc{#1}{#2}{#3}{#4}{#5}%
}%
```

31.9. Division

31.9.1. \CALX_fdiv

Syntax (use with \romannumeral0; P a \numexpr expression; X, Y f-expanded F, I or raw values):
  \CALX_fdiv{P}{X}{Y} -> <space>Z Z = round_P(round_P(X)/round_P(Y))
    \CALX_fdivu{P}{X}{Y} -> <space>Z   (X, Y finite F, at most P digits)
\CALX_fdiv{P}{X}{Y} (use with \romannumeral0, X and Y f-expanded): round_P(round_P(X) / round_P(Y)), round half away from zero, as xint's \XINTinFloatDiv[P].
Both operands are rounded by \CALX_fround. Special values (\CALX_fd_sp: I? propagates, Inf / Inf signals InvalidOperation, Inf / finite is Inf with the sign of the quotient, finite / Inf is zero), then zero operands: x / 0 signals DivisionByZero (handler argument the sign of x), 0 / 0 DivisionUndefined, 0 / y is F+0.0.
  Generic path (\CALX_fd_g): the quotient q = Mx 10^(8a) / My satisfies q > 10^(Dx + 8a - Dy - 1), so with
     a = max(0, ceil((P + 1 + Dy - Dx)/8))
zero blocks in front of Mx (\CALX_fa_z), the truncated integer quotient N (\CALX_ndivtrunc, sign of the quotient) has at least P + 1 digits. Round half away from zero to P digits depends only on the P + 1 top digits of the exact quotient, which are those of N; so the native float N 10^(8(ex - ey - a)) is rounded by \CALX_fround as in \CALX_fm_g (\CALX_fm_gb).

```
\def\CALX_fdiv #1#2%
   {\expandafter\CALX_fd_a\romannumeral0\CALX_fround{#1}{#2}{#1}}%
\def\CALX_fdivu #1#2%
   {\expandafter\CALX_fdu\romannumeral`&&@#2\xint:{#1}}%
\def\CALX_fdu F#1\xint:#2#3%
   {\expandafter\CALX_fd_btag\romannumeral`&&@#3\xint:{#2}#1\xint:}%
\def\CALX_fd_a #1#2#3\xint:#4#5%
{%
    \if I#1\CALX_fd_asx#2\fi
    \expandafter\CALX_fd_btag\romannumeral0\CALX_fround{#4}{#5}{#4}#2#3%
```

```
    \xint:%
}%
\def\CALX_fd_asx #1\fi#2\CALX_fd_btag%
   {\fi\expandafter\CALX_fd_asz\expandafter#1}%
\def\CALX_fd_asz #1#2#3#4\xint:#5#6\xint:{\CALX_fd_sp I#1#2#3}%
\def\CALX_fd_btag #1#2{\if I#1\CALX_fd_bsy#2\fi\CALX_fd_c#2}%
\def\CALX_fd_bsy #1\fi#2\xint:#3#4#5\xint:{\fi\CALX_fd_sp F#4I#1}%
\def\CALX_fd_sp #1#2#3#4%
{%
    \if ?#2\xint_dothis{ I?}\fi
    \if ?#4\xint_dothis{ I?}\fi
    \if F#1\xint_dothis{ F+0.0.}\fi
    \if F#3\xint_dothis{\CALX_nsp_ps#2#4}\fi
    \xint_orthat{\CALX_nsig{InvalidOperation}{inf / inf}{}}%
}%
\def\CALX_fd_c #1#2.#3.#4\xint:#5#6#7.#8.#9\xint:%
{%
    \csname CALX_fd_\ifnum#2=\xint_c_ zz\else\ifnum#7=\xint_c_ zero\else
    \ifnum#5>40 g\else\ifnum#5<5 g\else v\fi\fi\fi\endcsname
    {#6}{#7}{#8}{#9}{#1}{#2}{#3}{#4}{#5}%
}%
\def\CALX_fd_zz #1#2#3#4#5#6#7#8#9{\CALX_fd_z{#2}{#1}}%
\let\CALX_fd_zero\CALX_fm_zero
\def\CALX_fd_z #1#2%
{%
    \ifnum#1=\xint_c_ \xint_dothis{\CALX_nsig{DivisionUndefined}{0 / 0}{}}\fi
    \xint_orthat{\CALX_nsig{DivisionByZero}{x / 0}{#2}}%
}%
\def\CALX_fd_g #1#2#3#4#5#6#7#8#9%
{%
    \expandafter\CALX_fd_i\the\numexpr
      \ifnum\numexpr#9+\xint_c_i+#6-#2<\xint_c_ \xint_c_\else
      (\xint_c_ii*(#9+\xint_c_i+#6-#2)+\xint_c_vii)/16\fi.%
      {#1}{#2}{#3}{#4}{#5}{#6}{#7}{#8}{#9}%
}%
\def\CALX_fd_i #1.#2#3#4#5#6#7#8#9%
{%
    \expandafter\CALX_fm_gb\romannumeral0\CALX_ndivtrunc
      {\expanded{N#2\the\numexpr(\xint_c_ii*#3+\xint_c_vii)/16+#1.%
        \CALX_fa_z#1.}#5}%
      {\expanded{N#6\the\numexpr(\xint_c_ii*#7+\xint_c_vii)/16.}#9}%
    \xint:{#4-#8-#1}%
}%
```

31.9.2. \CALX_fd_v

Division kernel for 5 <= P <= 40 (\CALX_fd_c calls \CALX_fd_v instead of \CALX_fd_g). With m = ceil(P/4) and n = ceil((P+1)/4), each mantissa is read as an integer of exactly 4m digits (A = Mx 10^(4m-Dx), B = My 10^(4m-Dy), the dropped digits are zeros), in groups of four digits a_i, b_i (\CALX_dv_rd, \CALX_dv_r<k> reverses the k blocks, \CALX_fh_s<z> drops the z leading zeros of the top block, \CALX_dv_gg takes m groups; the counter tokens \CALX_dv_k<i> expand to \CALX_dv_k<i-1>). Then

X / Y = (A / B) 10^E, E = 8(ex - ey) + Dx - Dy.

Long division in base 10^4 with a signed remainder: R_0 = A, and for i = 1..n
q_i = round(10^4 R'/B'), R_i = 10^4 R_(i-1) - q_i B,
R' and B' being the two top groups of R_(i-1) and B (\numexpr a*b/c, 64 bit intermediate product). The estimate is within 0.0015 of 10^4 R_(i-1)/B, so |R_i| <= 0.5015 B for i >= 1, |q_i| <= 5100 for i >= 2, 1000 <= q_1 < 10^5, and no correction step is needed. Then
A 10^(4n) = B (q_1 10^(4(n-1)) + ... + q_n) + R_n,
so Q = floor(A 10^(4n)/B) is that sum, minus one if R_n < 0; Q has at least 4n >= P + 1 digits, enough for the rounding (as in \CALX_fd_g).

Data of a step (top first): <prefix><counter> b_(m-1) T; then the entries l.\CALX_dv_c b; of the lower groups of R and B, then 0.\CALX_dv_e; (T is the signed top group of R, the l are in [0, 10^4)). \CALX_dv_s computes q, \CALX_dv_sq writes the top column of R_i, the next columns are written by \CALX_dv_c (one entry l.\CALX_dv_c b; each: the entries carry the column macro, the next one is called with q after its b, while \numexpr is evaluating the column above: one nested chain of m \numexpr; the last entry 0.\CALX_dv_e; ends the chain). A column computes 2 10^9 + l - q b + carry (ten digits); the four digits of b are put in front of the result (\expandafter over single digits), \CALX_dv_sb, \CALX_dv_sc split it into the carry (six digits minus 200000) and the new group, and put the entry back after it. So the result has the layout of the input; the top column ends with \CALX_dv_tt, which starts the next step with the counter decremented.

The q_i are written into the output of \expanded as the text of a second nested chain: \CALX_dv_n1, \CALX_dv_ns split 2 10^8 + q_i + carry (nine digits), the last column gets the correction (-1 if T < 0 at the end, \CALX_dv_f). The groups are glued (\relax ends the \numexpr of the columns), \CALX_dv_nt receives the top carry and the 4n digits and finishes as \CALX_fm_q (\CALX_fq_p, exponent E - 4n).

```
\def\CALX_fd_v #1#2#3#4#5#6#7#8#9%
{%
    \expandafter\CALX_dv_a\the\numexpr(\xint_c_ii*#9+\xint_c_iii)/%
    \xint_c_viii\expandafter.\the\numexpr(\xint_c_ii*#9+5)/\xint_c_viii.%
    {#2}{#4}{#6}{#8}{\if#1#5+\else-\fi}{\xint_c_viii*(#3-#7)+#2-#6}{#9}%
}%
\def\CALX_dv_a #1.#2.#3#4#5#6% m, n, Dx, Mx, Dy, My
{%
    \expanded{\unexpanded{\expandafter\CALX_dv_nt\the\numexpr}0+%
      \expandafter\CALX_dv_s\expandafter\CALX_dv_ni
      \csname CALX_dv_k#2\expandafter\endcsname
      \expanded{\expandafter\CALX_dv_il\expanded{%
        \CALX_dv_rd{#3}{#1}#4\xint:!.\xint:\CALX_dv_rd{#5}{#1}#6\xint:!.%
            \xint:}}}%
    {#2}%
}%
\def\CALX_dv_rd #1%
   {\expandafter\CALX_dv_rda\the\numexpr(\xint_c_ii*#1+%
       \xint_c_vii)/16.#1.}%
\def\CALX_dv_rda #1.#2.#3%
{%
    \csname CALX_dv_r#1\expandafter\endcsname
    \the\numexpr\xint_c_viii*#1-#2\expandafter.\csname CALX_dv_g#3\endcsname
}%
\expandafter\def\csname CALX_dv_r1\endcsname #1.#21#3!#4\xint:%
   {\csname CALX_fh_s#1\endcsname#2#3%
    0000000000000000000000000000000000000000000000000\xint:}%
\expandafter\def\csname CALX_dv_r2\endcsname #1.#21#3!1#4!#5\xint:%
   {\csname CALX_fh_s#1\endcsname#2#4#3%
```

```
    00000000000000000000000000000000000000000000000000\xint:}%
\expandafter\def\csname CALX_dv_r3\endcsname #1.#21#3!1#4!1#5!#6\xint:%
   {\csname CALX_fh_s#1\endcsname#2#5#4#3%
    00000000000000000000000000000000000000000000000000\xint:}%
\expandafter\def\csname CALX_dv_r4\endcsname #1.#21#3!1#4!1#5!1#6!#7\xint:%
   {\csname CALX_fh_s#1\endcsname#2#6#5#4#3%
    00000000000000000000000000000000000000000000000000\xint:}%
\expandafter\def\csname CALX_dv_r5\endcsname
   #1.#21#3!1#4!1#5!1#6!1#7!#8\xint:%
   {\csname CALX_fh_s#1\endcsname#2#7#6#5#4#3%
    00000000000000000000000000000000000000000000000000\xint:}%
\expandafter\def\csname CALX_dv_r6\endcsname
   #1.#21#3!1#4!1#5!1#6!1#7!1#8!#9\xint:%
   {\csname CALX_fh_s#1\endcsname#2#8#7#6#5#4#3%
    00000000000000000000000000000000000000000000000000\xint:}%
\def\CALX_dv_gg #1#2#3#4#5%
{%
    \ifx#1\CALX_dv_kz\xint_dothis\CALX_dv_gz\fi
    \xint_orthat{#2#3#4#5.\expandafter\CALX_dv_gg#1}%
}%
\def\CALX_dv_gz #1\xint:{}%
\def\CALX_dv_il #1.#2\xint:#3.#4\xint:{#3#1;\CALX_dv_im#2\xint:#4\xint:}%
\def\CALX_dv_im #1.#2\xint:#3.#4\xint:%
{%
    \xint_gob_til_exclam #1\CALX_dv_ie!%
    #1.\noexpand\CALX_dv_c#3;\CALX_dv_im#2\xint:#4\xint:%
}%
\def\CALX_dv_ie!#1\xint:#2\xint:{0.\noexpand\CALX_dv_e;}%
\def\CALX_dv_kz {z}%
\expandafter\let\csname CALX_dv_k0\endcsname\CALX_dv_kz
\expandafter\def\csname CALX_dv_k1\endcsname{\CALX_dv_kz}%
\expandafter\edef\csname CALX_dv_k2\endcsname
   {\expandafter\noexpand\csname CALX_dv_k1\endcsname}%
\expandafter\edef\csname CALX_dv_k3\endcsname
   {\expandafter\noexpand\csname CALX_dv_k2\endcsname}%
\expandafter\edef\csname CALX_dv_k4\endcsname
   {\expandafter\noexpand\csname CALX_dv_k3\endcsname}%
\expandafter\edef\csname CALX_dv_k5\endcsname
   {\expandafter\noexpand\csname CALX_dv_k4\endcsname}%
\expandafter\edef\csname CALX_dv_k6\endcsname
   {\expandafter\noexpand\csname CALX_dv_k5\endcsname}%
\expandafter\edef\csname CALX_dv_k7\endcsname
   {\expandafter\noexpand\csname CALX_dv_k6\endcsname}%
\expandafter\edef\csname CALX_dv_k8\endcsname
   {\expandafter\noexpand\csname CALX_dv_k7\endcsname}%
\expandafter\edef\csname CALX_dv_k9\endcsname
   {\expandafter\noexpand\csname CALX_dv_k8\endcsname}%
\expandafter\edef\csname CALX_dv_k10\endcsname
   {\expandafter\noexpand\csname CALX_dv_k9\endcsname}%
\expandafter\edef\csname CALX_dv_k11\endcsname
   {\expandafter\noexpand\csname CALX_dv_k10\endcsname}%
\expandafter\def\csname CALX_dv_g2\endcsname
```

```
   {\expandafter\CALX_dv_gg\csname CALX_dv_k2\endcsname}%
\expandafter\def\csname CALX_dv_g3\endcsname
   {\expandafter\CALX_dv_gg\csname CALX_dv_k3\endcsname}%
\expandafter\def\csname CALX_dv_g4\endcsname
   {\expandafter\CALX_dv_gg\csname CALX_dv_k4\endcsname}%
\expandafter\def\csname CALX_dv_g5\endcsname
   {\expandafter\CALX_dv_gg\csname CALX_dv_k5\endcsname}%
\expandafter\def\csname CALX_dv_g6\endcsname
   {\expandafter\CALX_dv_gg\csname CALX_dv_k6\endcsname}%
\expandafter\def\csname CALX_dv_g7\endcsname
   {\expandafter\CALX_dv_gg\csname CALX_dv_k7\endcsname}%
\expandafter\def\csname CALX_dv_g8\endcsname
   {\expandafter\CALX_dv_gg\csname CALX_dv_k8\endcsname}%
\expandafter\def\csname CALX_dv_g9\endcsname
   {\expandafter\CALX_dv_gg\csname CALX_dv_k9\endcsname}%
\expandafter\def\csname CALX_dv_g10\endcsname
   {\expandafter\CALX_dv_gg\csname CALX_dv_k10\endcsname}%
\def\CALX_dv_s #1#2#3#4#5#6#7;#8.\CALX_dv_c#9;%
{%
    \ifx#2\CALX_dv_kz\xint_dothis{\CALX_dv_f{#7}}\fi
    \xint_orthat{\expandafter\CALX_dv_sq\the\numexpr
      (#7*10000+#8)*10000/(#3#4#5#6*10000+#9).%
      #1#2{#3#4#5#6}{#7}{#8}{#9}}%
}%
\def\CALX_dv_f #1#2\CALX_dv_e;{\ifnum#1<\xint_c_ -1\else0\fi;}%
\def\CALX_dv_sq #1.#2#3#4#5#6#7% q, prefix, counter, b, T, l, b'
{%
    \unexpanded{\expandafter#2\the\numexpr}#1+200000000+%
    \expandafter\CALX_dv_tt#3#4\the\numexpr
      #5*10000+#6-#1*#4+\CALX_dv_c#7{#1}%
}%
\def\CALX_dv_tt #1#2#3#4%
{%
    \expandafter\CALX_dv_s\expandafter\CALX_dv_ns\expandafter#1%
    \expandafter#2\expandafter#3\expandafter#4\expandafter
}%
\def\CALX_dv_c #1#2#3#4#5#6.#7;%
{%
    \expandafter\CALX_dv_sb\expandafter#1\expandafter#2\expandafter#3%
    \expandafter#4\the\numexpr#6-#5*#1#2#3#4+2000000000+#7{#5}%
}%
\def\CALX_dv_e #1{0;0.\CALX_dv_e;}%
\def\CALX_dv_sb #1#2#3#4#5#6#7#8#9{#5#6#7#8#9\CALX_dv_sc{#1#2#3#4}}%
\def\CALX_dv_sc #1#2#3#4#5#6{#2-200000;#3#4#5#6.\CALX_dv_c#1}%
\def\CALX_dv_ni #1#2#3#4#5{#1#2#3#4#5-20000;}%
\def\CALX_dv_ns #1#2#3#4#5{#1#2#3#4#5-20000\relax}%
\def\CALX_dv_nt #1;#2;#3#4#5% top, 4n digits, n, s, E (P follows)
{%
    \ifnum#1=\xint_c_ \xint_dothis{\CALX_fq_p{(\xint_c_iv*#3)}{}}\fi
    \xint_orthat{\CALX_fq_p{(\xint_c_iv*#3+\CALX_ndig{#1})}{#1}}%
    {#2}{#4}{(#5-\xint_c_iv*#3)}%
}%
```

31.9.3. \CALX_fdq

Syntax (use with \romannumeral0; P a \numexpr expression; X, Y f-expanded F, I or raw values; <k> one of \CALX_fdq_q, \CALX_fdq_r, \CALX_fdq_p):
  \CALX_fdq{P}\CALX_fdq_q{X}{Y} -> <space>Q
    \CALX_fdq{P}\CALX_fdq_r{X}{Y} -> <space>R
    \CALX_fdq{P}\CALX_fdq_p{X}{Y} -> {Q}{R}
Floor division of X' and Y', the values X and Y rounded to P digits (\CALX_fround): Q = floor(X'/Y') exact (an F value with e = 0, not rounded), R = X' - Y' Q rounded to P digits (its sign is that of Y'). They implement //, /: (and mod()), divmod() of \xintfloateval.

Both F and non-zero: with d = min(e_X, e_Y), the operand of larger block exponent gets that many zero blocks at its low end (\CALX_fa_z, as in \CALX_faddx); the two integers N<s><n>.<blocks> (n = ceil(D/8) plus the zero blocks) go to calxcore's \CALX_ndivfloor, \CALX_nmod or \CALX_ndivmod; Q is converted with e = 0 and R with e = d (\CALX_fx_c: digit count read on the top block). A zero operand or an I value: the integers without zero blocks (I values as they are, d = e_X if X is F), and the kernels apply their rules (Python's decimal model): x // 0 DivisionByZero (I+ or I-), 0 // 0 DivisionUndefined (I?), x mod 0 InvalidOperation (I?), Inf // Inf and Inf mod x InvalidOperation, x // Inf zero, x mod Inf x. An I result is passed on as it is.

```
\def\CALX_fdq #1#2#3%
   {\expandafter\CALX_fdq_a\romannumeral0\CALX_fround{#1}{#3}#2{#1}}%
\def\CALX_fdq_a #1\xint:#2#3#4%
   {\expandafter\CALX_fdq_b\romannumeral0\CALX_fround{#3}{#4}#1\xint:#2{#3}}%
\def\CALX_fdq_b #1%
{%
    \if I#1\xint_dothis{\CALX_fdq_bi #1}\fi
    \xint_orthat{\CALX_fdq_bf #1}%
}%
\def\CALX_fdq_bi #1#2\xint:#3{\CALX_fdq_s #1#2\xint:#3}%
\def\CALX_fdq_bf #1#2\xint:#3%
{%
    \if I#3\xint_dothis\CALX_fdq_s\fi
    \xint_orthat\CALX_fdq_f #1#2\xint:#3%
}%
\def\CALX_fdq_f F#1#2.#3.#4\xint:F#5#6.#7.#8\xint:#9%
{%
    \ifnum\numexpr#2*#6=\xint_c_
      \xint_dothis\CALX_fdq_s\fi
    \xint_orthat\CALX_fdq_fg
    F#1#2.#3.#4\xint:F#5#6.#7.#8\xint:#9%
}%
\def\CALX_fdq_fg F#1#2.#3.#4\xint:F#5#6.#7.#8\xint:#9%
{%
    \expandafter#9\the\numexpr\ifnum#3<#7 #3\else#7\fi
      \expandafter.\expanded
      {{N#5\the\numexpr(\xint_c_ii*#6+\xint_c_vii)/16\ifnum#7>#3 +#7-#3\fi.%
        \ifnum#7>#3 \CALX_fa_z#7-#3.\fi#8}%
       {N#1\the\numexpr(\xint_c_ii*#2+\xint_c_vii)/16\ifnum#3>#7 +#3-#7\fi.%
        \ifnum#3>#7 \CALX_fa_z#3-#7.\fi#4}}%
}%
\def\CALX_fdq_s #1#2\xint:#3#4\xint:#5%
{%
    \expandafter#5\the\numexpr\if I#3 \xint_c_\else\CALX_fdq_x#4\xint:\fi
    \expandafter.\expanded{{\CALX_fdq_n#3#4}{\CALX_fdq_n#1#2}}%
```

```
}%
\def\CALX_fdq_x #1#2.#3.#4\xint:{#3}%
\def\CALX_fdq_n #1{\if I#1\CALX_fdq_ni\fi\CALX_fdq_nf}%
\def\CALX_fdq_ni\fi\CALX_fdq_nf{\fi I}%
\def\CALX_fdq_nf #1#2.#3.{N#1\the\numexpr(\xint_c_ii*#2+\xint_c_vii)/16.}%
\def\CALX_fdq_q #1.#2#3#4%
   {\expandafter\CALX_fdq_t\expandafter0\expandafter.%
    \romannumeral0\CALX_ndivfloor{#2}{#3}\xint:}%
\def\CALX_fdq_r #1.#2#3#4%
   {\CALX_fnround{#4}{\expandafter\CALX_fdq_t\the\numexpr#1\expandafter.%
    \romannumeral0\CALX_nmod{#2}{#3}\xint:}}%
\def\CALX_fdq_p #1.#2#3%
   {\expandafter\CALX_fdq_pa\romannumeral0\CALX_ndivmod{#2}{#3}{#1}}%
\def\CALX_fdq_pa #1#2#3#4%
   {\expanded{{\romannumeral0\CALX_fdq_t 0.#1\xint:}%
    {\romannumeral0\CALX_fnround{#4}{\CALX_fdq_t #3.#2\xint:}}}}%
\def\CALX_fdq_t #1.#2{\if I#2\expandafter\CALX_fdq_i\fi\CALX_fx_c #1.#2}%
\def\CALX_fdq_i\CALX_fx_c #1.#2\xint:{ #2}%
```

31.9.4. \CALX_finv, \CALX_fe

Syntax (use with \romannumeral0; P a \numexpr expression; X f-expanded F, M, I or raw value; n a \numexpr expression):
  \CALX_finv{P}{X} -> <space>Y Y = round_P(1/X)
    \CALX_fe{P}{X}{n} -> <space>A[N] round_P(X) 10^n, raw (I unchanged)
\CALX_finv: one rounding of the exact inverse (not of the inverse of X rounded): X as raw A/B[N] (F by \CALX_ftoraw, M by \CALX_mtoraw_t), then \CALX_fround of the raw B/A[-N] (the sign moved to B; A = 0: the DivisionByZero of \CALX_fround's division path, I+). I+, I-: 0; I?: I?. The tag of X is made of catcode 12 by \string. inv() of \xintfloateval.
\CALX_fe: X rounded to P digits by \CALX_fround, then \CALX_ftoraw and n added to the exponent. The operator x[n] of \xintfloateval; X may be the bare digits of the integer literal before [n] (a raw value: [0] is appended, n added to its exponent, then rounded as F).

```
\def\CALX_finv #1#2{\expandafter\CALX_finv_s\romannumeral`&&@#2\xint:{#1}}%
\def\CALX_finv_in #1\xint:#2.{\CALX_finv_s #1\xint:{#2}}%
\def\CALX_finv_s{\expandafter\CALX_finv_a\string}%
\def\CALX_finv_a #1#2\xint:%
{%
    \if I#1\xint_dothis{\CALX_finv_i#2}\fi
    \if F#1\xint_dothis
      {\expandafter\CALX_finv_r\romannumeral0\CALX_ftoraw{#1#2}[0]\xint:}\fi
    \if M#1\xint_dothis
      {\expandafter\CALX_finv_r\romannumeral0\CALX_mtoraw_t#1#2\xint:%
       [0]\xint:}\fi
    \xint_orthat{\CALX_finv_r#1#2[0]\xint:}%
}%
\def\CALX_finv_i #1#2{\if ?#1\xint_dothis{ I?}\fi\xint_orthat{ F+0.0.}}%
\def\CALX_finv_r #1[#2]#3\xint:%
   {\expandafter\CALX_finv_t\the\numexpr-#2.#1/1/\xint:}%
\def\CALX_finv_t #1.#2#3/#4/#5\xint:#6%
{%
    \expandafter\CALX_fbr\romannumeral0\CALX_fround{#6}%
      {\expanded{\if-#2-#4/#3\else#4/#2#3\fi}[#1]}%
}%
```

```
\def\CALX_fe_in #1\xint:#2.{\CALX_fe_s #1\xint:{#2}}%
\def\CALX_fe #1#2%
   {\expandafter\CALX_fe_s\romannumeral`&&@#2\xint:{#1}}%
\def\CALX_fe_s #1%
{%
    \if I#1\xint_dothis\CALX_fe_i\fi
    \if F#1\xint_dothis\CALX_fe_n\fi
    \if M#1\xint_dothis\CALX_fe_n\fi
    \xint_orthat\CALX_fe_r #1%
}%
\def\CALX_fe_i #1\xint:#2#3{ #1}%
\def\CALX_fe_n #1\xint:#2%
   {\expandafter\CALX_fe_a\romannumeral0\CALX_fround{#2}{#1}}%
\def\CALX_fe_a #1\xint:%
   {\expandafter\CALX_fe_b\romannumeral0\CALX_ftoraw{#1}}%
\def\CALX_fe_b #1[#2]#3{\expandafter\CALX_fe_c\the\numexpr#2+#3.{#1}}%
\def\CALX_fe_c #1.#2{ #2[#1]}%
\def\CALX_fe_r #1\xint:{\CALX_fe_t #1[0]\xint:}%
\def\CALX_fe_t #1[#2]#3\xint:#4#5%
   {\expandafter\CALX_fe_u\the\numexpr#2+#5.{#1}{#4}}%
\def\CALX_fe_u #1.#2#3%
   {\expandafter\CALX_fe_a\romannumeral0\CALX_fround{#3}{#2[#1]}{0}}%
```

31.10. Square root

31.10.1. \CALX_fsqrt

Syntax (use with \romannumeral0; P a \numexpr expression; X f-expanded F, I or raw value):
  \CALX_fsqrt{P}{X} -> <space>Z Z = round_P(sqrt(round_P(X)))
\CALX_fsqrt{P}{X} (use with \romannumeral0, X f-expanded): round_P(sqrt(round_P(X))), round half away from zero, as xint's \XINTinFloatSqrt[P].
The operand is rounded by \CALX_fround. Special values: I? gives I?, I+ gives I+, I- and negative operands signal InvalidOperation (\CALX_nsig). Zero gives F+0.0.
  X = M 10^(8e), M of D digits (n blocks), o = 1 if D is odd, else 0.
  P <= 40 (\CALX_frt_ka...): K = max(2, ceil(P/8)), sh = 16K - D - o (even), Y0 = M 10^sh has 16K digits, the first one zero if o = 1, and sqrt(X) = sqrt(Y0) 10^(4e - sh/2). The top block v of Y0 (from the two top blocks of M, \CALX_frt_kt) is at least 10^6; Y = 4^j Y0 with j the least integer such that 4^j v >= 25000000 (j <= 3, no new block), as required by \CALX_nsqrt_core, which gives s = floor(sqrt(Y)) (8K digits) and r = Y - s^2. The blocks of Y are those of M 10^c 4^j, c = sh mod 8, below zb = floor(sh/8) zero blocks (\CALX_frt_mul: multiplication of a native by w < 2^31, base 10^4 limbs, w split as w1 10^4 + w0).
  Then sqrt(Y0) = s/2^j + f/2^j, 0 <= f < 1, and s/2^j = s 5^j 10^(-j) exactly. With E = 4e - sh/2 - j = 8a + b (0 <= b < 8), the native float s 5^j 10^b 10^(8a) (\CALX_frt_mul, \CALX_fm_gb) is rounded to P digits by \CALX_fround. This is the correctly rounded result:
  - if 8K > P, floor(sqrt(Y0)) = floor(s/2^j) has 8K >= P + 1 digits and the rounding bound is an integer, so the digits beyond it do not matter;
  - if 8K = P and j > 0, rounding up is right iff sqrt(Y0) >= S0 + 1/2 with S0 = floor(s/2^j), that is iff s mod 2^j >= 2^(j-1), that is iff the first digit of the fraction s/2^j - S0 (the first dropped digit) is at least 5;
  - if 8K = P and j = 0, rounding up is right iff r > s; s is then incremented (\CALX_frt_kh compares the block counts, then the top blocks, then uses \CALX_nsub).
  P > 40 (\CALX_frt_gb): with
     j = max(0, ceil((2P + 1 - D)/8)), plus one if j - e is odd,

the integer N = M 10^(8j) has at least 2P + 1 digits, its integer square root S (\CALX_nsqrtrem) has at least P + 1 digits, and sqrt(X) = sqrt(N) 10^(4(e-j)); the native float S 10^(8(e-j)/2) is rounded by \CALX_fround (\CALX_fm_gb), as in the first case.

```
\def\CALX_fsqrt #1#2%
   {\expandafter\CALX_frt_a\romannumeral0\CALX_fround{#1}{#2}{#1}}%
\def\CALX_frt_a #1#2%
   {\csname CALX_frt_\if I#1sp\else\if-#2ng\else b\fi\fi\endcsname#2}%
\def\CALX_frt_sp #1#2\xint:#3%
{%
    \if ?#1\xint_dothis{ I?}\fi
    \if +#1\xint_dothis{ I+}\fi
    \xint_orthat\CALX_frt_neg
}%
\def\CALX_frt_ng #1\xint:#2{\CALX_frt_neg}%
\def\CALX_frt_neg{\CALX_nsig{InvalidOperation}{sqrt of negative}{}}%
\def\CALX_frt_b #1#2.#3.#4\xint:#5%
{%
    \csname CALX_frt_\ifnum#2=\xint_c_ z\else\ifnum#5>40 gb\else kz\fi\fi
    \endcsname{#2}{#3}{#4}{#5}%
}%
\def\CALX_frt_z #1#2#3#4{ F+0.0.}%
\def\CALX_frt_kz #1#2#3#4%
{%
    \expandafter\CALX_frt_ka\the\numexpr
      \ifnum#4<9 \xint_c_ii\else(\xint_c_ii*#4+\xint_c_vii)/16\fi.%
      {#1}{#2}{#3}{#4}%
}%
\def\CALX_frt_ka #1.#2% K, D
   {\expandafter\CALX_frt_kb\the\numexpr16*#1-#2-\ifodd#2 1\else0\fi.%
       #1.{#2}}%
\def\CALX_frt_kb #1.%
   {\expandafter\CALX_frt_kc\the\numexpr(\xint_c_ii*#1-\xint_c_vii)/16.#1.}%
\def\CALX_frt_kc #1.#2.#3.#4#5#6% zb, sh, K, D, e, blocks
{%
    \expandafter\CALX_frt_kd\the\numexpr
      \CALX_nskip\CALX_frt_kt(\xint_c_ii*#4+\xint_c_vii)/16-\xint_c_i.%
        100000000!#6\xint:{\the\numexpr#2-\xint_c_viii*#1}.%
      #1.#2.#3.{#4}{#5}{#6}%
}%
\def\CALX_frt_kt 1#1!1#2!#3\xint:#4%
   {#2*\CALX_pten{#4}+(#1+\CALX_pten{\xint_c_vii-#4}*5)/%
       \CALX_pten{\xint_c_viii-#4}-\xint_c_i}%
\def\CALX_frt_kd #1.%
{%
    \expandafter\CALX_frt_ke\the\numexpr
      \ifnum#1<25000000 \ifnum#1<6250000 \ifnum#1<1562500 3\else2\fi
      \else1\fi\else0\fi.%
}%
\def\CALX_frt_ke #1.#2.#3.#4.#5#6#7% j, zb, sh, K, D, e, blocks
{%
    \expandafter\CALX_frt_kf\romannumeral0\CALX_frt_mul
      {\CALX_pten{#3-\xint_c_viii*#2}*\ifcase#1 1\or4\or16\or64\fi}0{N+.#7}%
```

```
    \xint:{#2}{#1}{#4}{#3}{#6}%
}%
\def\CALX_frt_kf N+#1.#2\xint:#3#4#5% product, zb, j, K
{%
    \expandafter\CALX_nsqrt_core\expandafter\CALX_frt_kg
      \expanded{{\the\numexpr\xint_c_ii*#5}{\CALX_fa_z#3.\unexpanded{#2}}}%
    {#4}{#5}%
}%
\def\CALX_frt_kg #1#2#3#4#5#6#7% s, r, j, K, sh, e, P
{%
    \ifnum\numexpr#3*100+\xint_c_viii*#4-#7=\xint_c_
      \xint_dothis{\CALX_frt_kh#2#1\CALX_frt_ki0{#1}{#3}%
        {\xint_c_iv*#6-#5/\xint_c_ii}{#7}}\fi
    \xint_orthat{\CALX_frt_ki0{#1}{#3}{\xint_c_iv*#6-#5/\xint_c_ii-#3}{#7}}%
}%
\def\CALX_frt_kh N+#1.#2N+#3.#4\CALX_frt_ki0%
{%
    \ifnum#1<#3 \xint_dothis{\CALX_frt_ki0}\fi
    \ifnum#1>#3 \xint_dothis{\CALX_frt_ki1}\fi
    \xint_orthat{\expandafter\CALX_frt_kj\the\numexpr
      \CALX_nskip\CALX_frt_kt #1-\xint_c_ii.#2\xint:{0}-%
      (\CALX_nskip\CALX_frt_kt #1-\xint_c_ii.#4\xint:{0})\relax.%
      N+#1.#2\xint:N+#3.#4\xint:}%
}%
\def\CALX_frt_kj #1.#2\xint:#3\xint:%
{%
    \ifnum#1<\xint_c_ \xint_dothis{\CALX_frt_ki0}\fi
    \ifnum#1>\xint_c_ \xint_dothis{\CALX_frt_ki1}\fi
    \xint_orthat{\expandafter\CALX_frt_kk\romannumeral0\CALX_nsub{#2}{#3}%
        \xint:}%
}%
\def\CALX_frt_kk N#1#2.#3\xint:%
{%
    \if-#1\xint_dothis{\CALX_frt_ki0}\fi
    \ifnum#2=\xint_c_ \xint_dothis{\CALX_frt_ki0}\fi
    \xint_orthat{\CALX_frt_ki1}%
}%
\def\CALX_frt_ki #1#2#3#4% d, s, j, E
   {\expandafter\CALX_frt_km\the\numexpr(\xint_c_ii*(#4)-\xint_c_vii)/16.%
       #1{#2}{#3}{#4}}%
\def\CALX_frt_km #1.#2#3#4#5% a, d, s, j, E (P follows)
{%
    \expandafter\CALX_fm_gb\romannumeral0\CALX_frt_mul
      {\ifcase#4 1\or5\or25\or125\fi*\CALX_pten{#5-\xint_c_viii*#1}}#2%
      {#3}\xint:{#1}%
}%
\def\CALX_frt_mul #1#2#3%
   {\expandafter\CALX_frt_mua\the\numexpr(#1+5000)/10000-\xint_c_i.%
       {#1}#2#3\xint:}%
\def\CALX_frt_mua #1.#2#3N+#4.#5\xint:%
   {\expanded{\noexpand\CALX_nnorm+%
       \expandafter\CALX_frt_mi\the\numexpr#2-#1*10000.#1.#3.#51;!}}%
```

```
\def\CALX_frt_mi #1.#2.#3.1#4!%
   {\expandafter\CALX_frt_mj\the\numexpr#4+#3.#1.#2.}%
\def\CALX_frt_mj #1.#2.#3.%
   {\expandafter\CALX_frt_mn\the\numexpr(#1+5000)/10000-\xint_c_i.%
       #1.#2.#3.0.0.}%
\def\CALX_frt_mm #1.#2.#3.#4.1#5#6!%
{%
    \xint_gob_til_sc#5\CALX_frt_mz;%
    \expandafter\CALX_frt_mn\the\numexpr(#5#6+5000)/10000-\xint_c_i.%
       #5#6.#1.#2.#3.#4.%
}%
\def\CALX_frt_mz;#1.#2.#3.#4.#5.#6.%
{%
    \the\numexpr#5*#4+#6-((#5*#4+#6+50000000)/\xint_c_e_viii-\xint_c_i)*%
       \xint_c_e_viii+\xint_c_e_viii!%
    \the\numexpr(#5*#4+#6+50000000)/\xint_c_e_viii-\xint_c_i+%
       \xint_c_e_viii!1;!%
}%
\def\CALX_frt_mn #1.#2.#3.#4.#5.#6.% gh, x, w0, w1, ph, c
   {\expandafter\CALX_frt_mo\the\numexpr(#2-#1*10000)*#3+#5*#4+#6.%
       #1.#2.#3.#4.}%
\def\CALX_frt_mo #1.#2.#3.#4.#5.% t0, gh, x, w0, w1
   {\expandafter\CALX_frt_mp\the\numexpr#2*#4+(#3-#2*10000)*#5+%
       (#1+5000)/10000-\xint_c_i.#1.#2.#4.#5.}%
\def\CALX_frt_mp #1.#2.#3.#4.#5.% t1, t0, gh, w0, w1
{%
    \the\numexpr(#1-((#1+5000)/10000-\xint_c_i)*10000)*10000+%
       #2-((#2+5000)/10000-\xint_c_i)*10000+\xint_c_e_viii!%
    \expandafter\CALX_frt_mq\the\numexpr(#1+5000)/10000-\xint_c_i.%
       #4.#5.#3.%
}%
\def\CALX_frt_mq #1.#2.#3.#4.{\CALX_frt_mm #2.#3.#4.#1.}%
\def\CALX_frt_gb #1#2#3#4%
{%
    \expandafter\CALX_frt_c\the\numexpr
      (\xint_c_ii*(\xint_c_ii*#4+\xint_c_i-#1)+\xint_c_vii)/16.%
      {#1}{#2}{#3}{#4}%
}%
\def\CALX_frt_c #1.#2#3%
{%
    \expandafter\CALX_frt_d\the\numexpr
      \ifnum#1<\xint_c_ 0\else#1\fi
      \ifodd\numexpr\ifnum#1<\xint_c_ 0\else#1\fi-#3\relax+\xint_c_i\fi.%
      {#2}{#3}%
}%
\def\CALX_frt_d #1.#2#3#4%
{%
    \expandafter\CALX_frt_e\romannumeral0\CALX_nsqrtrem
      {\expanded{N+\the\numexpr(\xint_c_ii*#2+\xint_c_vii)/16+#1.%
        \CALX_fa_z#1.}#4}%
    \xint:{(#3-#1)/\xint_c_ii}%
}%
```

```
\def\CALX_frt_e #1#2{\CALX_fm_gb #1}%
```

31.11. Factorials and binomial coefficients

Syntax (use with \romannumeral0; P, n, a, b, x, y \numexpr expressions with integer values):
  \CALX_ffac{P}{n} -> <space>Y n!
    \CALX_fpfac{P}{a}{b}   -> <space>Y   (a+1)(a+2)...b
    \CALX_fbinom{P}{x}{y}   -> <space>Y   binomial(x, y)
Y: the native F value rounded to P digits, or I? (InvalidOperation: n < 0 or n >= 10^8; a < b with b >= 10^8 or a <= -10^8; x < 0 or x >= 10^8). 0! = 1; pfactorial is 1 if a >= b, 0 if a < 0 <= b, and for a < b < 0 the product of the negative factors; binomial is 0 if y < 0 or y > x, 1 if y = 0 or y = x.

The engine is xintfrac's: the value is kept as Q blocks 1<8d>! (least significant first), terminated by 1;!, followed by [E], E a multiple of 8, and multiplied by products of 4, 3, 2 or 1 consecutive integers (each product < 10^8; \CALX_ffac_smallmul, own small multiplication which reports a carry). When a carry creates a block Q + 1 the least significant one is dropped and E increases by 8. \CALX_ffac_increaseP (\CALX_fpfac_increaseP) chooses Q from the target p and a factor c (their last two arguments) so that the relative error is below c 10^(-p-1): 8Q >= p + 9 + k, k the number of digits of the count of products divided by c (\CALX_ndig). The blocks are those of a native value: \CALX_ffac_out builds it (\CALX_nnorm, \CALX_fx_c) and rounds it once (\CALX_fnround).

Precisions. n! and pfactorial: p = P + 2, c = 1, relative error below 10^(-P-3), at most 0.001 ulp of the P-digit result. Binomial: pfac(y, x) / (x - y)! if y >= x - y, else pfac(x - y, x) / y!; numerator and denominator with p = P + 4, c = 4, each rounded to P + 5 digits, the quotient correctly rounded to P digits by \CALX_fr_xk (calxfp's division of natives): relative error below 2 (4 10^(-P-5) + 5 10^(-P-5)), less than 0.002 ulp.

```
\def\CALX_ffac #1#2%
   {\expandafter\CALX_ffac_fork_a\the\numexpr#2.\xint_c_i{#1+\xint_c_ii}%
    \CALX_ffac_out{#1}}%
\def\CALX_ffac_fork_a #1%
{%
     \xint_UDzerominusfork
     #1-\CALX_ffac_iszero
     0#1\CALX_ffac_isneg
      0-{\CALX_ffac_fork_b #1}%
    \krof
}%
\def\CALX_ffac_iszero #1.#2#3#4#5{ F+1.0.100000001!}%
\def\CALX_ffac_isneg #1.#2#3#4#5%
   {\CALX_nsig{InvalidOperation}{factorial of a negative integer}{}}%
\def\CALX_ffac_fork_b #1.%
{%
    \ifnum #1>\xint_c_e_viii_mone\xint_dothis\CALX_ffac_toobig\fi
    \ifnum #1>\xint_c_e_iv\xint_dothis\CALX_ffac_vbig \fi
    \ifnum #1>465  \xint_dothis\CALX_ffac_big\fi
    \ifnum #1>101  \xint_dothis\CALX_ffac_med\fi
                   \xint_orthat\CALX_ffac_small
    #1.%
}%
\def\CALX_ffac_toobig #1.#2#3#4#5%
   {\CALX_nsig{InvalidOperation}{factorial of an integer >= 10^8}{}}%
\def\CALX_ffac_vbig
```

```
2055    {\expandafter\CALX_ffac_vbigloop_a
2056     \the\numexpr \CALX_ffac_increaseP \xint_c_i   }%
2057 \def\CALX_ffac_big
2058    {\expandafter\CALX_ffac_bigloop_a
2059     \the\numexpr \CALX_ffac_increaseP \xint_c_ii  }%
2060 \def\CALX_ffac_med
2061    {\expandafter\CALX_ffac_medloop_a
2062     \the\numexpr \CALX_ffac_increaseP \xint_c_iii }%
2063 \def\CALX_ffac_small
2064    {\expandafter\CALX_ffac_smallloop_a
2065     \the\numexpr \CALX_ffac_increaseP \xint_c_iv  }%
2066 \def\CALX_ffac_increaseP #1#2.#3#4%
2067 {%
2068     #2\expandafter.\the\numexpr\xint_c_viii*%
2069     ((\xint_c_iv+#4+\CALX_ndig{#2/(#1*#3)})/\xint_c_viii).%
2070 }%
2071 \def\CALX_ffac_out #1;![#2]#3%
2072    {\CALX_fnround{#3}{\expandafter\CALX_fx_c\the\numexpr#2/\xint_c_viii
2073     \expandafter.\romannumeral0\CALX_nnorm +#1;!\xint:}}%
2074 \def\CALX_ffac_vbigloop_a #1.#2.%
2075 {%
2076     \CALX_ffac_bigloop_a \xint_c_e_iv.#2.%
2077     {\expandafter\CALX_ffac_vbigloop_loop\the\numexpr 100010001\expandafter.%
2078      \the\numexpr \xint_c_e_viii+#1.}%
2079 }%
2080 \def\CALX_ffac_vbigloop_loop #1.#2.%
2081 {%
2082     \ifnum #1>#2 \expandafter\CALX_ffac_loop_exit\fi
2083     \expandafter\CALX_ffac_vbigloop_loop
2084     \the\numexpr #1+\xint_c_i\expandafter.%
2085     \the\numexpr #2\expandafter.\the\numexpr\CALX_ffac_mul #1!%
2086 }%
2087 \def\CALX_ffac_bigloop_a #1.%
2088 {%
2089     \expandafter\CALX_ffac_bigloop_b \the\numexpr
2090     #1+\xint_c_i-\xint_c_ii*((#1-464)/\xint_c_ii).#1.%
2091 }%
2092 \def\CALX_ffac_bigloop_b #1.#2.#3.%
2093 {%
2094     \expandafter\CALX_ffac_medloop_a
2095         \the\numexpr #1-\xint_c_i.#3.{\CALX_ffac_bigloop_loop #1.#2.}%
2096 }%
2097 \def\CALX_ffac_bigloop_loop #1.#2.%
2098 {%
2099     \ifnum #1>#2 \expandafter\CALX_ffac_loop_exit\fi
2100     \expandafter\CALX_ffac_bigloop_loop
2101     \the\numexpr #1+\xint_c_ii\expandafter.%
2102     \the\numexpr #2\expandafter.\the\numexpr\CALX_ffac_bigloop_mul #1!%
2103 }%
2104 \def\CALX_ffac_bigloop_mul #1!%
2105 {%
2106     \expandafter\CALX_ffac_mul
```

```
        \the\numexpr \xint_c_e_viii+#1*(#1+\xint_c_i)!%
}%
\def\CALX_ffac_medloop_a #1.%
{%
    \expandafter\CALX_ffac_medloop_b
        \the\numexpr #1+\xint_c_i-\xint_c_iii*((#1-100)/\xint_c_iii).#1.%
}%
\def\CALX_ffac_medloop_b #1.#2.#3.%
{%
    \expandafter\CALX_ffac_smallloop_a
        \the\numexpr #1-\xint_c_i.#3.{\CALX_ffac_medloop_loop #1.#2.}%
}%
\def\CALX_ffac_medloop_loop #1.#2.%
{%
    \ifnum #1>#2 \expandafter\CALX_ffac_loop_exit\fi
    \expandafter\CALX_ffac_medloop_loop
    \the\numexpr #1+\xint_c_iii\expandafter.%
    \the\numexpr #2\expandafter.\the\numexpr\CALX_ffac_medloop_mul #1!%
}%
\def\CALX_ffac_medloop_mul #1!%
{%
    \expandafter\CALX_ffac_mul
    \the\numexpr
        \xint_c_e_viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
}%
\def\CALX_ffac_smallloop_a #1.%
{%
    \csname
       CALX_ffac_smallloop_\the\numexpr #1-\xint_c_iv*(#1/\xint_c_iv)\relax
    \endcsname #1.%
}%
\expandafter\def\csname CALX_ffac_smallloop_1\endcsname #1.#2.%
{%
    \CALX_ffac_addzeros #2.100000001!.{2.#1.}{#2}%
}%
\expandafter\def\csname CALX_ffac_smallloop_-2\endcsname #1.#2.%
{%
    \CALX_ffac_addzeros #2.100000002!.{3.#1.}{#2}%
}%
\expandafter\def\csname CALX_ffac_smallloop_-1\endcsname #1.#2.%
{%
    \CALX_ffac_addzeros #2.100000006!.{4.#1.}{#2}%
}%
\expandafter\def\csname CALX_ffac_smallloop_0\endcsname #1.#2.%
{%
    \CALX_ffac_addzeros #2.100000024!.{5.#1.}{#2}%
}%
\def\CALX_ffac_addzeros #1.%
{%
    \ifnum #1=\xint_c_viii \expandafter\CALX_ffac_addzeros_exit\fi
    \expandafter\CALX_ffac_addzeros
    \the\numexpr #1-\xint_c_viii.100000000!%
```

```
}%
\def\CALX_ffac_addzeros_exit #1.#2.#3#4%
   {\CALX_ffac_smallloop_loop #3#21;![-#4]}%
\def\CALX_ffac_smallloop_loop #1.#2.%
{%
    \ifnum #1>#2 \expandafter\CALX_ffac_loop_exit\fi
    \expandafter\CALX_ffac_smallloop_loop
    \the\numexpr #1+\xint_c_iv\expandafter.%
    \the\numexpr #2\expandafter.\romannumeral0\CALX_ffac_smallloop_mul #1!%
}%
\def\CALX_ffac_smallloop_mul #1!%
{%
    \expandafter\CALX_ffac_mul
    \the\numexpr
        \xint_c_e_viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
}%[[
\def\CALX_ffac_loop_exit #1!#2]#3{#3#2]}%
\def\CALX_ffac_mul 1#1!%
   {\expandafter\CALX_ffac_mul_a\the\numexpr\CALX_ffac_smallmul 10!{#1}}%
\def\CALX_ffac_mul_a #1-#2%
{%
    \if#21\xint_afterfi{\expandafter\space\xint_gob_til_exclam}\else
    \expandafter\space\fi #11;!%
}%
\def\CALX_ffac_minimulwc_a #1#2#3#4#5!#6#7#8#9%
{%
    \CALX_ffac_minimulwc_b {#1#2#3#4}{#5}{#6#7#8#9}%
}%
\def\CALX_ffac_minimulwc_b #1#2#3#4!#5%
{%
    \expandafter\CALX_ffac_minimulwc_c
    \the\numexpr \xint_c_e_ix+#5+#2*#4!{{#1}{#2}{#3}{#4}}%
}%
\def\CALX_ffac_minimulwc_c 1#1#2#3#4#5#6!#7%
{%
    \expandafter\CALX_ffac_minimulwc_d {#1#2#3#4#5}#7{#6}%
}%
\def\CALX_ffac_minimulwc_d #1#2#3#4#5%
{%
    \expandafter\CALX_ffac_minimulwc_e
    \the\numexpr \xint_c_e_ix+#1+#2*#5+#3*#4!{#2}{#4}%
}%
\def\CALX_ffac_minimulwc_e 1#1#2#3#4#5#6!#7#8#9%
{%
    1#6#9\expandafter!%
    \the\numexpr\expandafter\CALX_ffac_smallmul
    \the\numexpr \xint_c_e_viii+#1#2#3#4#5+#7*#8!%
}%
\def\CALX_ffac_smallmul 1#1!#21#3!%
{%
    \xint_gob_til_sc #3\CALX_ffac_smallmul_end;%
    \CALX_ffac_minimulwc_a #2!#3!{#1}{#2}%
```

```
}%
\def\CALX_ffac_smallmul_end;\CALX_ffac_minimulwc_a #1!;!#2#3[#4]%
{%
   \ifnum #2=\xint_c_
       \expandafter\xint_firstoftwo\else
       \expandafter\xint_secondoftwo
   \fi
   {-2\relax[#4]}%
   {1#2\expandafter!\expandafter-\expandafter1\expandafter
                  [\the\numexpr #4+\xint_c_viii]}%
}%
\def\CALX_fpfac #1#2#3%
{%
   \expandafter\CALX_fpfac_fork
   \the\numexpr #2\expandafter.\the\numexpr #3.\xint_c_i{#1+\xint_c_ii}{#1}%
}%
\def\CALX_fpfac_fork #1#2.#3#4.%
{%
    \unless\ifnum #1#2<#3#4 \xint_dothis\CALX_fpfac_one\fi
    \if-#3\xint_dothis\CALX_fpfac_neg \fi
    \if-#1\xint_dothis\CALX_fpfac_zero\fi
    \ifnum #3#4>\xint_c_e_viii_mone\xint_dothis\CALX_fpfac_outofrange\fi
    \xint_orthat \CALX_fpfac_increaseP #1#2.#3#4.%
}%
\def\CALX_fpfac_outofrange #1.#2.#3#4#5{\CALX_iipfac_inv #1.#2.}%
\def\CALX_fpfac_one  #1.#2.#3#4#5{ F+1.0.100000001!}%
\def\CALX_fpfac_zero #1.#2.#3#4#5{ F+0.0.}%
\def\CALX_fpfac_neg -#1.-#2.%
{%
    \ifnum #1>\xint_c_e_viii\xint_dothis\CALX_fpfac_outofrange\fi
    \xint_orthat {%
    \ifodd\numexpr#2-#1\relax
      \xint_afterfi{\expandafter\CALX_fpfac_m\romannumeral0}\fi
    \expandafter\CALX_fpfac_increaseP}%
    \the\numexpr #2-\xint_c_i\expandafter.\the\numexpr#1-\xint_c_i.%
}%
\def\CALX_fpfac_m F+{ F-}%
\def\CALX_fpfac_increaseP #1.#2.#3#4%
{%
    \expandafter\CALX_fpfac_a
    \the\numexpr \xint_c_viii*((\xint_c_iv+#4+\CALX_ndig{(#2-#1-\xint_c_i)%
      /\ifnum #2>\xint_c_e_iv #3\else(#3*\xint_c_ii)\fi})/\xint_c_viii).%
    #1.#2.%
}%
\def\CALX_fpfac_a #1.#2.#3.%
{%
    \expandafter\CALX_fpfac_b\the\numexpr \xint_c_i+#2\expandafter.%
    \the\numexpr#3\expandafter.%
    \romannumeral0\CALX_fpfac_addzeroes #1.100000001!1;![-#1]%
}%
\def\CALX_fpfac_addzeroes #1.%
{%
```

```
    \ifnum #1=\xint_c_viii \expandafter\CALX_fpfac_addzeroes_exit\fi
    \expandafter\CALX_fpfac_addzeroes\the\numexpr #1-\xint_c_viii.100000000!%
}%
\def\CALX_fpfac_addzeroes_exit #1.{ }%
\def\CALX_fpfac_b #1.%
{%
    \ifnum #1>9999 \xint_dothis\CALX_fpfac_vbigloop \fi
    \ifnum #1>463  \xint_dothis\CALX_fpfac_bigloop  \fi
    \ifnum #1>98   \xint_dothis\CALX_fpfac_medloop  \fi
                   \xint_orthat\CALX_fpfac_smallloop #1.%
}%
\def\CALX_fpfac_smallloop #1.#2.%
{%
    \ifcase\numexpr #2-#1\relax
        \expandafter\CALX_fpfac_end_
    \or \expandafter\CALX_fpfac_end_i
    \or \expandafter\CALX_fpfac_end_ii
    \or \expandafter\CALX_fpfac_end_iii
    \else\expandafter\CALX_fpfac_smallloop_a
    \fi #1.#2.%
}%
\def\CALX_fpfac_smallloop_a #1.#2.%
{%
    \expandafter\CALX_fpfac_smallloop_b
    \the\numexpr #1+\xint_c_iv\expandafter.%
    \the\numexpr #2\expandafter.%
    \romannumeral0\expandafter\CALX_ffac_mul
    \the\numexpr \xint_c_e_viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)*%
      (#1+\xint_c_iii)!%
}%
\def\CALX_fpfac_smallloop_b #1.%
{%
    \ifnum #1>98  \expandafter\CALX_fpfac_medloop   \else
                  \expandafter\CALX_fpfac_smallloop \fi #1.%
}%
\def\CALX_fpfac_medloop #1.#2.%
{%
    \ifcase\numexpr #2-#1\relax
        \expandafter\CALX_fpfac_end_
    \or \expandafter\CALX_fpfac_end_i
    \or \expandafter\CALX_fpfac_end_ii
    \else\expandafter\CALX_fpfac_medloop_a
    \fi #1.#2.%
}%
\def\CALX_fpfac_medloop_a #1.#2.%
{%
    \expandafter\CALX_fpfac_medloop_b
    \the\numexpr #1+\xint_c_iii\expandafter.%
    \the\numexpr #2\expandafter.%
    \romannumeral0\expandafter\CALX_ffac_mul
    \the\numexpr \xint_c_e_viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
}%
```

```
\def\CALX_fpfac_medloop_b #1.%
{%
    \ifnum #1>463 \expandafter\CALX_fpfac_bigloop   \else
                  \expandafter\CALX_fpfac_medloop   \fi #1.%
}%
\def\CALX_fpfac_bigloop #1.#2.%
{%
    \ifcase\numexpr #2-#1\relax
        \expandafter\CALX_fpfac_end_
    \or \expandafter\CALX_fpfac_end_i
    \else\expandafter\CALX_fpfac_bigloop_a
    \fi #1.#2.%
}%
\def\CALX_fpfac_bigloop_a #1.#2.%
{%
    \expandafter\CALX_fpfac_bigloop_b
    \the\numexpr #1+\xint_c_ii\expandafter.%
    \the\numexpr #2\expandafter.%
    \romannumeral0\expandafter\CALX_ffac_mul
    \the\numexpr \xint_c_e_viii+#1*(#1+\xint_c_i)!%
}%
\def\CALX_fpfac_bigloop_b #1.%
{%
    \ifnum #1>9999 \expandafter\CALX_fpfac_vbigloop  \else
                   \expandafter\CALX_fpfac_bigloop   \fi #1.%
}%
\def\CALX_fpfac_vbigloop #1.#2.%
{%
    \ifnum #2=#1
         \expandafter\CALX_fpfac_end_
    \else\expandafter\CALX_fpfac_vbigloop_a
    \fi #1.#2.%
}%
\def\CALX_fpfac_vbigloop_a #1.#2.%
{%
    \expandafter\CALX_fpfac_vbigloop
    \the\numexpr #1+\xint_c_i\expandafter.%
    \the\numexpr #2\expandafter.%
    \romannumeral0\expandafter\CALX_ffac_mul
    \the\numexpr\xint_c_e_viii+#1!%
}%
\def\CALX_fpfac_end_iii #1.#2.%
{%
    \expandafter\CALX_ffac_out
    \romannumeral0\expandafter\CALX_ffac_mul
    \the\numexpr \xint_c_e_viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)*%
      (#1+\xint_c_iii)!%
}%
\def\CALX_fpfac_end_ii #1.#2.%
{%
    \expandafter\CALX_ffac_out
    \romannumeral0\expandafter\CALX_ffac_mul
```

```
    \the\numexpr \xint_c_e_viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
}%
\def\CALX_fpfac_end_i #1.#2.%
{%
    \expandafter\CALX_ffac_out
    \romannumeral0\expandafter\CALX_ffac_mul
    \the\numexpr \xint_c_e_viii+#1*(#1+\xint_c_i)!%
}%
\def\CALX_fpfac_end_ #1.#2.%
{%
    \expandafter\CALX_ffac_out
    \romannumeral0\expandafter\CALX_ffac_mul
    \the\numexpr \xint_c_e_viii+#1!%
}%
\def\CALX_fbinom #1#2#3%
{%
    \expandafter\CALX_fbinom_a\the\numexpr#2\expandafter.%
    \the\numexpr#3\expandafter.\the\numexpr#1.%
}%
\def\CALX_fbinom_a #1.#2.%
{%
    \expandafter\CALX_fbinom_fork \the\numexpr #1-#2.#2.#1.%
}%
\def\CALX_fbinom_fork #1#2.#3#4.#5#6.%
{%
    \if-#5\xint_dothis \CALX_fbinom_inv\fi
    \if-#1\xint_dothis \CALX_fbinom_zero\fi
    \if-#3\xint_dothis \CALX_fbinom_zero\fi
    \if0#1\xint_dothis \CALX_fbinom_one\fi
    \if0#3\xint_dothis \CALX_fbinom_one\fi
    \ifnum #5#6>\xint_c_e_viii_mone \xint_dothis\CALX_fbinom_inv\fi
    \ifnum #1#2>#3#4  \xint_dothis\CALX_fbinom_ab \fi
                      \xint_orthat\CALX_fbinom_aa
    #1#2.#3#4.#5#6.%
}%
\def\CALX_fbinom_inv #1.#2.#3.#4.{\CALX_iibinom_inv}%
\def\CALX_fbinom_one  #1.#2.#3.#4.{ F+1.0.100000001!}%
\def\CALX_fbinom_zero #1.#2.#3.#4.{ F+0.0.}%
\def\CALX_fbinom_ab #1.#2.{\CALX_fbinom_aa #2.#1.}%
\def\CALX_fbinom_aa #1.#2.#3.#4.%
{%
    \expandafter\CALX_fbinom_q\romannumeral0\CALX_fpfac_increaseP
    #2.#3.\xint_c_iv{#4+\xint_c_iv}{#4+\xint_c_v}\xint:#1.{#4}%
}%
\def\CALX_fbinom_q #1\xint:#2.#3%
{%
    \expandafter\CALX_fbinom_r\romannumeral0\CALX_ffac_fork_b
    #2.\xint_c_iv{#3+\xint_c_iv}\CALX_ffac_out{#3+\xint_c_v}%
    \xint:#1\xint:{#3}%
}%
\def\CALX_fbinom_r #1\xint:#2\xint:{\CALX_fr_xk #2\xint:#1\xint:}%
```

31.12. Fixed-point multiply-subtract

\calx_txf<s><n> (s = 1..9, n = s or s+1 <= 9, written as letters; generated by tools/xfix_gen.py), used by the sin/cos series of calxtrig.sty, one call per level:

    \calx_txf<s><n>{u1}..{u4}{t1}..{ts}<U5..><cp><text><N>{w1}..{w4}
       gives <N>{w1}..{w4}{r1}..{rn}

Limbs of four digits, most significant first. T is the integer sum of the tj 10^(4(s-j)), U the fraction sum of the ui 10^(-4i), given as q = 4 ceil(s/4) limbs: <U5..> are u5 to uq, flat. C has n limbs ci: with ec = max(s-6, 0), <cp> is {cn}{c(n-1)}... for the columns e = 0..ec-1, <text> is +c(n-ec);+c(n-ec-1);...;+c1; with {} after the `;` of column 0 when ec = 0. <N> is one token, {w1}..{w4} one braced chunk put back in front of the result (the first limbs of U for the next level). R is C minus U T truncated, modulo 10^(4n): column e (weight 10^(4e)), e = n-1 down to -2, is

    2100000000 + c(n-e) - (sum of ui tj, i + j = s - e) + carry,

with c(n-e) only for e >= 0, i, j <= s, and also u(s+1) t1 in column -2 when s mod 4 > 0 (the terms of the former \calx_tm<s>_<n> of calxtrig: same results). The value has ten digits (s <= 11 products, see calxtrig); its first six minus 210000 are the carry, floor, into the next column; its last four are the limb, dropped for e < 0 and for the carry out of the top column.

Layout, after l3fp's __fp_fixed_mul:wwn: one nested \numexpr chain, innermost column evaluated first, written in two passes. Down pass: every column is opened from the top with its products u1..u4 times a sliding window of T. Up pass: the rest of column -2 (products with i >= 5), then `;`, the rest of column -1, `;`, and so on up to column ec, whose c and the following ones come from <text>. A column is thus continued after its inner columns: its pack \CALX_xq or \CALX_xp takes the next `;`-delimited text into the enclosing \numexpr and puts the limb behind it; \CALX_xp collects the limbs in one braced group (empty group after the `;` of column 0), \CALX_xe and \CALX_xg output <N>, the chunk that follows it, the top limb, the group's contents. Successive macros (suffixes x, y, z, w, v, u, r) take at most nine arguments: limbs passed on, the low limbs of T needed by the up pass in one braced group, the next items of the stream.

```
\def\CALX_xe #1#2#3#4#5#6#7;#8{\CALX_xg{{#7}#8}}%
\def\CALX_xg #1#2#3{#2#3#1}%
\def\CALX_xp #1#2#3#4#5#6#7;#8#9;{+#1#2#3#4#5#6-210000#9;{{#7}#8}}%
\def\CALX_xq #1#2#3#4#5#6#7;#8;{+#1#2#3#4#5#6-210000#8;}%
% generated by tools/xfix_gen.py
\def\CALX_xfab #1#2#3#4#5%
{%
\expandafter\CALX_xe\the\numexpr2100000000%
\expandafter\CALX_xp\the\numexpr2100000000%
\expandafter\CALX_xq\the\numexpr2100000000-#1*#5%
\expandafter\CALX_xq\the\numexpr2100000000-#2*#5;;%
}%
\def\CALX_xfbb #1#2#3#4#5#6%
{%
\expandafter\CALX_xe\the\numexpr2100000000%
\expandafter\CALX_xp\the\numexpr2100000000-#1*#5%
\expandafter\CALX_xq\the\numexpr2100000000-#2*#5-#1*#6%
\expandafter\CALX_xq\the\numexpr2100000000-#3*#5-#2*#6;;%
}%
\def\CALX_xfbc #1#2#3#4#5#6%
{%
\expandafter\CALX_xe\the\numexpr2100000000%
\expandafter\CALX_xp\the\numexpr2100000000%
\expandafter\CALX_xp\the\numexpr2100000000-#1*#5%
\expandafter\CALX_xq\the\numexpr2100000000-#2*#5-#1*#6%
```

```
\expandafter\CALX_xq\the\numexpr2100000000-#3*#5-#2*#6;;%
}%
\def\CALX_xfcc #1#2#3#4#5#6#7%
{%
\expandafter\CALX_xe\the\numexpr2100000000%
\expandafter\CALX_xp\the\numexpr2100000000-#1*#5%
\expandafter\CALX_xp\the\numexpr2100000000-#2*#5-#1*#6%
\expandafter\CALX_xq\the\numexpr2100000000-#3*#5-#2*#6-#1*#7%
\expandafter\CALX_xq\the\numexpr2100000000-#4*#5-#3*#6-#2*#7;;%
}%
\def\CALX_xfcd #1#2#3#4#5#6#7%
{%
\expandafter\CALX_xe\the\numexpr2100000000%
\expandafter\CALX_xp\the\numexpr2100000000%
\expandafter\CALX_xp\the\numexpr2100000000-#1*#5%
\expandafter\CALX_xp\the\numexpr2100000000-#2*#5-#1*#6%
\expandafter\CALX_xq\the\numexpr2100000000-#3*#5-#2*#6-#1*#7%
\expandafter\CALX_xq\the\numexpr2100000000-#4*#5-#3*#6-#2*#7;;%
}%
\def\CALX_xfdd #1#2#3#4#5#6#7#8%
{%
\expandafter\CALX_xe\the\numexpr2100000000%
\expandafter\CALX_xp\the\numexpr2100000000-#1*#5%
\expandafter\CALX_xp\the\numexpr2100000000-#2*#5-#1*#6%
\expandafter\CALX_xp\the\numexpr2100000000-#3*#5-#2*#6-#1*#7%
\expandafter\CALX_xq\the\numexpr2100000000-#4*#5-#3*#6-#2*#7-#1*#8%
\expandafter\CALX_xq\the\numexpr2100000000-#4*#6-#3*#7-#2*#8;;%
}%
\def\CALX_xfde #1#2#3#4#5#6#7#8%
{%
\expandafter\CALX_xe\the\numexpr2100000000%
\expandafter\CALX_xp\the\numexpr2100000000%
\expandafter\CALX_xp\the\numexpr2100000000-#1*#5%
\expandafter\CALX_xp\the\numexpr2100000000-#2*#5-#1*#6%
\expandafter\CALX_xp\the\numexpr2100000000-#3*#5-#2*#6-#1*#7%
\expandafter\CALX_xq\the\numexpr2100000000-#4*#5-#3*#6-#2*#7-#1*#8%
\expandafter\CALX_xq\the\numexpr2100000000-#4*#6-#3*#7-#2*#8;;%
}%
\def\CALX_xfee #1#2#3#4#5#6#7#8#9%
{%
\expandafter\CALX_xe\the\numexpr2100000000%
\expandafter\CALX_xp\the\numexpr2100000000-#1*#5%
\expandafter\CALX_xp\the\numexpr2100000000-#2*#5-#1*#6%
\expandafter\CALX_xp\the\numexpr2100000000-#3*#5-#2*#6-#1*#7%
\expandafter\CALX_xp\the\numexpr2100000000-#4*#5-#3*#6-#2*#7-#1*#8%
\expandafter\CALX_xq\the\numexpr2100000000-#4*#6-#3*#7-#2*#8-#1*#9%
\expandafter\CALX_xq\the\numexpr2100000000-#4*#7-#3*#8-#2*#9%
\CALX_xfeex{#5}{#6}%
}%
\def\CALX_xfeex #1#2#3#4#5#6%
{%
-#3*#2-#4*#1;-#3*#1;%
```

```
}%
\def\CALX_xfef #1#2#3#4#5#6#7#8#9%
{%
\expandafter\CALX_xe\the\numexpr2100000000%
\expandafter\CALX_xp\the\numexpr2100000000%
\expandafter\CALX_xp\the\numexpr2100000000-#1*#5%
\expandafter\CALX_xp\the\numexpr2100000000-#2*#5-#1*#6%
\expandafter\CALX_xp\the\numexpr2100000000-#3*#5-#2*#6-#1*#7%
\expandafter\CALX_xp\the\numexpr2100000000-#4*#5-#3*#6-#2*#7-#1*#8%
\expandafter\CALX_xq\the\numexpr2100000000-#4*#6-#3*#7-#2*#8-#1*#9%
\expandafter\CALX_xq\the\numexpr2100000000-#4*#7-#3*#8-#2*#9%
\CALX_xfefx{#5}{#6}%
}%
\def\CALX_xfefx #1#2#3#4#5#6%
{%
-#3*#2-#4*#1;-#3*#1;%
}%
\def\CALX_xfff #1#2#3#4#5#6#7#8#9%
{%
\expandafter\CALX_xe\the\numexpr2100000000%
\expandafter\CALX_xp\the\numexpr2100000000-#1*#5%
\expandafter\CALX_xp\the\numexpr2100000000-#2*#5-#1*#6%
\expandafter\CALX_xp\the\numexpr2100000000-#3*#5-#2*#6-#1*#7%
\expandafter\CALX_xp\the\numexpr2100000000-#4*#5-#3*#6-#2*#7-#1*#8%
\expandafter\CALX_xp\the\numexpr2100000000-#4*#6-#3*#7-#2*#8-#1*#9%
\expandafter\CALX_xq\the\numexpr2100000000-#4*#7-#3*#8-#2*#9%
\CALX_xfffx{#1}{#2}{#3}{#4}{#8}{#9}{{#5}{#6}{#7}}%
}%
\def\CALX_xfffx #1#2#3#4#5#6#7#8%
{%
-#1*#8%
\expandafter\CALX_xq\the\numexpr2100000000-#4*#5-#3*#6-#2*#8%
\CALX_xfffy#7%
}%
\def\CALX_xfffy #1#2#3#4#5#6#7%
{%
-#4*#3-#5*#2-#6*#1;-#4*#2-#5*#1;-#4*#1%
}%
\def\CALX_xffg #1#2#3#4#5#6#7#8#9%
{%
\expandafter\CALX_xe\the\numexpr2100000000%
\expandafter\CALX_xp\the\numexpr2100000000%
\expandafter\CALX_xp\the\numexpr2100000000-#1*#5%
\expandafter\CALX_xp\the\numexpr2100000000-#2*#5-#1*#6%
\expandafter\CALX_xp\the\numexpr2100000000-#3*#5-#2*#6-#1*#7%
\expandafter\CALX_xp\the\numexpr2100000000-#4*#5-#3*#6-#2*#7-#1*#8%
\expandafter\CALX_xp\the\numexpr2100000000-#4*#6-#3*#7-#2*#8-#1*#9%
\expandafter\CALX_xq\the\numexpr2100000000-#4*#7-#3*#8-#2*#9%
\CALX_xffgx{#1}{#2}{#3}{#4}{#8}{#9}{{#5}{#6}{#7}}%
}%
\def\CALX_xffgx #1#2#3#4#5#6#7#8%
{%
```

```
-#1*#8%
\expandafter\CALX_xq\the\numexpr2100000000-#4*#5-#3*#6-#2*#8%
\CALX_xffgy#7%
}%
\def\CALX_xffgy #1#2#3#4#5#6#7%
{%
-#4*#3-#5*#2-#6*#1;-#4*#2-#5*#1;-#4*#1%
}%
\def\CALX_xfgg #1#2#3#4#5#6#7#8#9%
{%
\expandafter\CALX_xe\the\numexpr2100000000%
\expandafter\CALX_xp\the\numexpr2100000000-#1*#5%
\expandafter\CALX_xp\the\numexpr2100000000-#2*#5-#1*#6%
\expandafter\CALX_xp\the\numexpr2100000000-#3*#5-#2*#6-#1*#7%
\expandafter\CALX_xp\the\numexpr2100000000-#4*#5-#3*#6-#2*#7-#1*#8%
\expandafter\CALX_xp\the\numexpr2100000000-#4*#6-#3*#7-#2*#8-#1*#9%
\expandafter\CALX_xp\the\numexpr2100000000-#4*#7-#3*#8-#2*#9%
\CALX_xfggx{#1}{#2}{#3}{#4}{#8}{#9}{{#5}{#6}{#7}}%
}%
\def\CALX_xfggx #1#2#3#4#5#6#7#8#9%
{%
-#1*#8%
\expandafter\CALX_xq\the\numexpr2100000000-#4*#5-#3*#6-#2*#8-#1*#9%
\expandafter\CALX_xq\the\numexpr2100000000-#4*#6-#3*#8-#2*#9%
\CALX_xfggy{#5}#7%
}%
\def\CALX_xfggy #1#2#3#4#5#6#7#8#9%
{%
-#5*#1-#6*#4-#7*#3-#8*#2;-#5*#4-#6*#3-#7*#2;-#5*#3-#6*#2+#9;{}-#5*#2%
}%
\def\CALX_xfgh #1#2#3#4#5#6#7#8#9%
{%
\expandafter\CALX_xe\the\numexpr2100000000%
\expandafter\CALX_xp\the\numexpr2100000000%
\expandafter\CALX_xp\the\numexpr2100000000-#1*#5%
\expandafter\CALX_xp\the\numexpr2100000000-#2*#5-#1*#6%
\expandafter\CALX_xp\the\numexpr2100000000-#3*#5-#2*#6-#1*#7%
\expandafter\CALX_xp\the\numexpr2100000000-#4*#5-#3*#6-#2*#7-#1*#8%
\expandafter\CALX_xp\the\numexpr2100000000-#4*#6-#3*#7-#2*#8-#1*#9%
\expandafter\CALX_xp\the\numexpr2100000000-#4*#7-#3*#8-#2*#9%
\CALX_xfghx{#1}{#2}{#3}{#4}{#8}{#9}{{#5}{#6}{#7}}%
}%
\def\CALX_xfghx #1#2#3#4#5#6#7#8#9%
{%
-#1*#8%
\expandafter\CALX_xq\the\numexpr2100000000-#4*#5-#3*#6-#2*#8-#1*#9%
\expandafter\CALX_xq\the\numexpr2100000000-#4*#6-#3*#8-#2*#9%
\CALX_xfghy{#5}#7%
}%
\def\CALX_xfghy #1#2#3#4#5#6#7#8#9%
{%
-#5*#1-#6*#4-#7*#3-#8*#2;-#5*#4-#6*#3-#7*#2;-#5*#3-#6*#2+#9;{}-#5*#2%
```

```
}%
\def\CALX_xfhh #1#2#3#4#5#6#7#8#9%
{%
\expandafter\CALX_xe\the\numexpr2100000000%
\expandafter\CALX_xp\the\numexpr2100000000-#1*#5%
\expandafter\CALX_xp\the\numexpr2100000000-#2*#5-#1*#6%
\expandafter\CALX_xp\the\numexpr2100000000-#3*#5-#2*#6-#1*#7%
\expandafter\CALX_xp\the\numexpr2100000000-#4*#5-#3*#6-#2*#7-#1*#8%
\expandafter\CALX_xp\the\numexpr2100000000-#4*#6-#3*#7-#2*#8-#1*#9%
\expandafter\CALX_xp\the\numexpr2100000000-#4*#7-#3*#8-#2*#9%
\CALX_xfhhx{#1}{#2}{#3}{#4}{#8}{#9}{{#5}{#6}{#7}}%
}%
\def\CALX_xfhhx #1#2#3#4#5#6#7#8#9%
{%
-#1*#8%
\expandafter\CALX_xp\the\numexpr2100000000-#4*#5-#3*#6-#2*#8-#1*#9%
\expandafter\CALX_xq\the\numexpr2100000000-#4*#6-#3*#8-#2*#9%
\CALX_xfhhy{#1}{#2}{#3}{#4}{#8}{#9}{{#5}{#6}#7}%
}%
\def\CALX_xfhhy #1#2#3#4#5#6#7#8%
{%
-#1*#8%
\expandafter\CALX_xq\the\numexpr2100000000-#4*#5-#3*#6-#2*#8%
\CALX_xfhhz#7%
}%
\def\CALX_xfhhz #1#2#3#4#5#6#7#8#9%
{%
-#6*#2-#7*#1-#8*#5-#9*#4;-#6*#1-#7*#5-#8*#4-#9*#3;-#6*#5-#7*#4-#8*#3%
\CALX_xfhhw{#3}{#4}{#6}{#7}%
}%
\def\CALX_xfhhw #1#2#3#4#5#6%
{%
+#5;{}-#3*#2-#4*#1+#6;-#3*#1%
}%
\def\CALX_xfhi #1#2#3#4#5#6#7#8#9%
{%
\expandafter\CALX_xe\the\numexpr2100000000%
\expandafter\CALX_xp\the\numexpr2100000000%
\expandafter\CALX_xp\the\numexpr2100000000-#1*#5%
\expandafter\CALX_xp\the\numexpr2100000000-#2*#5-#1*#6%
\expandafter\CALX_xp\the\numexpr2100000000-#3*#5-#2*#6-#1*#7%
\expandafter\CALX_xp\the\numexpr2100000000-#4*#5-#3*#6-#2*#7-#1*#8%
\expandafter\CALX_xp\the\numexpr2100000000-#4*#6-#3*#7-#2*#8-#1*#9%
\expandafter\CALX_xp\the\numexpr2100000000-#4*#7-#3*#8-#2*#9%
\CALX_xfhix{#1}{#2}{#3}{#4}{#8}{#9}{{#5}{#6}{#7}}%
}%
\def\CALX_xfhix #1#2#3#4#5#6#7#8#9%
{%
-#1*#8%
\expandafter\CALX_xp\the\numexpr2100000000-#4*#5-#3*#6-#2*#8-#1*#9%
\expandafter\CALX_xq\the\numexpr2100000000-#4*#6-#3*#8-#2*#9%
\CALX_xfhiy{#1}{#2}{#3}{#4}{#8}{#9}{{#5}{#6}#7}%
```

```
}%
\def\CALX_xfhiy #1#2#3#4#5#6#7#8%
{%
-#1*#8%
\expandafter\CALX_xq\the\numexpr2100000000-#4*#5-#3*#6-#2*#8%
\CALX_xfhiz#7%
}%
\def\CALX_xfhiz #1#2#3#4#5#6#7#8#9%
{%
-#6*#2-#7*#1-#8*#5-#9*#4;-#6*#1-#7*#5-#8*#4-#9*#3;-#6*#5-#7*#4-#8*#3%
\CALX_xfhiw{#3}{#4}{#6}{#7}%
}%
\def\CALX_xfhiw #1#2#3#4#5#6%
{%
+#5;{}-#3*#2-#4*#1+#6;-#3*#1%
}%
\def\CALX_xfii #1#2#3#4#5#6#7#8#9%
{%
\expandafter\CALX_xe\the\numexpr2100000000%
\expandafter\CALX_xp\the\numexpr2100000000-#1*#5%
\expandafter\CALX_xp\the\numexpr2100000000-#2*#5-#1*#6%
\expandafter\CALX_xp\the\numexpr2100000000-#3*#5-#2*#6-#1*#7%
\expandafter\CALX_xp\the\numexpr2100000000-#4*#5-#3*#6-#2*#7-#1*#8%
\expandafter\CALX_xp\the\numexpr2100000000-#4*#6-#3*#7-#2*#8-#1*#9%
\expandafter\CALX_xp\the\numexpr2100000000-#4*#7-#3*#8-#2*#9%
\CALX_xfiix{#1}{#2}{#3}{#4}{#8}{#9}{{#5}{#6}{#7}}%
}%
\def\CALX_xfiix #1#2#3#4#5#6#7#8#9%
{%
-#1*#8%
\expandafter\CALX_xp\the\numexpr2100000000-#4*#5-#3*#6-#2*#8-#1*#9%
\expandafter\CALX_xp\the\numexpr2100000000-#4*#6-#3*#8-#2*#9%
\CALX_xfiiy{#1}{#2}{#3}{#4}{#8}{#9}{{#5}{#6}#7}%
}%
\def\CALX_xfiiy #1#2#3#4#5#6#7#8#9%
{%
-#1*#8%
\expandafter\CALX_xq\the\numexpr2100000000-#4*#5-#3*#6-#2*#8-#1*#9%
\expandafter\CALX_xq\the\numexpr2100000000-#4*#6-#3*#8-#2*#9%
\CALX_xfiiz{#5}{#7}%
}%
\def\CALX_xfiiz #1#2#3#4#5#6%
{%
-#3*#1%
\CALX_xfiiw{#4}{#5}{#6}#2{{#3}}%
}%
\def\CALX_xfiiw #1#2#3#4#5#6#7#8#9%
{%
-#1*#5-#2*#4-#3*#8%
\CALX_xfiiv{#5}{#6}{#7}#9{{#1}{#2}{#3}{#4}{#8}}%
}%
\def\CALX_xfiiv #1#2#3#4#5#6#7#8#9%
```

```
{%
-#6*#3-#7*#2;-#4*#1%
\CALX_xfiiu{#2}{#3}{#4}{#6}#5%
}%
\def\CALX_xfiiu #1#2#3#4#5#6#7#8#9%
{%
-#5*#8-#6*#9-#7*#2-#4*#1;-#3*#8-#5*#9-#6*#2-#7*#1%
\CALX_xfiir{#1}{#2}{#3}{#5}{#6}{#9}%
}%
\def\CALX_xfiir #1#2#3#4#5#6#7#8#9%
{%
+#7;{}-#3*#6-#4*#2-#5*#1+#8;-#3*#2-#4*#1+#9;-#3*#1%
}%
\catcode`F=11 \catcode`N=11 \catcode`I=11
```

31.13. \calxFpFloat, \calxFpFloatdigits, \calxFpRoundTo, \calxFpTruncTo

Syntax (f-expandable; P, n \numexpr expressions, X an F, I or xint raw value, f-expanded; \calxFpRoundTo and \calxFpTruncTo accept M values too):

```
  \calxFpFloat[P]{X} -> Y
    \calxFpFloatdigits{X}    -> Y    (P = \XINTdigits)
    \calxFpRoundTo{n}{X}     -> Y    the multiple of 10^-n nearest to X
    \calxFpTruncTo{n}{X}     -> Y    the multiple of 10^-n between 0 and X
```

\calxFpFloat, \calxFpFloatdigits: X rounded to P significant digits by \CALX_fround (F or I value). They implement float(), float_dgt(), sfloat() of \xintfloateval. \calxFpRoundTo, \calxFpTruncTo: exact F value (ties away from zero), I unchanged, by \CALX_rt_a; they implement round(x, n) and trunc(x, n). The \xint: after Y is removed by \CALX_fbr.

```
\def\CALX_fbr #1\xint:{ #1}%
\def\calxFpFloatdigits
   {\romannumeral0\expandafter\CALX_fbr\romannumeral0\CALX_fround\XINTdigits}%
\def\calxFpFloat [#1]%
   {\romannumeral0\expandafter\CALX_fbr\romannumeral0\CALX_fround{#1}}%
\def\calxFpRoundTo #1#2%
   {\romannumeral0\expandafter\CALX_fbr\romannumeral0\expandafter
    \CALX_rt_a\expandafter\CALX_fr_b\the\numexpr#1\expandafter.%
    \romannumeral`&&@#2\xint:}%
\def\calxFpTruncTo #1#2%
   {\romannumeral0\expandafter\CALX_fbr\romannumeral0\expandafter
    \CALX_rt_a\expandafter\CALX_rt_t\the\numexpr#1\expandafter.%
    \romannumeral`&&@#2\xint:}%
```

31.14. \calxFpFac, \calxFpFacdigits, \calxFpPFactorial, \calxFpBinomial

Syntax (f-expandable; X, A, B parser values, f-expanded; P a \numexpr expression):

```
  \calxFpFacdigits{X} -> Y X! (P = \XINTdigits)
    \calxFpFac[P]{X}          -> Y   X!
    \calxFpPFactorial{A}{B}   -> Y   (A+1)(A+2)...B
    \calxFpBinomial{X}{A}     -> Y   binomial(X, A)
```

factorial(), !, pfactorial() and binomial() of \xintfloateval. The arguments are truncated to integers by \CALX_fnum (an I value signals InvalidOperation and counts as 0); Y: native F rounded to P digits, or I? (\CALX_ffac, \CALX_fpfac, \CALX_fbinom).

```
\def\CALX_fpint #1%
```

```
    {\romannumeral0\expandafter\CALX_fnum\romannumeral`&&@#1\xint:}%
\def\calxFpFacdigits #1{\romannumeral0\CALX_ffac\XINTdigits{\CALX_fpint{#1}}}%
\def\calxFpFac [#1]#2{\romannumeral0\CALX_ffac{#1}{\CALX_fpint{#2}}}%
\def\calxFpPFactorial #1#2%
    {\romannumeral0\CALX_fpfac\XINTdigits{\CALX_fpint{#1}}{\CALX_fpint{#2}}}%
\def\calxFpBinomial #1#2%
    {\romannumeral0\CALX_fbinom\XINTdigits{\CALX_fpint{#1}}{\CALX_fpint{#2}}}%
```

31.15. (WIP) \XINTinRandomFloatS, \XINTinRandomFloatSdigits

Added at 1.3b (2018/05/18). Support for random() function.

Thus as it is a priori only for xintexpr usage, it expands inside \csname context, but as we need to get rid of initial zeros we use \xintRandomDigits not \xintXRandomDigits (\expanded would have a use case here).

And anyway as we want to be able to use random() in \xintdeffunc/\xintNewExpr, it is good to have f-expandable macros, so we add the small overhead to make it f-expandable.

We don't have to be very efficient in removing leading zeroes, as there is only 10% chance for each successive one. Besides we use (current) internal storage format of the type A[N], where A is not required to be with \xintDigits digits, so N will simply be -\xintDigits and needs no adjustment.

In case we use in future with #1 something else than \xintDigits we do the 0-(#1) construct.

I had some qualms about doing a random float like this which means that when there are leading zeros in the random digits the (virtual) mantissa ends up with trailing zeros. That did not feel right but I checked random() in Python (which of course uses radix 2), and indeed this is what happens there.

```
\def\XINTinRandomFloatS{\romannumeral0\XINTinrandomfloatS}%
\def\XINTinRandomFloatSdigits{\XINTinRandomFloatS[\XINTdigits]}%
\def\XINTinrandomfloatS[#1]%
{%
    \expandafter\XINT_inrandomfloatS\the\numexpr\xint_c_-(#1)\xint:
}%
\def\XINT_inrandomfloatS-#1\xint:
{%
    \expandafter\XINT_inrandomfloatS_a
    \romannumeral0\xintrandomdigits{#1}[-#1]%
}%
```

We add one macro to handle a tiny bit faster 90% of cases, after all we also use one extra macro for the completely improbable all 0 case.

```
\def\XINT_inrandomfloatS_a#1%
{%
    \if#10\xint_dothis{\XINT_inrandomfloatS_b}\fi
    \xint_orthat{ #1}%
}%[
\def\XINT_inrandomfloatS_b#1%
{%
    \if#1[\xint_dothis{\XINT_inrandomfloatS_zero}\fi% ]
    \if#10\xint_dothis{\XINT_inrandomfloatS_b}\fi
    \xint_orthat{ #1}%
}%[
\def\XINT_inrandomfloatS_zero#1]{ 0[0]}%
```

31.16. (WIP) \XINTinRandomFloatSixteen

Added at 1.3b (2018/05/18). Support for qrand() function.

```
\def\XINTinRandomFloatSixteen%
{%
    \romannumeral0\expandafter\XINT_inrandomfloatS_a
    \romannumeral`&&@\expandafter\XINT_eightrandomdigits
                \romannumeral`&&@\XINT_eightrandomdigits[-16]%
}%
```

calx: random() and qrand() of \xintfloateval push native F values, the raw A[N] of the two macros above converted once by \CALX_ffromraw.

```
\def\calxFpRandomSdigits
   {\romannumeral0\CALX_ffromraw{\XINTinRandomFloatSdigits}}%
\def\calxFpRandomSixteen
   {\romannumeral0\CALX_ffromraw{\XINTinRandomFloatSixteen}}%
```

31.17. \XINT_pfloat_fork and the \xintPFloat hooks

Moved verbatim from xintfrac.sty at 2.0a1: the formatting core of \xintPFloat, used by the \xintfloateval printer (calx.sty's \CALX_fpr_a, fed by \CALX_frez) and by xintfrac's \xintPFloat (fed by \xintrez).

```
\def\xintPFloatE{e}%
\def\xintPFloatNoSciEmax{\xint_c_v}%   1e6  uses sci.not.
\def\xintPFloatNoSciEmin{-\xint_c_iv}% 1e-5 uses sci.not.
\def\xintPFloatIntSuffix{}%
\def\xintPFloatLengthOneSuffix{}%
\def\xintPFloatZero{0}%
\def\xintPFloatMinTrimmed{\xint_c_iv}%
\def\XINT_pfloat_fork#1%
{%
    \xint_UDzerominusfork
       #1-\XINT_pfloat_zero
       0#1\XINT_pfloat_neg
        0-\XINT_pfloat_pos
    \krof #1%
}%
\def\XINT_pfloat_zero#1]#2.{\expanded{ \xintPFloatZero}}%
\def\XINT_pfloat_neg-{\expandafter-\romannumeral0\XINT_pfloat_pos}%
\def\XINT_pfloat_pos#1/1[#2]#3.%
{%
    \expandafter\XINT_pfloat_aa\the\numexpr\xintLength{#1}.%
    #3.#2.#1.%
}%
```

```
   #1 est la longueur de la mantisse trimmée
   #2 est Digits ou P
   Si #2-#1 < MinTrimmed, on se prépare à peut-être remettre les trailing zeros
```

On teste pour #2=#1, car c'est le cas le plus fréquent (mais est-ce une bonne idée) car on sait qu'alors il n'y a pas de trailing zéros donc on va direct vers \XINT_pfloat_a.

```
\def\XINT_pfloat_aa #1.#2.%
{%
    \unless\ifnum\xintPFloatMinTrimmed>\numexpr#2-#1\relax
           \xint_dothis\XINT_pfloat_a\fi
```

```
    \ifnum#2>#1 \xint_dothis{\XINT_pfloat_i #2.}\fi
    \xint_orthat\XINT_pfloat_a #1.%
}%
```

Needed for \xintFracToSci, which uses old pre 1.4k interface, where the P parameter was not stored for counting how many zeros were trimmed. \xintFracToSci trims always.

```
\def\XINT_pfloat_a_fork#1%
{%
    \xint_UDzerominusfork
       #1-\XINT_pfloat_a_zero
       0#1\XINT_pfloat_a_neg
        0-\XINT_pfloat_a_pos
    \krof #1%
}%
\def\XINT_pfloat_a_zero#1]{\expanded{ \xintPFloatZero}}%
\def\XINT_pfloat_a_neg-{\expandafter-\romannumeral0\XINT_pfloat_a_pos}%
\def\XINT_pfloat_a_pos#1/1[#2]%
{%
    \expandafter\XINT_pfloat_a\the\numexpr\xintLength{#1}.#2.#1.%
}%
```

```
   #1 est P > #2 mais peut être encore sous la forme \XINTdigits
   #2 est la longueur de la mantisse trimmée
   #3 est l'exposant non normalisé
   #4 est la mantisse
```

On reconstitue les trailing zéros à remettre éventuellement.

```
\def\XINT_pfloat_i #1.#2.%#3.#4.%
{%
    \expandafter\XINT_pfloat_j\romannumeral\xintreplicate{#1-#2}0.#2.%
}%
```

```
   #1 est les trailing zeros à remettre peut-être
   #2 est la longueur de la mantisse trimmée
   #3#4 est l'exposant N pour mantisse trimmée entière
   #5 serait la mantisse trimmée
```

On calcule l'exposant scientifique.

La façon bizarre de mettre #3 est liée aux versions anciennes de la macro, héritage conservé pour minimiser effort d'adaptation.

```
\def\XINT_pfloat_j #1.#2.#3#4.%#5.
{%
    \expandafter\XINT_pfloat_b\the\numexpr#2+#3#4-\xint_c_i.%
    #3#2.#1.%
}%
```

```
   #1 est la longueur de la mantisse trimmée
   #2#3 est l'exposant N pour mantisse trimmée
   #4 serait la mantisse
```

On calcule l'exposant scientifique. On est arrivé ici dans une branche où on n'a pas besoin de remettre les zéros trimmés donc on positionne un dernier argument vide pour \XINT_pfloat_b

```
\def\XINT_pfloat_a #1.#2#3.%#4.
{%
    \expandafter\XINT_pfloat_b\the\numexpr#1+#2#3-\xint_c_i.%
    #2#1..%
}%
```

```
   #1 est l'exposant scientifique K
   #2 est le signe ou premier chiffre de l'exposant N pour mantisse trimmée
   #3 serait la longueur de la mantisse trimmée
   #4 serait les trailing zéros
   #5 serait la mantisse trimmée
```

On va vers \XINT_float_P lorsque l'on n'utilise pas la notation scientifique, mais qu'on a besoin de chiffres non nuls fractionnaires, et vers \XINT_float_Ps si on n'en a pas besoin.

On va vers \XINT_pfloat_N lorsque l'on n'utilise pas la notation scientifique et que l'exposant scientifique était strictement négatif.

```
\def\XINT_pfloat_b #1.#2%#3.#4.#5.
{%
    \ifnum \xintPFloatNoSciEmax<#1 \xint_dothis\XINT_pfloat_sci\fi
    \ifnum \xintPFloatNoSciEmin>#1 \xint_dothis\XINT_pfloat_sci\fi
    \ifnum #1<\xint_c_   \xint_dothis\XINT_pfloat_N\fi
    \if-#2\xint_dothis\XINT_pfloat_P\fi
    \xint_orthat\XINT_pfloat_Ps
    #1.%
}%
```

#1 is the scientific exponent, #2 is the length of the trimmed mantissa, #3 are the trailing zeros, #4 is the trimmed integer mantissa

\xintPFloatE can be replaced by any f-expandable macro with a dot-delimited argument which produces a dot-delimited output.

```
\def\XINT_pfloat_sci #1.#2.%
{%
    \ifnum#2=\xint_c_i\expandafter\XINT_pfloat_sci_i\expandafter\fi
    \expandafter\XINT_pfloat_sci_a\romannumeral`&&@\xintPFloatE #1.%
}%
\def\XINT_pfloat_sci_a #1.#2.#3#4.{ #3.#4#2#1}%
```

#1#2=\fi\XINT_pfloat_sci_a

1-digit mantissa, hesitation between d.0eK or deK Finally at 1.4k, \xintPFloatLengthOneSuffix for customization.

```
\def\XINT_pfloat_sci_i #1#2#3.#4.#5.{\expanded{#1 #5\xintPFloatLengthOneSuffix}#3}%
```

#1=sci.exp. K, #2=mant. wd L, #3=trailing zéros, #4=trimmed mantissa

For _N, #1 is at most -1, for _P, #1 is at least 0. For _P there will be fractional digits, and #1+1 digits before the mark.

```
\def\XINT_pfloat_N#1.#2.#3.#4.%
{%
    \expandafter\XINT_pfloat_N_e\romannumeral\xintreplicate{-#1}{0}#4#3%
}%
\def\XINT_pfloat_N_e 0{ 0.}%
```

#1=sci.exp. K, #2=mant. wd L, #3=trailing zéros, #4=trimmed mantissa

Abusive usage of internal \XINT_split_fromleft_a. It means using x = -1 - #1 in \xintDecSplit from xint.sty. We benefit also with the way \xintDecSplit is built upon \XINT_split_fromleft with a final clean-up which here we can shortcut via using terminator "\xint_bye." not "\xint_bye.."

```
\def\XINT_pfloat_P #1.#2.#3.#4.%
{%
    \expandafter\XINT_split_fromleft_a
    \the\numexpr\xint_c_vii-#1.#4\xint_bye2345678\xint_bye.#3%
}%
```

Here we have an integer so we only need to postfix the trimmed mantissa #4 with #1+1-#2 zeros (#1=sci exp., #2=trimmed mantissa width). Less cumbersome to do that with \expanded. And the trailing zeros #3 ignored here.

```
\def\XINT_pfloat_Ps #1.#2.#3.#4.%
{%
    \expanded{ #4%
    \romannumeral\xintreplicate{#1+\xint_c_i-#2}{0}\xintPFloatIntSuffix}%
}%
```

calx: \CALX_frez F<s><D>.<e>.<blocks>\xint: (use with \romannumeral0; the tag token is a parameter): <space><s>A/1[N] with A without trailing zeros (the form \xintrez gives), 0/1[0] for zero (first digit of D zero). Zero low blocks are dropped (\CALX_frez_b, e + 1 each); the t trailing zeros of the lowest nonzero block b (\CALX_frez_t, t <= 7) are removed: one block, A = b/10^t; else A = the digits of the higher blocks (\CALX_ntoraw_b) followed by the 8 - t leading digits of b; N = 8e + t.

```
\def\CALX_frez #1#2#3#4.#5.%
   {\xint_gob_til_zero #3\CALX_frez_z 0\CALX_frez_a #2#5.}%
\def\CALX_frez_z 0\CALX_frez_a #1\xint:{ 0/1[0]}%
\def\CALX_frez_a #1#2.1#3!%
   {\xint_gob_til_eightzeroes #3\CALX_frez_b 00000000\CALX_frez_c #1#2.#3.}%
\def\CALX_frez_b 00000000\CALX_frez_c #1#2.#3.%
   {\expandafter\CALX_frez_a\expandafter#1\the\numexpr#2+\xint_c_i.}%
\def\CALX_frez_c #1#2.#3.%
   {\expandafter\CALX_frez_d\the\numexpr\CALX_frez_t#3.#1#2.#3.}%
\def\CALX_frez_t #1#2#3#4#5#6#7#8%
   {\if0#8\if0#7\if0#6\if0#5\if0#4\if0#3\if0#2 7\else6\fi\else5\fi\else4\fi
    \else3\fi\else2\fi\else1\fi\else0\fi}%
\def\CALX_frez_p #1%
   {\ifcase#1 1\or10\or100\or1000\or10000\or100000\or1000000\or10000000\fi}%
\def\CALX_frez_d #1.#2#3.#4.#5\xint:%
{%
    \expanded{ \if-#2-\fi
      \if\relax\detokenize{#5}\relax
        \the\numexpr#4/\CALX_frez_p#1\relax
      \else
        \romannumeral0\CALX_ntoraw_b #5\xint:%
        \expandafter\xint_gobble_i\the\numexpr1#4/\CALX_frez_p#1\relax
      \fi
      /1[\the\numexpr\xint_c_viii*#3+#1\relax]}%
}%
\XINTrestorecatcodesendinput%
```

32. Package calx implementation

Contents

32.1 Introduction . . . 702
32.2 READ ME! Important warnings and explanations relative to the status of the code source at the time of the 1.4 release . . . 702
32.3 Old comments . . . 704
32.4 Catcodes, ε-TEX and reload detection . . . 704
32.5 Package identification . . . 706
32.6 \XINTfstop . . . 706
32.7 Support for output and transform of nested braced contents as core data type . . . 706
32.7.1 Bracketed list rendering with prettifying of leaves from nested braced contents . . . 706
32.7.2 Flattening nested braced contents . . . 707
32.7.3 Braced contents rendering via a TEX alignment with prettifying of leaves . . . 708
32.7.4 Transforming all leaves within nested braced contents . . . 709
32.8 Top level user TEX interface: \xintfloateval, \xintiieval . . . 710
32.8.1 \xintfloatexpr, \xintiiexpr . . . 710
32.8.2 \XINT_iiexpr_wrap, \XINT_flexpr_wrap . . . 711
32.8.3 \XINTiiexprprint, \XINTflexprprint . . . 712
32.8.4 \xintthe, \xintthealign, \xintthefloatexpr, \xinttheiiexpr . . . 713
32.8.5 \xintbarefloateval, \xintbareiieval . . . 713
32.8.6 \xintthebarefloateval, \xintthebareiieval . . . 714
32.8.7 \xintfloateval, \xintiieval . . . 714
32.8.8 \xintifboolfloatexpr, \xintifbooliiexpr . . . 715
32.8.9 \xintifNonFinitefloatexpr . . . 716
32.8.10 \xintifsgnfloatexpr, \xintifsgniiexpr . . . 716
32.8.11 Small bits we have to put somewhere . . . 716
\xintthecoords . . . 717
\xintthespaceseparated . . . 717
32.9 Hooks into the numeric parser for usage by the \xintdeffunc symbolic parser . . . 718
32.10 \XINT_expr_getnext: fetch some value then an operator and present them to last waiter with the found operator precedence, then the operator, then the value . . . 719
32.11 \XINT_expr_startint . . . 723
32.11.1 Integral part (skipping zeroes) . . . 725
32.11.2 Fractional part . . . 726
32.11.3 Scientific notation . . . 728
32.11.4 Hexadecimal numbers . . . 729
32.11.5 Octal numbers . . . 732
32.11.6 Binary numbers . . . 734
32.11.7 \XINT_expr_startfunc: collecting names of functions and variables . . . 736
32.11.8 \XINT_expr_func: dispatch to variable replacement or to function execution . . . 737
32.12 \XINT_expr_op_`: launch function or pseudo-function, or evaluate variable and insert operator of multiplication in front of parenthesized contents . . . 737
32.13 \XINT_expr_op__: replace a variable by its value and then fetch next operator . . . 738
32.14 \XINT_expr_getop: fetch the next operator or closing parenthesis or end of expression . . . 740
32.15 Expansion spanning; opening and closing parentheses . . . 742
32.16 The comma as binary operator . . . 744
32.17 The minus as prefix operator of variable precedence level . . . 745
32.18 The * as Python-like «unpacking» prefix operator . . . 746
32.19 Infix operators . . . 747
32.19.1 &&, ||, //, /:, +, −, *, /, ^, **, 'and', 'or', 'xor', and 'mod' . . . 747
32.19.2 .., ..[, and].. for a..b and a..[b]..c syntax . . . 749

32.19.3 <, >, ==, <=, >=, != with Python-like chaining . . . 751
32.19.4 Support macros for .., ..[and].. . . . 753
\xintSeq:tl:x, \xintfloatSeq:tl:x . . . 753
32.20 Square brackets [] both as a container and a Python slicer . . . 754
32.20.1 [...] as «oneple» constructor . . . 755
32.20.2 [...] brackets and : operator for NumPy-like slicing and item indexing syntax . . . 756
32.20.3 Macro layer implementing indexing and slicing . . . 758
32.21 Support for raw A/B[N] . . . 762
32.22 ? as two-way and ?? as three-way «short-circuit» conditionals . . . 763
32.23 ! as postfix factorial operator . . . 764
32.24 User defined variables . . . 764
32.24.1 \xintdefiivar, \xintdeffloatvar . . . 764
32.24.2 \xintunassignvar . . . 768
32.25 Support for dummy variables . . . 769
32.25.1 \xintnewdummy . . . 769
32.25.2 \xintensuredummy, \xintrestorevariable . . . 771
32.25.3 Checking (without expansion) that a symbolic expression contains correctly nested parentheses . . . 771
32.25.4 Fetching balanced expressions E1, E2 and a variable name Name from E1, Name=E2) . 772
32.25.5 Fetching a balanced expression delimited by a semi-colon . . . 772
32.25.6 Low-level support for omit and abort keywords, the break() function, the n++ construct and the semi-colon as used in the syntax of seq(), add(), mul(), iter(), rseq(), iterr(), rrseq(), subsm(), subsn(), ndseq(), ndmap() . . . 773
The n++ construct . . . 773
The break() function . . . 773
The omit and abort keywords . . . 773
The semi-colon . . . 774
32.25.7 Reserved dummy variables @, @1, @2, @3, @4, @@, @@(1), ..., @@@, @@@(1), ... for recursions . . . 774
32.26 Pseudo-functions involving dummy variables and generating scalars or sequences . . . 775
32.26.1 Comments . . . 776
32.26.2 subs(): substitution of one variable . . . 777
32.26.3 subsm(): simultaneous independent substitutions . . . 778
32.26.4 subsn(): leaner syntax for nesting (possibly dependent) substitutions . . . 780
32.26.5 seq(): sequences from assigning values to a dummy variable . . . 781
32.26.6 iter() . . . 782
32.26.7 add(), mul() . . . 784
32.26.8 rseq() . . . 785
32.26.9 iterr() . . . 786
32.26.10 rrseq() . . . 787
32.27 Pseudo-functions related to N-dimensional hypercubic lists . . . 788
32.27.1 ndseq() . . . 788
32.27.2 ndmap() . . . 790
32.27.3 ndfillraw() . . . 791
32.27.4 Making them known to the other parsers . . . 792
32.28 Other pseudo-functions: bool(), togl(), protect(), qraw(), qint(), qfrac(), qfloat(), qrand(), random(), rbit() . . . 792

32.29 Regular built-in functions: num(), reduce(), preduce(), abs(), sgn(), frac(), floor(), ceil(), sqr(), ?(), !(), not(), odd(), even(), isint(), isone(), factorial(), sqrt(), sqrtr(), inv(), round(), trunc(), float(), sfloat(), ilog10(), divmod(), mod(), binomial(), pfactorial(), randrange(), iquo(), irem(), gcd(), lcm(), max(), min(), `+`(), `*`(), all(), any(), xor(), len(), first(), last(), reversed(), if(), ifint(), ifone(), ifsgn(), nuple(), unpack(), flat() and zip() 794
32.30 User declared functions 806
32.30.1 \xintdeffunc, \xintdefiifunc, \xintdeffloatfunc 806
32.30.2 \xintdefufunc, \xintdefiiufunc, \xintdeffloatufunc 809
32.30.3 \xintunassignexprfunc, \xintunassigniiexprfunc, \xintunassignfloatexprfunc . 811
32.30.4 \xintNewFunction 811
32.30.5 Mysterious stuff 812
32.30.6 \XINT_expr_redefinemacros 824
32.30.7 Still more mysterious 826
32.30.8 \xintexprSafeCatcodes, \xintexprRestoreCatcodes 828
32.31 The calx layer 829
32.32 Floating point powers with integer exponent 842
32.33 Loading libraries 846
32.34 \xintreloadcalxlog, \xintreloadcalxtrig, \xintreloadscilibs 847
32.35 \calxeval, \calxinteval 847

32.1. Introduction

This is release 2.0a2 of 2026/11/26. THIS FILE IS AN EVOLUTION OF A PART OF FORMER xintexpr .sty. The latter now loads this file as a dependency.

32.2. READ ME! Important warnings and explanations relative to the status of the code source at the time of the 1.4 release

At release 1.4 the csname encapsulation of intermediate evaluations during parsing of expressions is dropped, and xintexpr requires the \expanded primitive. This means that there is no more impact on the string pool. And as internal storage now uses simply core \TeX{} syntax with braces rather than comma separated items inside a csname dummy control sequence, it became much easier to let the [...] syntax be associated to a true internal type of «tuple» or «list».

The output of \xintexpr (after \romannumeral0 or \romannumeral-`0 triggered expansion or double expansion) is thus modified at 1.4. It now looks like this:[78]

\XINTfstop \XINTexprprint .{{<number>}} in simplest case
\XINTfstop \XINTexprprint .{{...}...{...}} in general case

where ... stands for nested braces ultimately ending in {<num. rep.>} leaves. The <num. rep.> stands for some internal representation of numeric data. It may be empty, and currently as well as probably in future uses only catcode 12 tokens (no spaces currently).

{{}} corresponds (in input as in output) to []. The external TeX braces also serve as set-theoretical braces. The comma is concatenation, so for example [], [] will become {{}{}}, or rather {}{} if sub-unit of something else.

The associated vocabulary is explained in the user manual and we avoid too much duplication here. xintfrac numerical macros receiving an empty argument usually handle it as being 0, but this is not the case of the xintcore macros supporting \xintiiexpr, they usually break if exercised on some empty argument.

[78] The \XINTfstop was a simple \def\XINTfstop{\noexpand\XINTfstop} at 1.4 but has been converted into a fancier token at 1.5, to avoid the "blinking" effect when submitted to finitely many expansions à la \expandafter as first token of a replacement text.

The above expansion result \XINTfstop \XINTexprprint .{{<num1>}{<num2}...} uses only normal catcodes: the backslash, regular braces, ascii letters and catcode 12 characters. Scientific notation is internally converted to raw xintfrac representation [N].

Additional data may be located before the dot; this is the case only for \xintfloatexpr currently. As xintexpr actually defines three parsers \xintexpr, \xintiiexpr and \xintfloatexpr but tries to share as much code as possible, some overhead is induced to fit all into the same mold.

\XINTfstop stops \romannumeral-`0 (or 0) type spanned expansion, and is invariant under \edef, but simply disappears in typesetting context.[79] It is thus now legal to use \xintexpr directly in typesetting flow.

\XINTexprprint is \protected.

The f-expansion of an \xintexpr <expression>\relax is a complete expansion, i.e. one whose result remains invariant under \edef.

\xintthe\xintexpr <expression>\relax or \xinteval{<expression>} serve as formerly to deliver the explicit digits, or more exactly some prettifying view of the actual <internal number representation>.

Nested contents like this

```
{{1}{{2}{3}{{4}{5}{6}}}{9}}
```

will get delivered using nested square brackets like that

```
1, [2, 3, [4, 5, 6]], 9
```

and as conversely \xintexpr 1, [2, 3, [4, 5, 6]], 9\relax expands to

```
\XINTfstop \XINTexprprint .{{1}{{2}{3}{{4}{5}{6}}}{9}}
```

we obtain the gratifying result that

```
\xinteval{1, [2, 3, [4, 5, 6]], 9}
```

expands to

```
1, [2, 3, [4, 5, 6]], 9
```

See user manual for explanations on the plasticity of \xintexpr syntax regarding functions with multiple arguments, and the 1.4 «unpacking» Python-like * prefix operator.

I have suppressed (from the public dtx) many big chunks of comments. Some became obsolete and need to be updated, others are currently of value only to the author as a historical record.

ATTENTION! As the removal process itself took too much time, I ended up leaving as is many comments which are obsoleted and wrong to various degrees after the 1.4 release. Precedence levels of operators have all been doubled to make room for new constructs

Even comments added during 1.4 developement may now be obsolete because the preparation of 1.4 took a few weeks and that's enough of duration to provide the author many chances to contradict in the code what has been already commented upon.

Thus don't believe (fully) anything which is said here!

Warning: in text below and also in left-over old comments I may refer to «until» and «op» macros; due to the change of data storage at 1.4, I needed to refactor a bit the way expansion is controlled, and the situation now is mainly governed by «op», «exec», «check-» and «checkp» macros the latter three replacing the two «until_a» and «until_b» of former code. This allows to diminish the number of times an accumulated result will be grabbed in order to propagate expansion to its right. Formerly this was not an issue because such things were only a single token! I do not describe here how this is all articulated but it is not hard to see it from the code (the hardest thing in all such matter was in 2013 to actually write how the expansion would be initially launched because to do that one basically has to understand the mechanism in its whole and such things are not easy to develop piecemeal). Another thing to keep in mind is that operators in truth have a left precedence (i.e. the precedence they show to operators arising earlier) and a right precedence (which determines how they react to operators coming after them from the right). Only the

[79] It is not \let to \relax as it must be distinguished from it in \ifx tests.

first one is usually encapsulated in a chardef, the second one is most of the times identical to the first one and if not it is only virtual but implemented via \ifcase of \ifnum branching. A final remark is that some things are achieved by special «op» macros, which are a favorite tool to hack into the normal regular flow of things, via injection of special syntax elements. I did not rename these macros for avoiding too large git diffs, and besides the nice thing is that the 1.4 refactoring minimally had to modify them, and all hacky things using them kept on working with not a single modification. And a post-scriptum is that advanced features crucially exploit injecting sub-\xintexpr-essions, as all is expandable there is no real «context» (only a minimal one) which one would have to perhaps store and restore and doing this sub-expression injection is rather cheap and efficient operation.

32.3. Old comments

These general comments were last updated at the end of the 1.09x series in 2014. The principles remain in place to this day but refer to CHANGES.html for some significant evolutions since.

The first version was released in June 2013. I was greatly helped in this task of writing an expandable parser of infix operations by the comments provided in l3fp-parse.dtx (in its version as available in April-May 2013). One will recognize in particular the idea of the `until' macros; I have not looked into the actual l3fp code beyond the very useful comments provided in its documentation.

A main worry was that my data has no a priori bound on its size; to keep the code reasonably efficient, I experimented with a technique of storing and retrieving data expandably as *names* of control sequences. Intermediate computation results are stored as control sequences \.=a/b[n].

Roughly speaking, the parser mechanism is as follows: at any given time the last found ``operator'' has its associated until macro awaiting some news from the token flow; first getnext expands forward in the hope to construct some number, which may come from a parenthesized sub-expression, from some braced material, or from a digit by digit scan. After this number has been formed the next operator is looked for by the getop macro. Once getop has finished its job, until is presented with three tokens: the first one is the precedence level of the new found operator (which may be an end of expression marker), the second is the operator character token (earlier versions had here already some macro name, but in order to keep as much common code to expr and floatexpr common as possible, this was modified) of the new found operator, and the third one is the newly found number (which was encountered just before the new operator).

The until macro of the earlier operator examines the precedence level of the new found one, and either executes the earlier operator (in the case of a binary operation, with the found number and a previously stored one) or it delays execution, giving the hand to the until macro of the operator having been found of higher precedence.

A minus sign acting as prefix gets converted into a (unary) operator inheriting the precedence level of the previous operator.

Once the end of the expression is found (it has to be marked by a \relax) the final result is output as four tokens (five tokens since 1.09j) the first one a catcode 11 exclamation mark, the second one an error generating macro, the third one is a protection mechanism, the fourth one a printing macro and the fifth is \.=a/b[n]. The prefix \xintthe makes the output printable by killing the first three tokens.

32.4. Catcodes, ε-TEX and reload detection

The code for reload detection was initially copied from Heiko Oberdiek's packages, then modified.

The method for catcodes was also initially directly inspired by these packages.

```
\begingroup\catcode61\catcode48\catcode32=10\relax%
  \catcode13=5    % ^^M
  \endlinechar=13 %
  \catcode123=1   % {
```

```
  \catcode125=2   % }
  \catcode64=11   % @
  \catcode44=12   % ,
  \catcode46=12   % .
  \catcode58=12   % :
  \catcode94=7    % ^
  \catcode35=6    % #
  \def\odef#1{\expandafter\def\expandafter#1\expandafter}%
  \def\empty{}\def\space{ }\newlinechar10
  \def\z{\endgroup}%
  \expandafter\let\expandafter\x\csname ver@calx.sty\endcsname
  \expandafter\let\expandafter\t\csname ver@xinttools.sty\endcsname
  \expandafter\let\expandafter\b\csname ver@xintbinhex.sty\endcsname
  \expandafter\let\expandafter\f\csname ver@xintfrac.sty\endcsname
  \expandafter\let\expandafter\fp\csname ver@calxfp.sty\endcsname
  % I assume engines do not exist providing \expanded but not \numexpr
  \expandafter\ifx\csname expanded\endcsname\relax
    \expandafter\ifx\csname PackageWarningNoLine\endcsname\relax
      \immediate\write128{^^JPackage calx Warning:^^J%
                          \space\space\space\space
                          \expanded not available, aborting input.^^J}%
    \else
      \PackageWarningNoLine{calx}{\expanded not available, aborting input}%
    \fi
    \def\z{\endgroup\endinput}%
  \else
    \ifx\x\relax   % not LaTeX (perhaps Plain+miniltx), first loading of calx.sty
      \ifx\t\relax % but xinttools.sty not yet loaded.
         \odef\z{\z\input xinttools.sty\relax}%
      \fi
      \ifx\b\relax % but xintbinhex.sty not yet loaded.
         \odef\z{\z\input xintbinhex.sty\relax}%
      \fi
      \ifx\fp\relax % but calxfp.sty not yet loaded.
         \odef\z{\z\input calxfp.sty\relax}%
      \fi
    \else
      \ifx\x\empty % LaTeX, first loading,
      % variable is initialized, but \ProvidesPackage not yet seen
          \ifx\t\relax % xinttools not yet loaded.
            \odef\z{\z\RequirePackage{xinttools}}%
          \fi
          \ifx\b\relax % xintbinhex not yet loaded.
            \odef\z{\z\RequirePackage{xintbinhex}}%
          \fi
          \ifx\fp\relax % calxfp not yet loaded.
            \odef\z{\z\RequirePackage{calxfp}}%
          \fi
      \else
        \def\z{\endgroup\endinput}% xintexpr already loaded.
      \fi
    \fi
```

```
  \fi
\z%
\XINTsetupcatcodes%
```

32.5. Package identification

The ! is of catcode LETTER for most of the duration of the package. Prior to 1.4 it was the first token in the complete expansion of \xintexpr...\relax (i.e. without \xintthe prefix). It has since lost to \XINTfstop this prominent rôle, but its usage in various places of the code as delimiter in otherwise catcode 12 contexts has stuck.

```
\XINT_providespackage
\ProvidesPackage{calx}%
  [2026/11/26 v2.0a2 fp, bigint, and symbolic expression parsers (JFB)]%
\catcode`! 11
```

calx: ^^B, ^^C are the fake operators of the integer, resp. decimal and exponent, exits of the number scanner; catcode 12, restored at the end of the file.

```
\edef\CALX_restorecc{\catcode2=\the\catcode2\relax
    \catcode3=\the\catcode3\relax}\catcode2=12 \catcode3=12
```

32.6. \XINTfstop

Added at 1.4 (2020/01/31).
This replaced in 2019 the legacy usage of a catcode 11 exclamation mark to signal the start of a sub-expression (it was followed by some other tokens).

From 2022 (aiming at 1.5) to 2025 (releasing 1.4n) I had replaced the definition used here with a fancier one originating in a ``fake'' \relax from unravel. I think in 2022, unravel used a frozen let to \the, which I had transferred here as a definition of \XINTfstop, now reverted. When I checked again unravel in 2025, I saw a definition of the type

```
\expandafter\let\expandafter\foo\noexpand\foo
```

If I did that with \foo being \XINTfstop then \XINTfstop would test equal via \ifx to the unravel or any such \foo (if \foo is at first undefined)...

The inconvenience with our more naive \XINTfstop here, is that expanding a finite number of times \xintexpr...\relax depends on parity of number of expansions. Not really a problem as far as I can tell. The advantage is that \ifx will never confuse it with something else.

```
\def\XINTfstop{\noexpand\XINTfstop}%
```

32.7. Support for output and transform of nested braced contents as core data type

New at 1.4, of course. The former \csname.=...\endcsname encapsulation technique made very difficult implementation of nested structures.

32.7.1. Bracketed list rendering with prettifying of leaves from nested braced contents

Added at 1.4 (2020/01/31). The braces in \XINT:expr:toblistwith are there because there is an \expanded trigger.

Modified at 1.4d (2021/03/29). Add support for the polexpr 0.8 polynomial type. See \XINT:expr:toblist_a.

Modified at 1.4l (2022/05/29). Let \XINT:expr:toblist_b use #1{#2} with regular { and } in case macro #1 is \protected and things are output to external file where former #1<#2> would end up with catcode 12 < and >.

```
\def\XINT:expr:toblistwith#1#2%
{%
    {\expandafter\XINT:expr:toblist_checkempty
     \expanded{\noexpand#1!\expandafter}\detokenize{#2}^}%
}%
\def\XINT:expr:toblist_checkempty #1!#2%
{%
    \if ^#2\expandafter\xint_gob_til_^\else\expandafter\XINT:expr:toblist_a\fi
    #1!#2%
}%
\catcode`< 1 \catcode`> 2 \catcode`{ 12 \catcode`} 12
\def\XINT:expr:toblist_a #1{#2%
<%
    \if{#2\xint_dothis<[\XINT:expr:toblist_a>\fi
    \if P#2\xint_dothis<\XINT:expr:toblist_pol>\fi
    \xint_orthat\XINT:expr:toblist_b #1#2%
>%
\def\XINT:expr:toblist_pol #1!#2.{#3}}%
<%
    pol([\XINT:expr:toblist_b #1!#3}^])\XINT:expr:toblist_c #1!}%
>%
\catcode`{ 1 \catcode`} 2
\def\XINT:expr:toblist_b #1{%
\def\XINT:expr:toblist_b ##1!##2#1%
{%
    \if\relax##2\relax
    \xintexprEmptyItem\else##1{##2}\fi\XINT:expr:toblist_c ##1!#1%
}}\catcode`{ 12 \catcode`} 12 \XINT:expr:toblist_b<}>%
\def\XINT:expr:toblist_c #1}#2%
<%
    \if ^#2\xint_dothis<\xint_gob_til_^>\fi
    \if{#2\xint_dothis<, \XINT:expr:toblist_a>\fi
    \xint_orthat<]\XINT:expr:toblist_c>#1#2%
>%
\catcode`{ 1 \catcode`} 2 \catcode`< 12 \catcode`> 12
```

32.7.2. Flattening nested braced contents

1.4b I hesitated whether using this technique or some variation of the method of the ListSel macros. I chose this one which I downscaled from toblistwith, I will revisit later. I only have a few minutes right now.

Call form is \expanded\XINT:expr:flatten

See \XINT_expr_func_flat. I hesitated with «flattened», but short names are faster parsed.

```
\def\XINT:expr:flatten#1%
{%
   {{\expandafter\XINT:expr:flatten_checkempty\detokenize{#1}^}}%
}%
\def\XINT:expr:flatten_checkempty #1%
{%
    \if ^#1\expandafter\xint_gobble_i\else\expandafter\XINT:expr:flatten_a\fi
    #1%
}%
\begingroup
```

```
\catcode`[ 1 \catcode`] 2 \lccode`[`{ \lccode`]`}
\catcode`< 1 \catcode`> 2 \catcode`{ 12 \catcode`} 12
\lowercase<\endgroup
\def\XINT:expr:flatten_a {#1%
<%
    \if{#1\xint_dothis<\XINT:expr:flatten_a>\fi
    \xint_orthat\XINT:expr:flatten_b #1%
>%
\def\XINT:expr:flatten_b #1}%
<%
    [#1]\XINT:expr:flatten_c }%
>%
\def\XINT:expr:flatten_c }#1%
<%
    \if ^#1\xint_dothis<\xint_gobble_i>\fi
    \if{#1\xint_dothis<\XINT:expr:flatten_a>\fi
    \xint_orthat<\XINT:expr:flatten_c>#1%
>%
>% back to normal catcodes
```

32.7.3. Braced contents rendering via a TEX alignment with prettifying of leaves

1.4.

Breaking change at 1.4a as helper macros were renamed and their meanings refactored: no more \xintexpraligntab nor \xintexpraligninnercomma or \xintexpralignoutercomma but \xintexpraligninnersep, etc...

At 1.4c I remove the \protected from \xintexpralignend. I had made note a year ago that it served nothing. Let's trust myself on this one (risky one year later!).

```
\catcode`& 4
\protected\def\xintexpralignbegin
  {\halign\bgroup\tabskip2ex\hfil##&&##\hfil\cr}%
\def\xintexpralignend                   {\crcr\egroup}%
\protected\def\xintexpralignlinesep     {,\cr}%
\protected\def\xintexpralignleftbracket {[}%
\protected\def\xintexpralignrightbracket{]}%
\protected\def\xintexpralignleftsep     {&}%
\protected\def\xintexpralignrightsep    {&}%
\protected\def\xintexpraligninnersep    {,&}%
\catcode`& 7
\def\XINT:expr:toalignwith#1#2%
{%
    {\expandafter\XINT:expr:toalign_checkempty
     \expanded{\noexpand#1!\expandafter}\detokenize{#2}^\expandafter}%
    \xintexpralignend
}%
\def\XINT:expr:toalign_checkempty #1!#2%
{%
    \if ^#2\expandafter\xint_gob_til_^\else\expandafter\XINT:expr:toalign_a\fi
    #1!#2%
}%
\catcode`< 1 \catcode`> 2 \catcode`{ 12 \catcode`} 12
\def\XINT:expr:toalign_a #1{#2%
<%
```

```
    \if{#2\xint_dothis<\xintexpralignleftbracket\XINT:expr:toalign_a>\fi
    \xint_orthat<\xintexpralignleftsep\XINT:expr:toalign_b>#1#2%
>%
\def\XINT:expr:toalign_b #1!#2}%
<%
    \if\relax#2\relax\xintexprEmptyItem\else#1<#2>\fi\XINT:expr:toalign_c #1!}%
>%
\def\XINT:expr:toalign_c #1}#2%
<%
    \if ^#2\xint_dothis<\xint_gob_til_^>\fi
    \if {#2\xint_dothis<\xintexpraligninnersep\XINT:expr:toalign_A>\fi
    \xint_orthat<\xintexpralignrightsep\xintexpralignrightbracket
    \XINT:expr:toalign_C>#1#2%
>%
\def\XINT:expr:toalign_A #1{#2%
<%
    \if{#2\xint_dothis<\xintexpralignleftbracket\XINT:expr:toalign_A>\fi
    \xint_orthat\XINT:expr:toalign_b #1#2%
>%
\def\XINT:expr:toalign_C #1}#2%
<%
    \if ^#2\xint_dothis<\xint_gob_til_^>\fi
    \if {#2\xint_dothis<\xintexpralignlinesep\XINT:expr:toalign_a>\fi
    \xint_orthat<\xintexpralignrightbracket\XINT:expr:toalign_C>#1#2%
>%
\catcode`{ 1 \catcode`} 2 \catcode`< 12 \catcode`> 12
```

32.7.4. Transforming all leaves within nested braced contents

1.4. Leaves must be of catcode 12... This is currently not a constraint (or rather not a new constraint) for xintexpr because formerly anyhow all data went through csname encapsulation and extraction via string.

In order to share code with the functioning of universal functions, which will be allowed to transform a number into an ople, the applied macro is supposed to apply one level of bracing to its ouput. Thus to apply this with an xintfrac macro such as \xintiRound{0} one needs first to define a wrapper which will expand it inside an added brace pair:

\def\foo#1{{\xintiRound{0}{#1}}}

As the things will expand inside expanded, propagating expansion is not an issue.

This code is used by \xintiexpr and \xintfloatexpr in case of optional argument and by the «Universal functions».

Comment at 1.4l: this seems to be used only at private package level, else I should modify \XIN
↪ T:expr:mapwithin_b like I did with \XINT:expr:toblist_b to use regular braces in case the applied macro is \protected and things end up in external file.

```
\def\XINT:expr:mapwithin#1#2%
{%
    {{\expandafter\XINT:expr:mapwithin_checkempty
      \expanded{\noexpand#1!\expandafter}\detokenize{#2}^}}%
}%
\def\XINT:expr:mapwithin_checkempty #1!#2%
{%
    \if ^#2\expandafter\xint_gob_til_^\else
    \expandafter\XINT:expr:mapwithin_a\fi
    #1!#2%
```

```
}%
\begingroup
\catcode`[ 1 \catcode`] 2 \lccode`[`{ \lccode`]`}
\catcode`< 1 \catcode`> 2 \catcode`{ 12 \catcode`} 12
\lowercase<\endgroup
\def\XINT:expr:mapwithin_a #1{#2%
<%
    \if{#2\xint_dothis<[\iffalse]\fi\XINT:expr:mapwithin_a>\fi%
    \xint_orthat\XINT:expr:mapwithin_b #1#2%
>%
\def\XINT:expr:mapwithin_b #1!#2}%
<%
    #1<#2>\XINT:expr:mapwithin_c #1!}%
>%
\def\XINT:expr:mapwithin_c #1}#2%
<%
    \if ^#2\xint_dothis<\xint_gob_til_^>\fi
    \if{#2\xint_dothis<\XINT:expr:mapwithin_a>\fi%
    \xint_orthat<\iffalse[\fi]\XINT:expr:mapwithin_c>#1#2%
>%
>% back to normal catcodes
```

32.8. Top level user TEX interface: \xintfloateval, \xintiieval

32.8.1 \xintfloatexpr, \xintiiexpr . . . 710
32.8.2 \XINT_iiexpr_wrap, \XINT_flexpr_wrap . . . 711
32.8.3 \XINTiiexprprint, \XINTflexprprint . . . 712
32.8.4 \xintthe, \xintthealign, \xintthefloatexpr, \xintiiexpr . . . 713
32.8.5 \xintbarefloateval, \xintbareiieval . . . 713
32.8.6 \xintthebarefloateval, \xintthebareiieval . . . 714
32.8.7 \xintfloateval, \xintiieval . . . 714
32.8.8 \xintifboolfloatexpr, \xintifbooliiexpr . . . 715
32.8.9 \xintifNonFinitefloatexpr . . . 716
32.8.10 \xintifsgnfloatexpr, \xintifsgniiexpr . . . 716
32.8.11 Small bits we have to put somewhere . . . 716
\xintthecoords . . . 717
\xintthespaceseparated . . . 717

32.8.1. \xintfloatexpr, \xintiiexpr

Modified at 1.4o (2025/09/06).

While preparing 1.4n I had made changes to tame Babel active characters via a \csname trick. At last minute, after having first implemented correctly the feature both here and in bnumexpr I thought I had a way requiring not so many ``top level'' changes, but this was a grave logic error on my part, due to having a bit forgotten the way things are done here, and due to external circumstances. I released too hastily 1.4n. But the feature was working only with \xintiieval. Any test other than simply evaluating 3! with ! the Babel-French (e.g.1+3!) would have revealed it. I was planning to add test units after (what?) doing the release. By the way one has to be careful to do such tests after a \noindent because the French active ! in vertical mode does things not breaking the parser. Anyway, here I do it right.

So this is why for example \XINT_flexpr_noopt has a \csname construct.

```
\def\xintfloatexpr  {\romannumeral0\xintfloatexpro }%
```

```
\def\xintiiexpr     {\romannumeral0\xintiiexpro    }%
\def\xintiiexpro {\csname XINT_iiexpr_wrap\expandafter\endcsname
                  \romannumeral0\xintbareiieval }%
\def\xintfloatexpro #1%
{%
    \ifx [#1\expandafter\XINT_flexpr_withopt\else\expandafter\XINT_flexpr_noopt
    \fi #1%
}%
\def\XINT_flexpr_noopt
{%
    \csname XINT_flexpr_wrap\expandafter\endcsname
    \the\numexpr\XINTdigits\expandafter.%
    \romannumeral0\xintbarefloateval
}%
\def\XINT_flexpr_withopt [#1]%
{%
    \csname XINT_flexpr_withopt_a\expandafter\endcsname
    \the\numexpr\xint_zapspaces #1 \xint_gobble_i\expandafter.%
    \romannumeral0\xintbarefloateval
}%
\def\XINT_flexpr_withopt_a #1#2.%
{%
    \expandafter\XINT_flexpr_withopt_b\the\numexpr\if#1-\XINTdigits\fi#1#2.%
}%
\def\XINT_flexpr_withopt_b #1.%
{%
    \expandafter\XINT_flexpr_wrap
    \the\numexpr#1\expandafter.%
    \expanded
    \XINT:NEhook:x:mapwithin\XINT:expr:mapwithin{\XINTinFloat_braced[#1]}%
}%
```

calx: [Q] rounding of native leaves. Legacy was
`\def\XINTinFloat_braced[#1]#2{{\XINTinFloat[#1]{#2}}}`
but xe don't want an xintfrac.sty dependency.

There was a legacy comment "utiliser \XINTinFloatS ?", let us recall that this is a variant of \XINTinFloat which does not insist on a length P mantissa. So this is purely a legacy question which is obsolete with calx.

Maybe consider in this context \CALX_frez which trims trailing zeros, but if this ends up being fed as I believe it is again to \CALX_fround when going through the printer, it would be probably only a waste of time.

FIXME: calxfp.sty has `\def\CALX_fbr #1\xint:{ #1}`. Check \CALX_fround final stage to see if the \xint: removal could happen earlier.

```
\def\XINTinFloat_braced[#1]#2%
   {{\romannumeral0\expandafter\CALX_fbr
     \romannumeral0\CALX_fround{#1}{#2}}}%
```

32.8.2. \XINT_iiexpr_wrap, \XINT_flexpr_wrap

1.3e removes some leading space tokens which served nothing.

1.4e has \XINT_iexpr_wrap separated from \XINT_expr_wrap, thus simplifying internal matters as output printer for \xintexpr will not have to handle fixed point input but only extended-raw type input (i.e. A, A/B, A[N] or A/B[N]).

```
\def\XINT_iiexpr_wrap {\XINTfstop\XINTiiexprprint.}%
\def\XINT_flexpr_wrap {\XINTfstop\XINTflexprprint}%
```

32.8.3. \XINTiiexprprint, \XINTflexprprint

Modified at 1.4 (2020/01/31). Requires \expanded.
Comments here also apply to material moved to xintexpr.sty at 2.0a1.

Modified at 1.4e (2021/05/05). 1.4e has a breaking change of \XINTflexprprint and \xintfloatexprPrintOne which now requires \xintfloatexprPrintOne[D]{x} syntax, with first argument in brackets.

Modified at 1.4l (2022/05/29). The code does \let\xintexprPrintOne\xint_FracToSci_x but the latter is not yet defined so this is delayed until \xint_FracToSci_x definition.

Modified at 1.4m (2022/06/10). \xintboolexprPrintOne outputs true or false, not True or False. By the way (undocumented) all four keywords true, false, True, False are recognized as genuine variables since 1.4i.

```
\def\xintexprEmptyItem{[]}%
\protected\def\XINTiiexprprint.%
    {\XINT:NEhook:x:toblist\XINT:expr:toblistwith\CALX_iipr}%
\def\CALX_iipr #1{\expandafter\xintiiexprPrintOne
                  \romannumeral0\CALX_mr_a#1\xint:}%
\def\xintiiexprPrintOne #1{%
\def\xintiiexprPrintOne ##1%
{%
    \xint_gob_til_Ixii ##1\CALX_iipr_special #1##1%
}}\expandafter\xintiiexprPrintOne\string I%
\def\CALX_iipr_special #1#2#3{%
    \if ?#3\xint_dothis{NaN}\fi
    \xint_orthat{#3Infinity}%
}%
\protected\def\XINTflexprprint #1.%
    {\XINT:NEhook:x:toblist\XINT:expr:toblistwith{\xintfloatexprPrintOne[#1]}}%
```

calx: native leaves: \CALX_fround then \CALX_ftoraw replace the \XINTinfloatS of xint's \xintPFloat_wopt, whose formatting macros (\XINT_pfloat, hooks \xintPFloatE etc.) are used unchanged. Specials I+, I-, I? are printed as is (provisional).

```
\def\xintfloatexprPrintOne[#1]#2%
{%
    \romannumeral0\expandafter\CALX_fpr_a
    \romannumeral0\CALX_fround{#1}{#2}#1.%
}%
\def\CALX_fpr_a#1{%
  \def\xint_gob_til_Ixii##1#1{}%
  \def\CALX_fpr_a ##1%
  {%
    \xint_gob_til_Ixii ##1\CALX_fpr_special #1%
    \expandafter\XINT_pfloat_fork\romannumeral0\CALX_frez ##1%
  }%
}\expandafter\CALX_fpr_a\string I%
\def\CALX_fpr_special #1\CALX_frez #2#3\xint:#4.{%
    \if ?#3\xint_dothis{ NaN}\fi
    \xint_orthat{ #3Infinity}%
}%
```

32.8.4. \xintthe, \xintthealign, \xintthefloatexpr, \xinttheiiexpr

The reason why \xinttheiiexpr et \xintthefloatexpr are handled differently is that they admit an optional argument which acts via a custom «printing» stage.

We exploit here that \expanded expands forward until finding an implicit or explicit brace, and that this expansion overrules \protected macros, forcing them to expand, similarly as \romannumeral expands \protected macros, and contrarily to what happens *within* the actual \expanded scope. I discovered this fact by testing (with pdftex) and I don't know where this is documented apart from the source code of the relevant engines. This is useful to us because there are contexts where we will want to apply a complete expansion before printing, but in purely numerical context this is not needed (if I converted correctly after dropping at 1.4 the \csname governed expansions; however I rely at various places on the fact that the xint macros are f-expandable, so I have tried to not use zillions of expanded all over the place), hence it is not needed to add the expansion overhead by default. But the \expanded here will allow \xintNewExpr to create macro with suitable modification or the printing step, via some hook rather than having to duplicate all macros here with some new «NE» meaning (aliasing does not work or causes big issues due to desire to support \xinteval also in «NE» context as sub-constituent. The \XINT:NEhook:x:toblist is something else which serves to achieve this support of *sub* \xinteval, it serves nothing for the actual produced macros. For \xintdeffunc, things are simpler, but still we support the [N] optional argument of \xintiiexpr and \xintfloatexpr, which required some work...

The \expanded upfront ensures \xintthe mechanism does expand completely in two steps.

The fact that \xintthe grabs a #1 is legacy and I am not sure why. I vaguely remember thinking the overhead was minimal because #1 will simply be one token such as \xintexpr and that it could potentially be useful. Of course \xintthealign imitates \xintthe in that matter.

Modified at 1.4o (2025/09/06). Again here some \csname's to handle babel-active characters.

```
\def\xintthe      #1{\expanded\expandafter\xint_gobble_i\romannumeral`&&@#1}%
\def\xintthealign #1{\expandafter\xintexpralignbegin
                     \expanded\expandafter\XINT:expr:toalignwith
    \romannumeral0\expandafter\expandafter\expandafter\expandafter
    \expandafter\expandafter\expandafter\xint_gob_andstop_ii
                     \expandafter\xint_gobble_i\romannumeral`&&@#1}%
\def\xinttheiiexpr
   {\expanded\csname XINTiiexprprint\expandafter\endcsname
    \expandafter.\romannumeral0\xintbareiieval}%
```

Modified at 1.4p (2026/03/01). I don't know why I had been using \romannumeral`&&@\xintiiexpr. Perhaps at some point \xintiiexpro and \xintfloatexpro were not available? Or maybe I simply was under influence of \xintthe. I have not checked history.

No need at 1.4o of \csname's for Babel here, as \xintiiexpro and \xintfloatexpro are already taken care of.

```
\def\xintthefloatexpr
   {\expanded\expandafter\xint_gobble_i\romannumeral0\xintfloatexpro}%
```

32.8.5. \xintbarefloateval, \xintbareiieval

Modified at 1.4 (2020/01/31). one expansion step added via \XINT_expr_start, \XINT_iiexpr_start, \XINT_flexpr_start. Trigger is expected to be via \romannumeral`^^@ or \romannumeral0.

For the benefit of those who like the author while finishing 1.4n looked in vain for where the ``_start'' macros were defined, this documentation was updated so that one can find the location via clicking on their names in the previous paragraph. Spoiler: they are in subsection 32.15. This was hidden because the three are defined in an \xintFor in one-go.

```
\def\xintbarefloateval{\XINT_flexpr_start}%
\def\xintbareiieval   {\XINT_iiexpr_start}%
```

32.8.6. \xintthebarefloateval, \xintthebareiieval

For matters of \XINT_NewFunc

Modified at 1.4o (2025/09/06).
The matter of Babel active here too, although these macros are destined to use by \xintdeffunc and \xintNewExpr and both attempt to sanitize catcodes. But I had already edited at 1.4n the user documentation to mention cases where changing catcodes is impossible. It would have been too time-costly late at night to re-edit while I was eager to push the 1.4o hotfix. So let's live with some added overhead.

Modified at 1.4p (2026/03/01). s/_stop_atfirstofone/_firstofone because anyhow it will encounter an opening brace.

```
\def\XINT_expr_unlock{\expandafter\xint_firstofone\romannumeral`&&@}%
\def\xintthebareiieval{%
  \romannumeral0\csname xint_firstofone\expandafter\endcsname
  \romannumeral0\xintbareiieval
}%
\def\xintthebarefloateval{%
  \romannumeral0\csname xint_firstofone\expandafter\endcsname
  \romannumeral0\xintbarefloateval
}%
\def\xintthebareroundedfloateval{%
  \romannumeral0\csname xintthebareroundedfloateval_a\expandafter\endcsname
  \romannumeral0\xintbarefloateval
}%
\def\xintthebareroundedfloateval_a{%
  \expandafter\xint_firstofone
  \expanded\XINT:NEhook:x:mapwithin
           \XINT:expr:mapwithin{\XINTinFloatSdigits_braced}%
}%
```

calx: native rounding (F, I, or raw input) for \xintdeffloatvar.

```
\def\XINTinFloatSdigits_braced#1%
   {{\romannumeral0\expandafter\CALX_fbr
     \romannumeral0\CALX_fround{\XINTdigits}{#1}}}%
```

32.8.7. \xintfloateval, \xintiieval

Refactored at 1.4.
The \expanded upfront ensures \xinteval still expands completely in two steps. No \romannumeral trigger here, in relation to the fact that \XINTexprprint is not f-expandable, only e-expandable.
(and attention that \xintexpr\relax is now legal, and an empty ople can be produced in output also from \xintexpr [17][1]\relax for example)

Modified at 1.4k (2022/05/18).
The \xintieval and \xintfloateval optional bracketed argument can now be located outside the braces... took me years to finally make the step towards LaTeX users expectations for the interface.

Modified at 1.4o (2025/09/06). Some \csname's for the Babel-active feature which was botched at 1.4n. But \xintiexpr and \xintifloatexpr are already taken care of.
Should I use #{ rather delimited argument? (as for \xintiieval at 1.4n).
Same updates at 1.4p as for \xintieval.

```
\def\xintfloateval {\expanded\expandafter\xint_floateval_chkopt\string}%
\def\xint_floateval_chkopt #1%
```

```
{%
    \ifx [#1\expandafter\xint_floateval_opt
       \else\expandafter\xint_floateval_noopt
    \fi  #1%
}%
\def\xint_floateval_opt [#1]#2{%
  \expandafter\xint_gobble_i\romannumeral0\XINT_flexpr_withopt[#1]#2\relax
}%
\def\xint_floateval_noopt #1%
   {\expandafter\xint_floateval\expandafter{\iffalse}\fi}%
\def\xint_floateval#1{%
  \expandafter\xint_gobble_i\romannumeral0\xintfloatexpro#1\relax
}%
```

Modified at 1.4n (2025/09/05). \xintiieval admits optional arguments [h], [o] and [b]. The way this is implemented tames Babel active characters.

```
\def\xintiieval #1#{\expanded\xintiieval_a{#1}}%
\def\xintiieval_a#1#2%
{%
    \csname XINTiiexprprint\xint_zapspaces #1 \xint_gobble_i
    \expandafter\endcsname
    \expandafter.\romannumeral0\xintbareiieval#2\relax
}%
\expandafter\def\csname XINTiiexprprint[h]\endcsname.%
    {\XINT:NEhook:x:toblist\XINT:expr:toblistwith\CALX_iiprhex}%
\expandafter\def\csname XINTiiexprprint[o]\endcsname.%
    {\XINT:NEhook:x:toblist\XINT:expr:toblistwith\CALX_iiproct}%
\expandafter\def\csname XINTiiexprprint[b]\endcsname.%
    {\XINT:NEhook:x:toblist\XINT:expr:toblistwith\CALX_iiprbin}%
\expandafter\let\csname XINTiiexprprint[]\endcsname\XINTiiexprprint
\def\CALX_iiprhex #1{\xintiiexprPrintOneHex{\CALX_mr{#1}}}%
\def\CALX_iiproct #1{\xintiiexprPrintOneOct{\CALX_mr{#1}}}%
\def\CALX_iiprbin #1{\xintiiexprPrintOneBin{\CALX_mr{#1}}}%
\let\xintiiexprPrintOneHex\xintDecToHex
\let\xintiiexprPrintOneOct\xintDecToOct
\let\xintiiexprPrintOneBin\xintDecToBin
```

32.8.8. \xintifboolfloatexpr, \xintifbooliiexpr

Added at 1.09c (2013/10/09). They do not accept (i.e.they cause breakage if used this way) comma separated expressions as input.

Modified at 1.4p (2026/03/01). Sadly at 1.4e, some internal changes to the output format of \xintheexpr broke that the assumptions which excused the use of integer only \xintiiifnotzero. As a result the macro became badly broken as it started thinking that for example 0.75 is zero. Fortunately most usage of it is with some inequality test, which produces either one or zero. So the author became aware of this only 5 years later.

```
\def\xintifboolfloatexpr #1{%
    \romannumeral0\expandafter\CALX_ifnz
    \romannumeral0\csname xint_firstofone\expandafter\endcsname
    \romannumeral0\xintbarefloateval #1\relax
}%
\def\xintifbooliiexpr #1{%
    \romannumeral0\expandafter\CALX_ifnz
```

```
    \romannumeral0\csname xint_firstofone\expandafter\endcsname
    \romannumeral0\xintbareiieval #1\relax
}%
```

32.8.9. \xintifNonFinitefloatexpr

The expression is evaluated once. Branch on its native result, I+, I-, or I? for non-finite, any other numeric form for finite. This is independent of signal flags and of the configurable printed spelling of special values.

```
\def\xintifNonFinitefloatexpr #1{%
    \romannumeral0\expandafter\CALX_ifnonfinite
    \romannumeral0\csname xint_firstofone\expandafter\endcsname
    \romannumeral0\xintbarefloateval #1\relax
}%
\def\CALX_ifnonfinite #1{%
    \expandafter\CALX_ifnonfinite_a\romannumeral`&&@#1\xint:}%
\def\CALX_ifnonfinite_a #1{%
    \if I#1\expandafter\CALX_ifnonfinite_y
    \else\expandafter\CALX_ifnonfinite_n
    \fi
}%
\def\CALX_ifnonfinite_y #1\xint:{\xint_stop_atfirstoftwo}%
\def\CALX_ifnonfinite_n #1\xint:{\xint_stop_atsecondoftwo}%
```

32.8.10. \xintifsgnfloatexpr, \xintifsgniiexpr

Added at 1.3d (2019/01/06). They do not accept (i.e.they will cause breakage if used with) comma separated expressions.

Modified at 1.4p (2026/03/01). Badly broken since a 1.4e regression, see \xintifboolexpr. Situation here was much worse, of course. I use \xintthebareeval to fix this as I am stubborn and do not want the parsing from \xintifsgn but this is silly. For \xintifsgniiexpr, matters are more moot but there is always the danger of a custom \xintiiexprPrintOne, so \xintbareiieval here too. For the other two really I should stop using the ii macros.

```
\def\xintifsgnfloatexpr #1{%
    \romannumeral0\expandafter\CALX_ifsgn
    \romannumeral0\csname xint_firstofone\expandafter\endcsname
    \romannumeral0\xintbarefloateval #1\relax
}%
\def\xintifsgniiexpr #1{%
    \romannumeral0\expandafter\CALX_ifsgn
    \romannumeral0\csname xint_firstofone\expandafter\endcsname
    \romannumeral0\xintbareiieval #1\relax
}%
```

32.8.11. Small bits we have to put somewhere

Some renaming and modifications here with release 1.2 to switch from using chains of \romannumeral-`0 in order to gather numbers, possibly hexadecimals, to using a \csname governed expansion. In this way no more limit at 5000 digits, and besides this is a logical move because the \xintexpr parser is already based on \csname...\endcsname storage of numbers as one token.

The limitation at 5000 digits didn't worry me too much because it was not very realistic to launch computations with thousands of digits... such computations are still slow with 1.2 but less so

now. Chains or \romannumeral are still used for the gathering of function names and other stuff which I have half-forgotten because the parser does many things.

In the earlier versions we used the lockscan macro after a chain of \romannumeral-`0 had ended gathering digits; this uses has been replaced by direct processing inside a \csname...\endcsname and the macro is kept only for matters of dummy variables.

Currently, the parsing of hexadecimal numbers needs two nested \csname...\endcsname, first to gather the letters (possibly with a hexadecimal fractional part), and in a second stage to apply \xintHexToDec to do the actual conversion. This should be faster than updating on the fly the number (which would be hard for the fraction part...).

```
\def\XINT_embrace#1{{#1}}%
\def\xint_gob_til_! #1!{}% ! with catcode 11
\def\xintError:noopening
{%
    \XINT_expandableerror{Extra ). This is serious and prospects are bleak.}%
}%
```

\xintthecoords Added at 1.1 (2014/10/28). (see user manual for interface)

```
\def\xintthecoords#1%
   {\romannumeral`&&@\expandafter\XINT_thecoords_a\romannumeral0#1}%
\def\XINT_thecoords_a #1#2.#3% #2.=\XINTfloatprint<digits>. etc...
   {\expanded{\expandafter\XINT_thecoords_b\expanded#2.{#3},!,!,^}}%
\def\XINT_thecoords_b #1#2,#3#4,%
   {\xint_gob_til_! #3\XINT_thecoords_c ! (#1#2, #3#4)\XINT_thecoords_b }%
\def\XINT_thecoords_c #1^{}%
```

\xintthespaceseparated Added at 1.4a (2020/02/19).

This is a utility macro which was distributed previously separately for usage with PSTricks \listplot.

```
\def\xintthespaceseparated#1%
   {\expanded\expandafter\xintthespaceseparated_a\romannumeral0#1}%
\def\xintthespaceseparated_a #1#2.#3%
   {{\expandafter
      \xintthespaceseparated_b\expanded#2.{#3},!,!,!,!,!,!,!,!,^}}%
\def\xintthespaceseparated_b #1,#2,#3,#4,#5,#6,#7,#8,#9,%
   {\xint_gob_til_! #9\xintthespaceseparated_c !%
    #1#2#3#4#5#6#7#8#9%
    \xintthespaceseparated_b}%
```

1.4c I add a space here to stop the \romannumeral`&&@ in case of empty input. But this space induces an extra un-needed space token after 9, 18, 27,... items before the last group of less than 9 items.

Fix (at 1.4h) is simple because I already use \expanded anyhow: I don't need at all the \romannumeral`&&@ which was first in \xintthespaceseparated, let's move the first \expanded which was in \xintthespaceseparated_a to \xintthespaceseparated, and remove the extra space here in _c.

(alternative would have been to put the space after #1 and accept a systematic trailing space, at least it is more aesthetic).

Again, I did have a test file, but it was not incorporated in my test suite, so I discovered the problem accidentally by compiling all files in an archive.

```
\def\xintthespaceseparated_c !#1!#2^{#1}%
```

32.9. Hooks into the numeric parser for usage by the \xintdeffunc symbolic parser

This is new with 1.3 and considerably refactored at 1.4. See «Mysterious stuff».

```
\let\XINT:NEhook:f:one:from:one\expandafter
\let\XINT:NEhook:f:one:from:one:direct\empty
\let\XINT:NEhook:f:one:from:two\expandafter
\let\XINT:NEhook:f:one:from:two:direct\empty
\let\XINT:NEhook:x:one:from:two\empty
\let\XINT:NEhook:f:one:and:opt:direct      \empty
\let\XINT:NEhook:f:tacitzeroifone:direct   \empty
\let\XINT:NEhook:f:iitacitzeroifone:direct \empty
\let\XINT:NEhook:x:select:obey\empty
\let\XINT:NEhook:x:listsel\empty
\let\XINT:NEhook:f:reverse\empty
```

At 1.4 it was \def\XINT:NEhook:f:from:delim:u #1#2^{#1#2^} which was trick to allow automatic unpacking of a nutple argument to multi-arguments functions such as gcd() or max(). But this sacrificied the usage with a single numeric argument.

Modified at 1.4i (2021/06/11). More sophisticated code to check if the argument ople was actually a single number. Notice that this forces numeric types to actually use catcode 12 tokens, and polexpr diverges a bit using P, but actually always testing with \if not \ifx.

This is used by gcd(), lcm(), max(), min(), `+`(), `*`(), all(), any(), xor().

The nil and None will give the same result due to the initial brace stripping done by \XINT:NEhook:f:from:delim:u (there was even a prior brace stripping to provide the #2 which is empty here for the nil and {} for the None).

```
\def\XINT:NEhook:f:from:delim:u #1#2^%
{%
    \expandafter\XINT_fooof_checkifnumber\expandafter#1\string#2^%
}%
\def\XINT_fooof_checkifnumber#1#2%
{%
    \expandafter#1%
    \romannumeral0\expanded{\if ^#2^\else
              \if\bgroup#2\noexpand\XINT_fooof_no\else
              \noexpand\XINT_fooof_yes#2\fi\fi}%
}%
\def\XINT_fooof_yes#1^{{#1}^}%
\def\XINT_fooof_no{\expandafter{\iffalse}\fi}%
```

Modified at 1.4i (2021/06/11). Same changes as for the other multiple arguments functions, making them again usable with a single numeric input.

Was at 1.4 \def\XINT:NEhook:f:noeval:from:braced:u#1#2^{#1{#2}} which is not compatible with a single numeric input.

Used by len(), first(), last() but it is a potential implementation bug that the three share this as the location where expansion takes places is one level deeper for the support macro of len().

The None is here handled as nil, i.e. it is unpacked, which is fine as the documentations says nutples are unpacked.

```
\def\XINT:NEhook:f:LFL #1{\expandafter#1\expandafter}%
\def\XINT:NEhook:r:check #1^%
{%
    \expandafter\XINT:NEhook:r:check_a\string#1^%
}%
```

```
\let\XINT:NEsaved:r:check \XINT:NEhook:r:check
\def\XINT:NEhook:r:check_a #1%
{%
    \if ^#1\xint_dothis\xint_c_\fi
    \if\bgroup#1\xint_dothis\XINT:NEhook:r:check_no\fi
    \xint_orthat{\XINT:NEhook:r:check_yes#1}%
}%
\def\XINT:NEhook:r:check_no
{%
    \expandafter\XINT:NEhook:r:check_no_b
    \expandafter\xint_c_\expandafter{\iffalse}\fi
}%
\def\XINT:NEhook:r:check_no_b#1^{#1}%
\def\XINT:NEhook:r:check_yes#1^{\xint_c_{#1}}%
\let\XINT:NEhook:branch\expandafter
\let\XINT:NEhook:seqx\empty
\let\XINT:NEhook:iter\expandafter
\let\XINT:NEhook:opx\empty
\let\XINT:NEhook:rseq\expandafter
\let\XINT:NEhook:iterr\expandafter
\let\XINT:NEhook:rrseq\expandafter
\let\XINT:NEhook:x:toblist\empty
\let\XINT:NEhook:x:mapwithin\empty
\let\XINT:NEhook:x:ndmapx\empty
```

32.10. \XINT_expr_getnext: fetch some value then an operator and present them to last waiter with the found operator precedence, then the operator, then the value

Big change in 1.1, no attempt to detect braced stuff anymore as the [N] notation is implemented otherwise. Now, braces should not be used at all; one level removed, then \romannumeral-`0 expansion.

Refactored at 1.4 to put expansion of \XINT_expr_getop after the fetched number, thus avoiding it to have to fetch it (which could happen then multiple times, it was not really important when it was only one token in pre-1.4 xintexpr).

Allow \xintexpr\relax at 1.4.

Refactored at 1.4 the articulation \XINT_expr_getnext/XINT_expr_func/XINT_expr_getop. For some legacy reason the first token picked by getnext was soon turned to catcode 12 The next ones after the first were not a priori stringified but the first token was, and this made allowing things such as \xintexpr\relax, \xintexpr,,\relax, [], 1+(), [:] etc... complicated and requiring each time specific measures.

The \expandafter chain in \XINT_expr_put_op_first is an overhead related to an 1.4 attempt, the "varvalue" mechanism. I.e.: expansion of \XINT_expr_var_foo is {\XINT_expr_varvalue_foo } and then for example \XINT_expr_varvalue_foo expands to {4/1[0]}. The mechanism was originally conceived to have only one token with idea its makes things faster. But the xintfrac macros break with syntax such as \xintMul\foo\bar and \foo expansion giving braces. So at 1.4c I added here these \expandafter, but this is REALLY not satisfactory because the \expandafter are needed it seems only for this variable "varvalue" mechanism.

See also the discussion of \XINT_expr_op__ which distinguishes variables from functions.

After a 1.4g refactoring it would be possible to drop here the \expandafter if the \XINT_expr_var
↪ _foo macro was defined to f-expand to {actual expanded value (as ople)} for example explicit {{3}}.
I have to balance the relative weights of doing always the \expandafter but they are needed only for

the case the value was encapsulated in a variable, and of never doing the \expandafter and ensure f-expansion of the _var_foo gives explicit value (now that the refactoring let it be f-expanded, and the case of fake variables omit and abort in particular was safely separated instead of being treated like other and imposing restrictions on general variable handling), and then there is the overhead of possibly moving around many digits in the #1 of \XINT_expr_put_op_first.

```
\def\XINT_expr_getnext #1%
{%
    \expandafter\XINT_expr_put_op_first\romannumeral`&&@%
    \expandafter\XINT_expr_getnext_a\romannumeral`&&@#1%
}%
\def\XINT_expr_put_op_first #1#2#3{%
     \expandafter#2\expandafter#3\expandafter{#1}%
}%
\def\XINT_expr_getnext_a #1%
{%
    \ifx\relax #1\xint_dothis\XINT_expr_foundprematureend\fi
    \ifx\XINTfstop#1\xint_dothis\XINT_expr_subexpr\fi
    \ifcat\relax#1\xint_dothis\XINT_expr_countetc\fi
    \xint_orthat{}\XINT_expr_getnextfork #1%
}%
\def\XINT_expr_foundprematureend\XINT_expr_getnextfork #1{{}\xint_c_\relax}%
\def\XINT_expr_subexpr #1.#2%
{%
    \expanded{\xint_noxpd{{#2}}\expandafter}\romannumeral`&&@\XINT_expr_getop
}%
```

The "fetch as number" must be avoided for those cases where \number can not be hoped to need only one token. Hence the liste of exceptions.

Modified at 1.2 (2015/10/10). Add \ht, \dp, \wd and the eTEX font related primitives.

Modified at 1.4 (2020/01/31). Refactor for readability to avoid huge amounts of \fi's.

Modified at 1.4g (2021/05/25). Check also for \catcode so that \xinteval{\catcode`@} does not crash. But later I observed that \fpeval{\catcode`@} does crash, so in retrospect I wonder if I should have added the overhead.

Modified at 1.4n (2025/09/05). Added _catcode to the list for OpTEX. This means though that an undefined token with other engines will be expanded via \number...

```
\def\XINT_expr_countetc\XINT_expr_getnextfork#1%
{%
    \if0\ifx\count#11\fi
        \ifx\numexpr#11\fi
        \ifx\catcode#11\fi
        \ifx\dimen#11\fi
        \ifx\dimexpr#11\fi
        \ifx\skip#11\fi
        \ifx\glueexpr#11\fi
        \ifx\fontdimen#11\fi
        \ifx\ht#11\fi
        \ifx\dp#11\fi
        \ifx\wd#11\fi
        \ifx\fontcharht#11\fi
        \ifx\fontcharwd#11\fi
        \ifx\fontchardp#11\fi
        \ifx\fontcharic#11\fi
```

```
499         \ifx\_catcode#11\fi
500        0\expandafter\XINT_expr_fetch_as_number\fi
501    \expandafter\XINT_expr_getnext_a\number #1%
502 }%
503 \def\XINT_expr_fetch_as_number
504     \expandafter\XINT_expr_getnext_a\number #1%
505 {%
506     \expanded{{{\number#1}}\expandafter}\romannumeral`&&@\XINT_expr_getop
507 }%
```

This is the key initial dispatch component. It has been refactored at 1.4g to give priority to identifying letter and digit tokens first. It thus combines former \XINT_expr_getnextfork, \XINT_expr_scan_nbr_or_func and \XINT_expr_scanfunc. A branch of the latter having become \XINT_expr_startfunc. The handling of non-catcode 11 underscore _ has changed: it is now skipped completely like the +. Formerly it would cause an infinite loop because it triggered first insertion of a nil variable, (being confused with a possible operator at a location where one looks for a value), then tacit multiplication (being now interpreted as starting some name), and then it came back to getnextfork creating loop. The @ of catcode 12 could have caused the same issue if it was not handled especially because it is used in the syntax as special variable for recursion hence was recognized even if of catcode 12. Anyway I could have handled the _ like the @, to avoid this problem of infinite loop with a non-letter underscore used as first character but decided finally to have it be ignored (it is already ignored if among digits, but it can be a constituent of a function of variable name). It is not ignored of course if of catcode 11. It may then start a variable or function name, but only for use by the package (by polexpr for example), not by users.

Then the matter is handed over to specialized routines: gathering digits of a number (inclusive of a decimal mark, an exponential part) or letters of a function or variable. And we have to intercept some tokens to implement various functionalities.

In each dothis/orthat structure, the first encountered branches are usually handled slower than the next, because \if..\fi test cost less than grabbing tokens. The exception is in the first one where letters pass through slightly faster than digits, presumably because the \ifnum test is more costly. Prior to this 1.4g refactoring the case of a starting letter of a variable or function name was handled last, it is now handled first. Now, this is only first letter...

Here are the various possibilities in order that they are appear below (the indicative order of speed of treatment is given as a number).

-1 tokens of catcode letter start a variable or function name

-2 digits (I apply \string for the test, but I will have to review, it seems natural anyhow to require digits to be of catcode 12 and this is in fact basically done by the package, \numexpr does not work if not the case.),

-7 support for Python-like * "unpacking" unary operator (added at 1.4),

-6 support for [as opener for the [..] nutple constructor (1.4),

-5 support for the minus as unary operator of variable precedence,

-4 support for @ as first character of special variables even if not letter,

-3 support for opening parentheses (possibly triggering tacit multiplication),

-13 support for skipping over ignored + character,

-12 support for numbers starting with a decimal point,

-11 support for the `+`() and `*`() functions,

-10 support for the !() function,

-9 support for the ?() function,

-8 support for " for input of hexadecimal numbers and (1.4n) of ' for octal numbers.

-17 support for \xintdeffunc via special handling of # token,

-16 support for ignoring _ if not of catcode 11 and at start of numbers or names (this 1.4g change fixes \xinteval{_4} creating infinite loop)

-15 support for inserting "nil" in front of operators, as needed in particular for the Python slicing syntax. This covers the comma, the :, the] and the) and also the ; although I don't think using ; to delimit nil is licit.

-14 support for inserting 0 as missing value if / or ^ are encountered directly. This 1.4g changes avoids \xinteval{/3} causing unrecoverable low level errors from \xintDiv receiving only one argument.

I did not see here other bad syntax to protect.

The handling of "nil" insertion penalizes Python slicing but anyway time differences in the 14-15-16-17 group are less than 5%. The alternative will be to do some positive test for the targets (:,], the comma and closing parenthesis) and do this in the prior group but this then penalizes others. Anyway. This is all negligible compared to actual computations...

Note: the above may not be in sync with code as it is extremely time-consuming to maintain correspondence in case of re-factoring.

```
\def\XINT_expr_getnextfork #1%
{%
    \ifcat a#1\xint_dothis\XINT_expr_startfunc\fi
    \ifnum \xint_c_ix<1\string#1 \xint_dothis\XINT_expr_startint\fi
    \xint_orthat\XINT_expr_getnextfork_a #1%
}%
\def\XINT_expr_getnextfork_a #1%
{%
    \if#1*\xint_dothis {{}\xint_c_ii^v 0}\fi
    \if#1[\xint_dothis {{}\xint_c_ii^v \XINT_expr_itself_obracket}\fi
    \if#1-\xint_dothis {{}{}-}\fi
    \if#1@\xint_dothis{\XINT_expr_startfunc @}\fi
    \if#1(\xint_dothis {{}\xint_c_ii^v (}\fi
    \xint_orthat{\XINT_expr_getnextfork_b#1}%
}%
\catcode96 11 % `
\def\XINT_expr_getnextfork_b #1%
{%
    \if#1+\xint_dothis \XINT_expr_getnext_a\fi
    \if#1.\xint_dothis \XINT_expr_startdec\fi
    \if#1`\xint_dothis {\XINT_expr_onliteral_`}\fi
    \if#1!\xint_dothis {\XINT_expr_startfunc !}\fi
    \if#1?\xint_dothis {\XINT_expr_startfunc ?}\fi
    \if#1'\xint_dothis \XINT_expr_startoct\fi
    \if#1"\xint_dothis \XINT_expr_starthex\fi
    \xint_orthat{\XINT_expr_getnextfork_c#1}%
}%
\def\XINT_tmpa #1{%
\def\XINT_expr_getnextfork_c ##1%
{%
    \if##1#1\xint_dothis \XINT_expr_getmacropar\fi
    \if##1_\xint_dothis \XINT_expr_getnext_a\fi
    \if0\if##1/1\fi\if##1^1\fi0\xint_dothis{\XINT_expr_insertnil##1}\fi
    \xint_orthat{\XINT_expr_missing_arg##1}%
}%
}\expandafter\XINT_tmpa\string#%
```

The ` syntax is here used for special constructs like `+`(..), `*`(..) where + or * will be treated as functions. Current implementation picks only one token (could have been braced stuff), here it will be + or *, and via \XINT_expr_op_` this then becomes a suitable \XINT

```
↪ _{expr|iiexpr|flexpr}_func_+ (or *).  Documentation says to use `+`(...), but `+(...)  is also
valid.  The opening parenthesis must be there, it is not allowed to require some expansion.
544 \def\XINT_expr_onliteral_` #1#2({{#1}\xint_c_ii^v `}%
545 \catcode96 12 % `
```

Prior to 1.4g, I was using a \lowercase technique to insert the catcode 12 #, but this is a bit risky when one does not ensure a priori control of all lccodes.

```
546 \def\XINT_tmpa #1{%
547 \def\XINT_expr_getmacropar ##1%
548 {%
549     \expandafter{\expandafter{\expandafter#1\expandafter
550     ##1\expandafter}\expandafter}\romannumeral`&&@\XINT_expr_getop
551 }%
552 }\expandafter\XINT_tmpa\string#%
553 \def\XINT_expr_insertnil #1%
554 {%
555     \expandafter{\expandafter}\romannumeral`&&@\XINT_expr_getop_a#1%
556 }%
557 \def\XINT_expr_missing_arg#1%
558 {%
559     \expanded{\XINT_expandableerror
560               {Expected a value, got nothing before `#1'.
561                Inserting 0.}{{0}}\expandafter}%
562     \romannumeral`&&@\XINT_expr_getop_a#1%
563 }%
```

32.11. \XINT_expr_startint

32.11.1 Integral part (skipping zeroes) . . . 725
32.11.2 Fractional part . . . 726
32.11.3 Scientific notation . . . 728
32.11.4 Hexadecimal numbers . . . 729
32.11.5 Octal numbers . . . 732
32.11.6 Binary numbers . . . 734
32.11.7 \XINT_expr_startfunc: collecting names of functions and variables . . . 736
32.11.8 \XINT_expr_func: dispatch to variable replacement or to function execution . . . 737

Following comments are in part OBSOLETE as the code was refactored at some stage to use \expand
↪ ed.

1.2 release has replaced chains of \romannumeral-`0 by \csname governed expansion. Thus there is no more the limit at about 5000 digits for parsed numbers.

In order to avoid having to lock and unlock in succession to handle the scientific part and adjust the exponent according to the number of digits of the decimal part, the parsing of this decimal part counts on the fly the number of digits it encounters.

OBSOLETE START There is some slight annoyance with \xintiiexpr which should never be given a [n] inside its \csname.=<digits>\endcsname storage of numbers (because its arithmetic uses the ii macros which know nothing about the [N] notation). Hence if the parser has only seen digits when hitting something else than the dot or e (or E), it will not insert a [0]. Thus we very slightly compromise the efficiency of \xintexpr and \xintfloatexpr in order to be able to share the same code with \xintiiexpr.

Indeed, the parser at this location is completely common to all, it does not know if it is working inside \xintexpr or \xintiiexpr. On the other hand if a dot or a e (or E) is met, then the (common)

parser has no scrupules ending this number with a [n], this will provoke an error later if that was within an \xintiiexpr, as soon as an arithmetic macro is used. OBSOLETE END

calx: the scanner is common to the three parsers; it ends a number by a fake operator of precedence 22 (above every right precedence), which the waiting operator of the running parser executes at once through \XINT_<p>_op_<char>. Integer exits (digits only, zeros, an integer followed by .., integer hexadecimal, octal, binary): ^^B; the receivers append [0] (\xintexpr, \xintfloatexpr) or build the native integer M (\xintiiexpr). Decimal mark or exponent (and fractional hexadecimal...): the exponent N is handed to ^^C outside the value braces; the receivers append [N] (\xintexpr, \xintfloatexpr) or check for an integer value (\xintiiexpr, \CALX_iilit). The receivers call \XINT_expr_getop for the real operator.

As the gathered numbers have no spaces, no pluses, no minuses, the only remaining issue is with leading zeroes, which are discarded on the fly. The hexadecimal numbers leading zeroes are stripped in a second stage by the \xintHexToDec macro.

With 1.2, \xinttheexpr . \relax does not work anymore (it did in earlier releases). There must be digits either before or after the decimal mark. Thus both \xinttheexpr 1.\relax and \xinttheexpr .1\relax are legal.

Attention at this location #1 was of catcode 12 in all versions prior to 1.4.

We assume anyhow that catcodes of digits are 12...

Style of location of the braces around replacement text varies, but is kept as is to avoid wasting time on cosmetics.

Modified at 1.4n (2025/09/05). Support for the 0b, 0o and 0x prefixes! Implementation of this extension of the syntax was very easy. I may have thought of it briefly in the past but presumed it would break things.

This change seems to only break stuff such as +0bar+3 with bar a variable being interpreted +0*bar+3 (tacit multiplication) and will cause the parser to raise then an error message, but this is not very serious breaking change.

Supporting 0b meant that like for octal 1.4n needed to add the needed support macros, which opens this TODO: perhaps try to share code between binary, octal, hexadecimal more than currently.

```
\def\XINT_expr_startint #1%
{%
    \if #10\expandafter\XINT_expr_gobz_a\else
           \expandafter\XINT_expr_scanint_a\fi #1%
}%
\def\XINT_expr_scanint_a #1#2%
    {\expanded\bgroup{{\iffalse}}\fi #1% spare a \string
     \expandafter\XINT_expr_scanint_main\romannumeral`&&@#2}%
\def\XINT_expr_gobz_a #1#2%
    {\expandafter\XINT_expr_gobz_b\romannumeral`&&@#2}%
```

Perhaps use \if and not \ifx tests here?

```
\def\XINT_expr_gobz_b #1%%
{%
    \ifx b#1\xint_dothis \XINT_expr_startbin \fi
    \ifx o#1\xint_dothis \XINT_expr_startoct \fi
    \ifx x#1\xint_dothis \XINT_expr_starthex \fi
    \xint_orthat {\XINT_expr_gobz_c #1}%
}%
\def\XINT_expr_gobz_c #1%
    {\expanded\bgroup{{\iffalse}}\fi
     \expandafter\XINT_expr_gobz_scanint_main#1}%
\def\XINT_expr_startdec #1%
    {\expanded\bgroup{{\iffalse}}\fi
     \expandafter\XINT_expr_scandec_a\romannumeral`&&@#1}%
```

32.11.1. Integral part (skipping zeroes)

1.2 has modified the code to give highest priority to digits, the accelerating impact is non-negligeable. I don't think the doubled \string is a serious penalty.

(reference to \string is obsolete: it is only used in the test but the tokens are not submitted to \string anymore)

```
\def\XINT_expr_scanint_main #1%
{%
    \ifcat \relax #1\expandafter\XINT_expr_scanint_hit_cs \fi
    \ifnum\xint_c_ix<1\string#1 \else\expandafter\XINT_expr_scanint_next\fi
    #1\XINT_expr_scanint_again
}%
\def\XINT_expr_scanint_again #1%
{%
    \expandafter\XINT_expr_scanint_main\romannumeral`&&@#1%
}%
```

1.4f had _getop here, but let's jump directly to _getop_a.

```
\def\XINT_expr_scanint_hit_cs \ifnum#1\fi#2\XINT_expr_scanint_again
{%
    \iffalse{{{\fi}}}\xint_c_xxii&&B#2%
}%
```

With 1.2d the tacit multiplication in front of a variable name or function name is now done with a higher precedence, intermediate between the common one of * and / and the one of ^. Thus x/2y is like x/(2y), but x^2y is like x^2*y and 2y! is not (2y)! but 2*y!.

Finally, 1.2d has moved away from the _scan macros all the business of the tacit multiplication in one unique place via \XINT_expr_getop. For this, the ending token is not first given to \string as was done earlier before handing over back control to \XINT_expr_getop. Earlier we had to identify the catcode 11 ! signaling a sub-expression here. With no \string applied we can do it in \XINT_expr_getop. As a corollary of this displacement, parsing of big numbers should be a tiny bit faster now.

Extended for 1.2l to ignore underscore character _ if encountered within digits; so it can serve as separator for better readability.

It is not obvious at 1.4 to support [] for three things: packing, slicing, ... and raw xintfrac syntax A/B[N]. The only good way would be to actually really separate completely \xintexpr, \xintfloatexpr and \xintiiexpr code which would allow to handle both / and [] from A/B[N] as we handle e and E. But triplicating the code is something I need to think about. It is not possible as in pre 1.4 to consider [only as an operator of same precedence as multiplication and division which was the way we did this, but we can use the technique of fake operators. Thus we intercept hitting a [here, which is not too much of a problem as anyhow we dropped temporarily 3*[1,2,3]+5 syntax so we don't have to worry that 3[1,2,3] should do tacit multiplication. I think only way in future will be to really separate the code of the three parsers (or drop entirely support for A/B[N]; as 1.4 has modified output of \xinteval to not use this notation this is not too dramatic).

Anyway we find a way to inject here the former handling of [N], which will use a delimited macro to directly fetch until the closing]. We do still need some fake operator because A/B[N] is (A/B) times 10^N and the /B is allowed to be missing. We hack this using the $ which is not used currently as operator elsewhere in the syntax and need to hook into \XINT_expr_getop_b. No finally I use the null char. It must be of catcode 12.

1.4f had _getop here, but let's jump directly to _getop_a.

```
\def\XINT_expr_scanint_next #1\XINT_expr_scanint_again
{%
    \if    [#1\xint_dothis\XINT_expr_rawxintfrac\fi
```

```
    \if    _#1\xint_dothis\XINT_expr_scanint_again\fi
    \if    e#1\xint_dothis{\iffalse{{\fi}}\xint_c_xxii&&C%
                           \the\numexpr0\XINT_expr_scanexp_a +}\fi
    \if    E#1\xint_dothis{\iffalse{{\fi}}\xint_c_xxii&&C%
                           \the\numexpr0\XINT_expr_scanexp_a +}\fi
    \if    .#1\xint_dothis{\XINT_expr_startdec_a .}\fi
    \xint_orthat
    {\iffalse{{{\fi}}}\xint_c_xxii&&B#1}%
}%
\def\XINT_expr_rawxintfrac
{%
    [0]\iffalse{{{\fi}}}\expandafter}%
       \csname XINT_expr_precedence_&&@\endcsname&&@%
}%
\def\XINT_expr_gobz_scanint_main #1%
{%
    \ifcat \relax #1\expandafter\XINT_expr_gobz_scanint_hit_cs\fi
    \ifnum\xint_c_x<1\string#1 \else\expandafter\XINT_expr_gobz_scanint_next\fi
    #1\XINT_expr_scanint_again
}%
\def\XINT_expr_gobz_scanint_again #1%
{%
    \expandafter\XINT_expr_gobz_scanint_main\romannumeral`&&@#1%
}%
```

1.4f had _getop here, but let's jump directly to _getop_a.

```
\def\XINT_expr_gobz_scanint_hit_cs\ifnum#1\fi#2\XINT_expr_scanint_again
{%
    0\iffalse{{{\fi}}}\xint_c_xxii&&B#2%
}%
\def\XINT_expr_gobz_scanint_next #1\XINT_expr_scanint_again
{%
    \if    [#1\xint_dothis{\expandafter0\XINT_expr_rawxintfrac}\fi
    \if    _#1\xint_dothis\XINT_expr_gobz_scanint_again\fi
    \if    e#1\xint_dothis{0\iffalse{{\fi}}\xint_c_xxii&&C%
                           \the\numexpr0\XINT_expr_scanexp_a +}\fi
    \if    E#1\xint_dothis{0\iffalse{{\fi}}\xint_c_xxii&&C%
                           \the\numexpr0\XINT_expr_scanexp_a +}\fi
    \if    .#1\xint_dothis{\XINT_expr_gobz_startdec_a .}\fi
    \if    0#1\xint_dothis\XINT_expr_gobz_scanint_again\fi
    \xint_orthat
    {0\iffalse{{{\fi}}}\xint_c_xxii&&B#1}%
}%
```

32.11.2. Fractional part

Annoying duplication of code to allow 0. as input.

1.2a corrects a very bad bug in 1.2 \XINT_expr_gobz_scandec_b which should have stripped leading zeroes in the fractional part but didn't; as a result \xinttheexpr 0.01\relax returned 0 =:-(((Thanks to Kroum Tzanev who reported the issue. Does it improve things if I say the bug was introduced in 1.2, it wasn't present before ?

1.4f had _getop here, but let's jump directly to _getop_a.

```
\def\XINT_expr_startdec_a .#1%
```

```
{%
    \expandafter\XINT_expr_scandec_a\romannumeral`&&@#1%
}%
\def\XINT_expr_scandec_a #1%
{%
    \if .#1\xint_dothis{\iffalse{{{\fi}}}\xint_c_xxii&&B..}\fi
    \xint_orthat {\XINT_expr_scandec_main 0.#1}%
}%
\def\XINT_expr_gobz_startdec_a .#1%
{%
    \expandafter\XINT_expr_gobz_scandec_a\romannumeral`&&@#1%
}%
\def\XINT_expr_gobz_scandec_a #1%
{%
    \if .#1\xint_dothis
    {0\iffalse{{{\fi}}}\xint_c_xxii&&B..}\fi
    \xint_orthat {\XINT_expr_gobz_scandec_main 0.#1}%
}%
\def\XINT_expr_scandec_main #1.#2%
{%
    \ifcat \relax #2\expandafter\XINT_expr_scandec_hit_cs\fi
    \ifnum\xint_c_ix<1\string#2 \else\expandafter\XINT_expr_scandec_next\fi
    #2\expandafter\XINT_expr_scandec_again\the\numexpr #1-\xint_c_i.%
}%
\def\XINT_expr_scandec_again #1.#2%
{%
    \expandafter\XINT_expr_scandec_main
    \the\numexpr #1\expandafter.\romannumeral`&&@#2%
}%
```

1.4f had _getop here, but let's jump directly to _getop_a.

```
\def\XINT_expr_scandec_hit_cs\ifnum#1\fi
    #2\expandafter\XINT_expr_scandec_again\the\numexpr#3-\xint_c_i.%
{%
    \iffalse{{{\fi}}}\xint_c_xxii&&C#3]#2%
}%
\def\XINT_expr_scandec_next #1#2\the\numexpr#3-\xint_c_i.%
{%
    \if    _#1\xint_dothis{\XINT_expr_scandec_again#3.}\fi
    \if    e#1\xint_dothis{\iffalse{{\fi}}\xint_c_xxii&&C%
                           \the\numexpr#3\XINT_expr_scanexp_a +}\fi
    \if    E#1\xint_dothis{\iffalse{{\fi}}\xint_c_xxii&&C%
                           \the\numexpr#3\XINT_expr_scanexp_a +}\fi
    \xint_orthat
    {\iffalse{{{\fi}}}\xint_c_xxii&&C#3]#1}%
}%
\def\XINT_expr_gobz_scandec_main #1.#2%
{%
    \ifcat \relax #2\expandafter\XINT_expr_gobz_scandec_hit_cs\fi
    \ifnum\xint_c_ix<1\string#2
          \else\expandafter\XINT_expr_gobz_scandec_next\fi
    \if0#2\expandafter\xint_firstoftwo\else\expandafter\xint_secondoftwo\fi
    {\expandafter\XINT_expr_gobz_scandec_main}%
```

```
    {#2\expandafter\XINT_expr_scandec_again}\the\numexpr#1-\xint_c_i.%
}%
```

1.4f had _getop here, but let's jump directly to _getop_a.

```
\def\XINT_expr_gobz_scandec_hit_cs \ifnum#1\fi\if0#2#3\xint_c_i.%
{%
    0\iffalse{{{\fi}}}\xint_c_xxii&&B#2%
}%
\def\XINT_expr_gobz_scandec_next\if0#1#2\fi #3\numexpr#4-\xint_c_i.%
{%
    \if    _#1\xint_dothis{\XINT_expr_gobz_scandec_main #4.}\fi
    \if    e#1\xint_dothis{0\iffalse{{\fi}}\xint_c_xxii&&C%
                          \the\numexpr0\XINT_expr_scanexp_a +}\fi
    \if    E#1\xint_dothis{0\iffalse{{\fi}}\xint_c_xxii&&C%
                          \the\numexpr0\XINT_expr_scanexp_a +}\fi
    \xint_orthat
    {0\iffalse{{{\fi}}}\xint_c_xxii&&B#1}%
}%
```

32.11.3. Scientific notation

Some pluses and minuses are allowed at the start of the scientific part, however not later, and no parenthesis.

ATTENTION! 1e\numexpr2+3\relax or 1e\xintiexpr i\relax, i=1..5 are not allowed and 1e1\numexpr2\relax does 1e1 * \numexpr2\relax. Use \the\numexpr, \xinttheiexpr, etc...

```
\def\XINT_expr_scanexp_a #1#2%
{%
    #1\expandafter\XINT_expr_scanexp_main\romannumeral`&&@#2%
}%
\def\XINT_expr_scanexp_main #1%
{%
    \ifcat \relax #1\expandafter\XINT_expr_scanexp_hit_cs\fi
    \ifnum\xint_c_ix<1\string#1 \else\expandafter\XINT_expr_scanexp_next\fi
    #1\XINT_expr_scanexp_again
}%
\def\XINT_expr_scanexp_again #1%
{%
    \expandafter\XINT_expr_scanexp_main_b\romannumeral`&&@#1%
}%
```

1.4f had _getop here, but let's jump directly to _getop_a.

```
\def\XINT_expr_scanexp_hit_cs\ifnum#1\fi#2\XINT_expr_scanexp_again
{%
    ]\iffalse{\fi}#2%
}%
\def\XINT_expr_scanexp_next #1\XINT_expr_scanexp_again
{%
    \if    _#1\xint_dothis  \XINT_expr_scanexp_again   \fi
    \if    +#1\xint_dothis {\XINT_expr_scanexp_a +}\fi
    \if    -#1\xint_dothis {\XINT_expr_scanexp_a -}\fi
    \xint_orthat
    {]\iffalse{\fi}#1}%
}%
\def\XINT_expr_scanexp_main_b #1%
```

```
{%
    \ifcat \relax #1\expandafter\XINT_expr_scanexp_hit_cs_b\fi
    \ifnum\xint_c_ix<1\string#1 \else\expandafter\XINT_expr_scanexp_next_b\fi
    #1\XINT_expr_scanexp_again_b
}%
```

1.4f had _getop here, but let's jump directly to _getop_a.

```
\def\XINT_expr_scanexp_hit_cs_b\ifnum#1\fi#2\XINT_expr_scanexp_again_b
{%
    ]\iffalse{\fi}#2%
}%
\def\XINT_expr_scanexp_again_b #1%
{%
    \expandafter\XINT_expr_scanexp_main_b\romannumeral`&&@#1%
}%
\def\XINT_expr_scanexp_next_b #1\XINT_expr_scanexp_again_b
{%
    \if _#1\xint_dothis\XINT_expr_scanexp_again\fi
    \xint_orthat
    {]\iffalse{\fi}#1}%
}%
```

32.11.4. Hexadecimal numbers

1.2d has moved most of the handling of tacit multiplication to \XINT_expr_getop, but we have to do some of it here, because we apply \string before calling \XINT_expr_scanhexI_aa. I do not insert the * in \XINT_expr_scanhexI_a, because it is its higher precedence variant which will is expected, to do the same as when a non-hexadecimal number prefixes a sub-expression. Tacit multiplication in front of variable or function names will not work (because of this \string).

Extended for 1.2l to ignore underscore character _ if encountered within digits.

(some above remarks have been obsoleted for some long time, no more applied \string since 1.4)

Notice that internal representation adds a [N] part only in case input used "DDD.dddd form, for compatibility with \xintiiexpr which is not compatible with such internal representation.

At 1.4g a very long-standing bug was fixed: input such as "\foo broke the parser because (incredibly) the \foo token was picked up unexpanded and ended up as is in an \ifcat !

Another long-standing bug was fixed at 1.4g: contrarily to the decimal case, here in the hexadecimal input leading zeros were not trimmed. This was ok, because formerly \xintHexToDec trimmed leading zeros, but at 1.2m 2017/07/31 xintbinhex.sty was modified and this ceased being the case. But I forgot to upgrade the parser here at that time. Leading zeros would in many circumstances (presence of a fractional part, or \xintiiexpr context) lead to wrong results. Leading zeros are now trimmed during input.

Sadly this fix forgot the case of integer part being zero itself. So \xinteval{"0.0001»} produced 0 because of this oversight. Fixed in un-released 1.4q, soon superseded by 2.0a1.

```
\def\XINT_expr_hex_in #1.#2#3;%
{%
    \expanded{{{\if#2>%
      \xintHexToDec{#1}}}\xint_c_xxii&&B%
    \else
      \expandafter\CALX_rdx\the\numexpr
    \XINT_length_loop
    #3\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:
       \xint_c_viii\xint_c_vii\xint_c_vi\xint_c_v
       \xint_c_iv\xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye
```

```
    \relax.{#1#3}%
        \xinthextodec{100000625}4%
    \fi}%
}%
```

Syntax (inside the \expanded of the scanner exits; A the digits of the integer and fractional parts, k the number of fractional digits):
\CALX_rdx k.{A}<conv>{b}<j> -> <A'5^(jk)>}}\xint_c_xxii<C><-jk>]
Fractional literal in radix 2^j (j = 4, 3, 1: hexadecimal, octal, binary). A' = <conv>{A} (\xinthextodec, \xintocttodec, \xintbintodec) is the integer of all the digits; the value is A' 2^(-jk) = A' 5^(jk) 10^(-jk). <b> is the native block of 5^j (100000625, 100000125, 100000005): A' 5^(jk) is calxint's \CALX_npowmul. The loop of \xintiNum (\XINT_num_loop, \XINT_num_cleanup) removes the leading zeros of A' (there are some when the integer part is zero: the conversions keep leading zeros of their input). The closing braces come from `\iffalse{{\fi}}`, as in the other exits of the scanner; <C> is the fake operator (character 3) of the decimal and exponent exits.

```
\def\CALX_rdx #1.#2#3#4#5%
{%
    \romannumeral0\CALX_npowmul#1.{#4}%
      {\expandafter\XINT_num_cleanup
       \the\numexpr\expandafter\XINT_num_loop\romannumeral0#3{#2}%
       \xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\Z}%
    \iffalse{{\fi}}\xint_c_xxii&&C\the\numexpr-#5*#1]%
}%
```

Let's not forget to grab-expand next token first as is normal rule of operation. Formerly called \XINT_expr_scanhex_I and had " upfront.

```
\def\XINT_expr_starthex #1%
{%
    \expandafter\XINT_expr_hex_in\expanded\bgroup
    \expandafter\XINT_expr_scanhexIgobz_a\romannumeral`&&@#1%
}%
\def\XINT_expr_scanhexIgobz_a #1%
{%
    \ifcat #1\relax
      0.>;\iffalse{\fi\expandafter}\expandafter\xint_gobble_i\fi
    \XINT_expr_scanhexIgobz_aa #1%
}%
\def\XINT_expr_scanhexIgobz_aa #1%
{%
    \if\ifnum`#1>`0
       \ifnum`#1>`9
       \ifnum`#1>`@
       \ifnum`#1>`F
       0\else1\fi\else0\fi\else1\fi\else0\fi 1%
       \xint_dothis\XINT_expr_scanhexI_b
    \fi
    \if 0#1\xint_dothis\XINT_expr_scanhexIgobz_bgob\fi
    \if _#1\xint_dothis\XINT_expr_scanhexIgobz_bgob\fi
    \if .#1\xint_dothis\XINT_expr_scanhexIgobz_toII\fi
    \xint_orthat
     {\XINT_expandableerror
        {Expected hexadecimal digit, `_', or `.'. got `#1'. Using `0'.}%
      0.>;\iffalse{\fi}}%
```

```
    #1%
}%
\def\XINT_expr_scanhexIgobz_bgob #1#2%
{%
    \expandafter\XINT_expr_scanhexIgobz_a\romannumeral`&&@#2%
}%
\def\XINT_expr_scanhexIgobz_toII .#1%
{%
    0..\expandafter\XINT_expr_scanhexII_a\romannumeral`&&@#1%
}%
\def\XINT_expr_scanhexI_a #1%
{%
    \ifcat #1\relax
      .>;\iffalse{\fi\expandafter}\expandafter\xint_gobble_i\fi
    \XINT_expr_scanhexI_aa #1%
}%
\def\XINT_expr_scanhexI_aa #1%
{%
    \if\ifnum`#1>`/
       \ifnum`#1>`9
       \ifnum`#1>`@
       \ifnum`#1>`F
       0\else1\fi\else0\fi\else1\fi\else0\fi 1%
       \expandafter\XINT_expr_scanhexI_b
    \else
       \if _#1\xint_dothis{\expandafter\XINT_expr_scanhexI_bgob}\fi
       \if .#1\xint_dothis{\expandafter\XINT_expr_scanhexI_toII}\fi
       \xint_orthat {.>;\iffalse{\fi\expandafter}}%
    \fi
    #1%
}%
\def\XINT_expr_scanhexI_b #1#2%
{%
    #1\expandafter\XINT_expr_scanhexI_a\romannumeral`&&@#2%
}%
\def\XINT_expr_scanhexI_bgob #1#2%
{%
    \expandafter\XINT_expr_scanhexI_a\romannumeral`&&@#2%
}%
\def\XINT_expr_scanhexI_toII .#1%
{%
    ..\expandafter\XINT_expr_scanhexII_a\romannumeral`&&@#1%
}%
\def\XINT_expr_scanhexII_a #1%
{%
    \ifcat #1\relax\xint_dothis{;\iffalse{\fi}#1}\fi
    \xint_orthat {\XINT_expr_scanhexII_aa #1}%
}%
\def\XINT_expr_scanhexII_aa #1%
{%
    \if\ifnum`#1>`/
       \ifnum`#1>`9
```

```
        \ifnum`#1>`@
        \ifnum`#1>`F
        0\else1\fi\else0\fi\else1\fi\else0\fi 1%
        \expandafter\XINT_expr_scanhexII_b
    \else
        \if _#1\xint_dothis{\expandafter\XINT_expr_scanhexII_bgob}\fi
        \xint_orthat{;\iffalse{\fi\expandafter}}%
    \fi
    #1%
}%
\def\XINT_expr_scanhexII_b #1#2%
{%
    #1\expandafter\XINT_expr_scanhexII_a\romannumeral`&&@#2%
}%
\def\XINT_expr_scanhexII_bgob #1#2%
{%
    \expandafter\XINT_expr_scanhexII_a\romannumeral`&&@#2%
}%
```

32.11.5. Octal numbers

Added at 1.4n (2025/09/05).

The parsing goes exactly as with hexadecimal inputs, except that we require an octal digit when expected.

TODO: think about perhaps sharing code between binary, octal, and hexadecimal.

```
\def\XINT_expr_oct_in #1.#2#3;%
{%
    \expanded{{{\if#2>%
      \xintOctToDec{#1}}}\xint_c_xxii&&B%
    \else
      \expandafter\CALX_rdx\the\numexpr
    \XINT_length_loop
    #3\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:
       \xint_c_viii\xint_c_vii\xint_c_vi\xint_c_v
       \xint_c_iv\xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye
    \relax.{#1#3}%
        \xintocttodec{100000125}3%
    \fi}%
}%
\def\XINT_expr_startoct #1%
{%
    \expandafter\XINT_expr_oct_in\expanded\bgroup
    \expandafter\XINT_expr_scanoctIgobz_a\romannumeral`&&@#1%
}%
\def\XINT_expr_scanoctIgobz_a #1%
{%
    \ifcat #1\relax
      0.>;\iffalse{\fi\expandafter}\expandafter\xint_gobble_i\fi
    \XINT_expr_scanoctIgobz_aa #1%
}%
```

Wondering if a bunch of \if tests comparing with 0 to 7 would not be more efficient here. But testing would take some time I am not motivated enough to use for that. In the code, also added at

1.4n for parsing binary input, I do this, but of course then it was rather more natural to do. But I did not test either comparing with style as here.

```
\def\XINT_expr_scanoctIgobz_aa #1%
{%
    \if\ifnum`#1>`0 \ifnum`#1>`7 0\else1\fi\else0\fi 1%
       \xint_dothis\XINT_expr_scanoctI_b
    \fi
    \if 0#1\xint_dothis\XINT_expr_scanoctIgobz_bgob\fi
    \if _#1\xint_dothis\XINT_expr_scanoctIgobz_bgob\fi
    \if .#1\xint_dothis\XINT_expr_scanoctIgobz_toII\fi
    \xint_orthat
     {\XINT_expandableerror
        {Expected an octal digit, `_', or `.'.  Got `#1'. Using `0'.}%
      0.>;\iffalse{\fi}}%
    #1%
}%
\def\XINT_expr_scanoctIgobz_bgob #1#2%
{%
    \expandafter\XINT_expr_scanoctIgobz_a\romannumeral`&&@#2%
}%
\def\XINT_expr_scanoctIgobz_toII .#1%
{%
    0..\expandafter\XINT_expr_scanoctII_a\romannumeral`&&@#1%
}%
\def\XINT_expr_scanoctI_a #1%
{%
    \ifcat #1\relax
      .>;\iffalse{\fi\expandafter}\expandafter\xint_gobble_i\fi
    \XINT_expr_scanoctI_aa #1%
}%
\def\XINT_expr_scanoctI_aa #1%
{%
    \if\ifnum`#1>`/ \ifnum`#1>`7 0\else1\fi\else0\fi 1%
       \expandafter\XINT_expr_scanoctI_b
    \else
       \if _#1\xint_dothis{\expandafter\XINT_expr_scanoctI_bgob}\fi
       \if .#1\xint_dothis{\expandafter\XINT_expr_scanoctI_toII}\fi
       \xint_orthat {.>;\iffalse{\fi\expandafter}}%
    \fi
    #1%
}%
\def\XINT_expr_scanoctI_b #1#2%
{%
    #1\expandafter\XINT_expr_scanoctI_a\romannumeral`&&@#2%
}%
\def\XINT_expr_scanoctI_bgob #1#2%
{%
    \expandafter\XINT_expr_scanoctI_a\romannumeral`&&@#2%
}%
\def\XINT_expr_scanoctI_toII .#1%
{%
    ..\expandafter\XINT_expr_scanoctII_a\romannumeral`&&@#1%
```

```
}%
\def\XINT_expr_scanoctII_a #1%
{%
    \ifcat #1\relax\xint_dothis{;\iffalse{\fi}#1}\fi
    \xint_orthat {\XINT_expr_scanoctII_aa #1}%
}%
\def\XINT_expr_scanoctII_aa #1%
{%
    \if\ifnum`#1>`/ \ifnum`#1>`7 0\else1\fi\else0\fi 1%
       \expandafter\XINT_expr_scanoctII_b
    \else
       \if _#1\xint_dothis{\expandafter\XINT_expr_scanoctII_bgob}\fi
       \xint_orthat{;\iffalse{\fi\expandafter}}%
    \fi
    #1%
}%
\def\XINT_expr_scanoctII_b #1#2%
{%
    #1\expandafter\XINT_expr_scanoctII_a\romannumeral`&&@#2%
}%
\def\XINT_expr_scanoctII_bgob #1#2%
{%
    \expandafter\XINT_expr_scanoctII_a\romannumeral`&&@#2%
}%
```

32.11.6. Binary numbers

Added at 1.4n (2025/09/05). Analogous to hexadecimal or octal with some simpler tests for digits.

```
\def\XINT_expr_bin_in #1.#2#3;%
{%
    \expanded{{{\if#2>%
      \xintBinToDec{#1}}}\xint_c_xxii&&B%
    \else
      \expandafter\CALX_rdx\the\numexpr
    \XINT_length_loop
    #3\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:
       \xint_c_viii\xint_c_vii\xint_c_vi\xint_c_v
       \xint_c_iv\xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye
    \relax.{#1#3}%
        \xintbintodec{100000005}1%
    \fi}%
}%
\def\XINT_expr_startbin #1%
{%
    \expandafter\XINT_expr_bin_in\expanded\bgroup
    \expandafter\XINT_expr_scanbinIgobz_a\romannumeral`&&@#1%
}%
\def\XINT_expr_scanbinIgobz_a #1%
{%
    \ifcat #1\relax
      0.>;\iffalse{\fi\expandafter}\expandafter\xint_gobble_i\fi
    \XINT_expr_scanbinIgobz_aa #1%
```

```
}%
\def\XINT_expr_scanbinIgobz_aa #1%
{%
    \if _#1\xint_dothis\XINT_expr_scanbinIgobz_bgob\fi
    \if .#1\xint_dothis\XINT_expr_scanbinIgobz_toII\fi
    \if 0#1\xint_dothis\XINT_expr_scanbinIgobz_bgob\fi
    \if 1#1\xint_dothis\XINT_expr_scanbinI_b\fi
    \xint_orthat
     {\XINT_expandableerror
        {Expected a binary digit, `_', or `.'. Got `#1'. Using `0'.}%
      0.>;\iffalse{\fi}}%
    #1%
}%
\def\XINT_expr_scanbinIgobz_bgob #1#2%
{%
    \expandafter\XINT_expr_scanbinIgobz_a\romannumeral`&&@#2%
}%
\def\XINT_expr_scanbinIgobz_toII .#1%
{%
    0..\expandafter\XINT_expr_scanbinII_a\romannumeral`&&@#1%
}%
\def\XINT_expr_scanbinI_a #1%
{%
    \ifcat #1\relax
      .>;\iffalse{\fi\expandafter}\expandafter\xint_gobble_i\fi
    \XINT_expr_scanbinI_aa #1%
}%
\def\XINT_expr_scanbinI_aa #1%
{%
    \if _#1\xint_dothis\XINT_expr_scanbinI_bgob\fi
    \if .#1\xint_dothis\XINT_expr_scanbinI_toII\fi
    \if 0#1\xint_dothis\XINT_expr_scanbinI_b\fi
    \if 1#1\xint_dothis\XINT_expr_scanbinI_b\fi
    \xint_orthat {.>;\iffalse{\fi}}%
    #1%
}%
\def\XINT_expr_scanbinI_b #1#2%
{%
    #1\expandafter\XINT_expr_scanbinI_a\romannumeral`&&@#2%
}%
\def\XINT_expr_scanbinI_bgob #1#2%
{%
    \expandafter\XINT_expr_scanbinI_a\romannumeral`&&@#2%
}%
\def\XINT_expr_scanbinI_toII .#1%
{%
    ..\expandafter\XINT_expr_scanbinII_a\romannumeral`&&@#1%
}%
\def\XINT_expr_scanbinII_a #1%
{%
    \ifcat #1\relax\xint_dothis{;\iffalse{\fi}#1}\fi
    \xint_orthat {\XINT_expr_scanbinII_aa #1}%
```

```
}%
\def\XINT_expr_scanbinII_aa #1%
{%
    \if _#1\xint_dothis\XINT_expr_scanbinII_bgob\fi
    \if 0#1\xint_dothis\XINT_expr_scanbinII_b\fi
    \if 1#1\xint_dothis\XINT_expr_scanbinII_b\fi
    \xint_orthat{;\iffalse{\fi}}%
    #1%
}%
\def\XINT_expr_scanbinII_b #1#2%
{%
    #1\expandafter\XINT_expr_scanbinII_a\romannumeral`&&@#2%
}%
\def\XINT_expr_scanbinII_bgob #1#2%
{%
    \expandafter\XINT_expr_scanbinII_a\romannumeral`&&@#2%
}%
```

32.11.7. \XINT_expr_startfunc: collecting names of functions and variables

At 1.4 the first token left over has not been submitted to \string. We also know it is not a control sequence. So we can test catcode to identify if operator is found. And it is allowed to hit some operator such as a closing parenthesis we will then insert the «nil» value (edited: which however will cause certain breakage of the infix binary operators: I notice I did not insert None {{}} but nil {}, perhaps by oversight).

There was prior to 1.4 solely the dispatch in \XINT_expr_scanfunc_b but now we do it immediately and issue \XINT_expr_func only in certain cases.

Comments here have been removed because 1.4g did a refactoring and renamed \XINT_expr_scanfunc to \XINT_expr_startfunc, moving half of it earlier inside the getnextfork macros.

```
\def\XINT_expr_startfunc #1%
   {\expandafter\XINT_expr_func\expanded\bgroup#1\XINT_expr_scanfunc_a}%
\def\XINT_expr_scanfunc_a #1%
{%
    \expandafter\XINT_expr_scanfunc_b\romannumeral`&&@#1%
}%
```

This handles: 1) (indirectly) tacit multiplication by a variable in front a of sub-expression, 2) (indirectly) tacit multiplication in front of a \count etc..., 3) functions which are recognized via an encountered opening parenthesis (but later this must be disambiguated from variables with tacit multiplication) 4) 5) 6) 7) acceptable components of a variable or function names: @, underscore, digits, letters (or chars of category code letter.)

The short lived 1.2d which followed the even shorter lived 1.2c managed to introduce a bug here as it removed the check for catcode 11 !, which must be recognized if ! is not to be taken as part of a variable name. Don't know what I was thinking, it was the time when I was moving the handling of tacit mutliplication entirely to the \XINT_expr_getop side. Fixed in 1.2e.

I almost decided to remove the \ifcat\relax test whose rôle is to avoid the \string#1 to do something bad is the escape char is a digit! Perhaps I will remove it at some point ! I truly almost did it, but also the case of no escape char is a problem (\string\0, if \0 is a count ...)

The (indirectly) above means that via \XINT_expr_func then \XINT_expr_op__ one goes back to \XINT_expr_getop then \XINT_expr_getop_b which is the location where tacit multiplication is now centralized. This makes the treatment of tacit multiplication for situations such as <variable>\count or <variable>\xintexpr..\relax, perhaps a bit sub-optimal, but first the variable name must be gathered, second the variable must expand to its value.

```
\def\XINT_expr_scanfunc_b #1%
{%
  \ifcat \relax#1\xint_dothis{\iffalse{\fi}(_#1}\fi
  \if (#1\xint_dothis{\iffalse{\fi}(`}\fi
  \if 1\ifcat a#10\fi
       \ifnum\xint_c_ix<1\string#1 0\fi
       \if @#10\fi
       \if _#10\fi
      1%
      \xint_dothis{\iffalse{\fi}(_#1}\fi
  \xint_orthat {#1\XINT_expr_scanfunc_a}%
}%
```

32.11.8. \XINT_expr_func: dispatch to variable replacement or to function execution

Comments written 2015/11/12: earlier there was an \ifcsname test for checking if we had a variable in front of a (, for tacit multiplication for example in x(y+z(x+w)) to work. But after I had implemented functions (that was yesterday...), I had the problem if was impossible to re-declare a variable name such as "f" as a function name. The problem is that here we can not test if the function is available because we don't know if we are in expr, iiexpr or floatexpr. The \xint_c_ii^v causes all fetching operations to stop and control is handed over to the routines which will be expr, iiexpr ou floatexpr specific, i.e. the \XINT_{expr|iiexpr|flexpr}_op_{`|_} which are invoked by the until_<op>_b macros earlier in the stream. Functions may exist for one but not the two other parsers. Variables are declared via one parser and usable in the others, but naturally \xintiiexpr has its restrictions.

Thinking about this again I decided to treat a priori cases such as x(...) as functions, after having assigned to each variable a low-weight macro which will convert this into _getop\.=<value of x>*(...). To activate that macro at the right time I could for this exploit the "onliteral" intercept, which is parser independent (1.2c).

This led to me necessarily to rewrite partially the seq, add, mul, subs, iter ... routines as now the variables fetch only one token. I think the thing is more efficient.

1.2c had \def\XINT_expr_func #1(#2{\xint_c_ii^v #2{#1}}

In \XINT_expr_func the #2 is _ if #1 must be a variable name, or #2=` if #1 must be either a function name or possibly a variable name which will then have to be followed by tacit multiplication before the opening parenthesis.

The \xint_c_ii^v is there because _op_` must know in which parser it works. Dispendious for _. Hence I modify for 1.2d.

```
\def\XINT_expr_func #1(#2{\if _#2\xint_dothis{\XINT_expr_op__{#1}}\fi
                          \xint_orthat{{#1}\xint_c_ii^v #2}}%
```

32.12. \XINT_expr_op_`: launch function or pseudo-function, or evaluate variable and insert operator of multiplication in front of parenthesized contents

The "onliteral" intercepts is for bool, togl, protect, ... but also for add, mul, seq, etc... Genuine functions have expr, iiexpr and flexpr versions (or only one or two of the three) and trigger here the use of the suitable parser-dependant form. The former (pseudo functions and functions handling dummy variables) first trigger a parser independent mechanism.

With 1.2c "onliteral" is also used to disambiguate a variable followed by an opening parenthesis from a function and then apply tacit multiplication. However as I use only a \ifcsname test, in order to be able to re-define a variable as function, I move the check for being a function first. Each variable name now has its onliteral_<name> associated macro. This used to be decided much earlier at the time of \XINT_expr_func.

The advantage of 1.2c code is that the same name can be used for a variable or a function.

Modified at 1.4i (2021/06/11). The 1.2c abuse of «onliteral» for both tacit multiplication in front of an opening parenthesis and «generic» functions or pseudo-functions meant that the latter were vulnerable against user redefinition of a function name as a variable name. This applied to subs, subsm, subsn, seq, add, mul, ndseq, ndmap, ndfillraw, bool, togl, protect, qint, qfrac, qfloat, qraw, random, qrand, rbit and the most susceptible in real life was probably "seq".

Now variables have an associated «var*» named macro, not «onliteral».

In passing I refactor here in a \romannumeral inspired way how \csname and TeX booleans are intertwined, minimizing \expandafter usage.

Modified at 2.0a1 (2026/11/01). Per-parser "onliteral" macros.

```
\def\XINT_tmpa #1#2#3{%
  \def #1##1%
  {%
    \csname
      XINT_\ifcsname XINT_#3_func_##1\endcsname
        #3_func_##1\expandafter\endcsname\romannumeral`&&@\expandafter#2%
      \romannumeral\else
      \ifcsname XINT_#3_onliteral_##1\endcsname
        #3_onliteral_##1\expandafter\endcsname\romannumeral
      \else
      \ifcsname XINT_expr_var*_##1\endcsname
        expr_var*_##1\expandafter\endcsname\romannumeral
      \else
        #3_func_\XINT_expr_unknown_function {##1}%
                \expandafter\endcsname\romannumeral`&&@\expandafter#2%
      \romannumeral
      \fi\fi\fi\xint_c_
  }%
}%
\xintFor #1 in {flexpr,iiexpr} \do {%
     \expandafter\XINT_tmpa
                  \csname XINT_#1_op_`\expandafter\endcsname
                  \csname XINT_#1_oparen\endcsname
                  {#1}%
}%
\def\XINT_expr_unknown_function #1%
   {\XINT_expandableerror{`#1' is unknown, say `Isome_func' or I use 0.}}%
\def\XINT_expr_func_ #1#2#3{#1#2{{0}}}%
\let\XINT_flexpr_func_\XINT_expr_func_
\let\XINT_iiexpr_func_\XINT_expr_func_
```

32.13. \XINT_expr_op__: replace a variable by its value and then fetch next operator

The 1.1 mechanism for \XINT_expr_var_<varname> has been modified in 1.2c. The <varname> associated macro is now only expanded once, not twice. We arrive here via \XINT_expr_func.

At 1.4 \XINT_expr_getop is launched with accumulated result on its left. But the omit and abort keywords are implemented via fake variables which rely on possibility to modify incoming upfront tokens. If we did here something such as

```
_var_#1\expandafter\endcsname\romannumeral`^^@\XINT_expr_getop
```

the premature expansion of getop would break the var_omit and var_abort mechanism. Thus we revert to former code which locates an \XINT_expr_getop (call it _legacy) before the tokens from

the variable expansion (in xintexpr < 1.4 the normal variables expanded to a single token so the overhead was not serious) so we can expand fake variables first.

Abusing variables to manipulate the incoming token stream is a bit bad, usually I prefer functions for this (such as the break() function) but then I have to define 3 macros for the 3 parsers.

This trick of fake variables puts thus a general overhead at various locations, and the situation here is REALLY not satisfactory. But 1.4 has (had) to be released now.

Even if I could put the \csname XINT_expr_var_foo\endcsname upfront, which would then be f-expanded, this would still need \XINT_expr_put_op_first to use its \expandafter's as long as \XINT_expr_var_foo expands to {\XINT_expr_varvalue_foo} with a not-yet expanded \XINT_expr_var_value.

I could let \XINT_expr_var_foo expand to \expandafter{\XINT_expr_varvalue_foo} allowing then (if it gets f-expanded) probably to drop the \expandafter in \XINT_expr_put_op_first. But I can not consider this option in the form

```
_var_foo\expandafter\endcsname\romannumeral`^^@\XINT_expr_getop
```

until the issue with fake variables such as omit and abort which must act before \XINT_expr_getop has some workaround. This could be implemented here with some extra branch, i.e. there would not be some \XINT_expr_var_omit but something else filtered out in the \else branch here.

The above comments mention only omit and abort, but the case of real dummy variables also needs consideration.

At 1.4g, I test first for existence of \XINT_expr_onliteral_foo.

Updated for 1.4i: now rather existence of \XINT_expr_var*_foo is tested.

This is a trick which allows to distinguish actual or dummy variables from really fake variables omit and abort (must check if there are others). For the real or dummy variables we can trigger the expansion of the \XINT_expr_getop before the one of the variable. I could test vor varvalue_foo but this applies only to real variables not dummy variables. Actual and dummy variables are thus handled slightly faster at 1.4g as there is less induced moving around (the \expandafter chain in \XINT_expr_put_op_first still applies at this stage, as I have not yet re-examined the var/varlue mechanism). And the test for var_foo is moved directly inside the \csname construct in the \else branch which now handles together fake variables and non-existing variables.

I only have to make sure dummy variables are really safe being handled this way with the getop action having being done before they expand, but it looks ok. Attention it is crucial that if \XINT_expr_getop finds a \relax it inserts \xint_c_\relax so the \relax token is still there!

With this refactoring the \XINT_expr_getop_legacy is applied only in case of non-existent variables or fake variables omit/abort or things such as nil, None, false, true, False, True.

If user in interactive mode fixes the variable name, the \XINT_expr_var_foo expanded once with deliver {\XINT_expr_varvalue_foo} (if not dummy), and the braces are maintained by \XINT_expr_getop_legacy.

```
\def\XINT_expr_op__  #1% op__ with two _'s
{%
    \ifcsname XINT_expr_var*_#1\endcsname
       \csname XINT_expr_var_#1\expandafter\endcsname
       \romannumeral`&&@\expandafter\XINT_expr_getop
    \else
      \expandafter\expandafter\expandafter\XINT_expr_getop_legacy
      \csname XINT_expr_var_%
              \ifcsname XINT_expr_var_#1\endcsname#1%
              \else\XINT_expr_unknown_variable{#1}\fi
      \expandafter\endcsname
    \fi
}%
\def\XINT_expr_unknown_variable #1%
   {\XINT_expandableerror {`#1' unknown, say `Isome_var' or I use 0.}}%
```

```
\def\XINT_expr_var_{{{0}}}%
\let\XINT_flexpr_op__ \XINT_expr_op__
\let\XINT_iiexpr_op__ \XINT_expr_op__
\def\XINT_expr_getop_legacy #1%
{%
    \expanded{\xint_noxpd{{#1}}\expandafter}\romannumeral`&&@\XINT_expr_getop
}%
```

32.14. \XINT_expr_getop: fetch the next operator or closing parenthesis or end of expression

Release 1.1 implements multi-character operators.

1.2d adds tacit mutiplication also in front of variable or functions names starting with a letter, not only a @ or a _ as was already the case. This is for (x+y)z situations. It also applies higher precedence in cases like x/2y or x/2@, or x/2max(3,5), or x/2\xintexpr 3\relax.

In fact, finally I decide that all sorts of tacit multiplication will always use the higher precedence.

Indeed I hesitated somewhat: with the current code one does not know if \XINT_expr_getop as invoked after a closing parenthesis or because a number parsing ended, and I felt distinguishing the two was unneeded extra stuff. This means cases like (a+b)/(c+d)(e+f) will first multiply the last two parenthesized terms.

1.2q adds tacit multiplication in cases such as (1+1)3 or 5!7!

1.4 has simplified coding here as \XINT_expr_getop expansion happens at a time when a fetched value has already being stored.

Prior to 1.4g there was an \if _#1\xint_dothis\xint_secondofthree\fi because the _ can be used to start names, for private use by package (for example by polexpr). But this test was silly because these usages are only with a _ of catcode 11. And allowing non-catcode 11 _ also to trigger tacit multiplication caused an infinite loop in collaboration with \XINT_expr_scanfunc, see explanations there (now removed after refactoring, see \XINT_expr_startfunc).

The situation with the @ is different because we must allow it even as catcode 12 as a name, as it used in the syntax and must work the same if of catcode 11 or 12. No infinite loop because it is filtered out by one of the \XINT_expr_getnextfork macros.

The check for : to send it to thirdofthree "getop" branch is needed, last time I checked, because during some part of at least \xintdeffunc, some scantokens are done which need to work with the : of catcode 11, and it would be misconstrued to start a name if not filtered out.

```
\def\XINT_expr_getop #1%
{%
    \expandafter\XINT_expr_getop_a\romannumeral`&&@#1%
}%
\catcode`* 11
\def\XINT_expr_getop_a #1%
{%
    \ifx   \relax #1\xint_dothis\xint_firstofthree\fi
    \ifcat \relax #1\xint_dothis\xint_secondofthree\fi
    \ifnum\xint_c_ix<1\string#1 \xint_dothis\xint_secondofthree\fi
    \if    :#1\xint_dothis      \xint_thirdofthree\fi
    \if    @#1\xint_dothis      \xint_secondofthree\fi
    \if    (#1\xint_dothis      \xint_secondofthree\fi %)
    \ifcat a#1\xint_dothis      \xint_secondofthree\fi
    \xint_orthat \xint_thirdofthree
```

Formerly \XINT_expr_foundend as firstofthree but at 1.4g let's simply insert \xint_c_ as the #1 is \relax (and anyhow a place-holder according to remark in definition of \XINT_expr_foundend

```
    \xint_c_
```

Tacit multiplication with higher precedence. Formerly \XINT_expr_precedence_*** was used, renamed to \XINT_expr_prec_tacit at 1.4g in case a backport is done of the \bnumdefinfix from bnumexpr.

```
    {\XINT_expr_prec_tacit *}%
```

This is only location which jumps to \XINT_expr_getop_b. At 1.4f and perhaps for old legacy reasons this was \expandafter\XINT_expr_getop_b \string#1 but I see no reason now for applying \string to #1. Removed at 1.4g. And the #1 now moved out of the secondofthree and thirdofthree branches.

```
    \XINT_expr_getop_b
    #1%
}%
\catcode`* 12
```

\relax is a place holder here. At 1.4g, we don't use \XINT_expr_foundend anymore in \XINT_expr_getop_a which was slightly refactored, but it is used elsewhere.

Attention that keeping a \relax around if \XINT_expr_getop hits it is crucial to good functioning of dummy variables after 1.4g refactoring of \XINT_expr_op__, the \relax being used as delimiter by dummy variables, and \XINT_expr_getop is now expanded before the variable itself does its thing.

```
\def\XINT_expr_foundend {\xint_c_ \relax}%
```

? is a very special operator with top precedence which will check if the next token is another ?, while avoiding removing a brace pair from token stream due to its syntax. Pre 1.1 releases used : rather than ??, but we need : for Python like slices of lists.

null char is used as hack to implement A/B[N] raw input at 1.4. See also \XINT_expr_scanint_c.

Memo: 1.4g, the token fetched by \XINT_expr_getop_b has not anymore been previously submitted in \XINT_expr_getop_a to \string.

```
\def\XINT_expr_getop_b#1{\def\XINT_expr_getop_b ##1%
{%
     \if &&@##1\xint_dothis{#1&&@}\fi
     \if   '##1\xint_dothis{\XINT_expr_binopwrd }\fi
     \if   ?##1\xint_dothis{\XINT_expr_precedence_? ?}\fi
     \xint_orthat          {\XINT_expr_scanop_a ##1}%
}}\expandafter\XINT_expr_getop_b\csname XINT_expr_precedence_&&@\endcsname
\def\XINT_expr_binopwrd #1'%
{%
    \expandafter\XINT_expr_foundop_a
    \csname XINT_expr_itself_\xint_zapspaces #1 \xint_gobble_i\endcsname
}%
\def\XINT_expr_scanop_a #1#2%
{%
    \expandafter\XINT_expr_scanop_b\expandafter#1\romannumeral`&&@#2%
}%
```

Multi-character operators have an associated itself macro at each stage of decomposition starting at two characters. Here, nothing imposes to the operator characters not to be of catcode letter, this constraint applies only on the first character and is done via \XINT_expr_getop_a, to handle in particular tacit multiplication in front of variable or function names.

But it would be dangerous to allow letters in operator characters, again due to existence of variables and functions, and anyhow there is no user interface to add such custom operators. However in bnumexpr, such a constraint does not exist.

I don't worry too much about efficiency here... and at 1.4g I have re-written for code readability only. Once we see that #1#2 is not a candidate to be or start an operator, we need to check

if single-character operator #1 is really an operator and this is done via the existence of the precedence token.

Unfortunately the 1.4g refactoring of the scanop macros had a bad bug: \XINT_expr_scanop_c inserted \romannumeral`^^@ in stream but did not grab a token first so a space would stop the \romannumeral and then the #2 in \XINT_expr_scanop_d was not pre-expanded and ended up alone in \ifcat. It is too distant in the past the time when I wrote the core of xintexpr in 2013... older and dumber now.

```
\def\XINT_expr_scanop_b #1#2%
{%
    \unless\ifcat#2\relax
           \ifcsname XINT_expr_itself_#1#2\endcsname
           \XINT_expr_scanop_c
    \fi\fi
    \XINT_expr_foundop_a #1#2%
}%
\def\XINT_expr_scanop_c #1#2#3#4#5#6% #1#2=\fi\fi
{%
    #1#2%
    \expandafter\XINT_expr_scanop_d
                \csname XINT_expr_itself_#4#5\expandafter\endcsname
    \romannumeral`&&@#6%
}%
\def\XINT_expr_scanop_d #1#2%
{%
    \unless\ifcat#2\relax
           \ifcsname XINT_expr_itself_#1#2\endcsname
           \XINT_expr_scanop_c
    \fi\fi
    \XINT_expr_foundop #1#2%
}%
\def\XINT_expr_foundop_a #1%
{%
    \ifcsname XINT_expr_precedence_#1\endcsname
        \csname XINT_expr_precedence_#1\expandafter\endcsname
        \expandafter #1%
    \else
        \expandafter\XINT_expr_getop\romannumeral`&&@%
        \xint_afterfi{\XINT_expandableerror
        {Expected an operator but got `#1'. Ignoring.}}%
     \fi
}%
\def\XINT_expr_foundop #1{\csname XINT_expr_precedence_#1\endcsname #1}%
```

32.15. Expansion spanning; opening and closing parentheses

This is also where \XINT_iiexpr_start and \XINT_flexpr_start are defined.

These comments apply to all definitions coming next relative to execution of operations from parsing of syntax.

Refactored (and unified) at 1.4. In particular the 1.4 scheme uses op, exec, check-, and checkp. Formerly it was until_a (check-) and until_b (now split into checkp and exec).

This way neither check- nor checkp have to grab the accumulated number so far (top of stack if you like) and besides one never has to go back to check- from checkp (and neither from check-).

Prior to 1.4, accumulated intermediate results were stored as one token, but now we have to use \expanded to propagate expansion beyond possibly arbitrary long braced nested data. With the 1.4 refactoring we do this only once and only grab a second time the data if we actually have to act upon it.

Version 1.1 had a hack inside the until macros for handling the omit and abort in iterations over dummy variables. This has been removed by 1.2c, see the subsection where omit and abort are discussed.

Exceptionally, the check- is here abbreviated to check.

```
\catcode`) 11
\def\XINT_def_start_and_check #1#2#3#4#5#6%
{%
    \def#1% \XINT_expr_start, \XINT_iiexpr_start, \XINT_flexpr_start
    {%
        \expandafter#2\romannumeral`&&@\XINT_expr_getnext
    }%
    \def#2##1% check
    {%
        \xint_UDsignfork
          ##1{\expandafter#3\romannumeral`&&@#4}%
            -{#3##1}%
        \krof
    }%
    \def#3##1##2% checkp
    {%
        \ifcase ##1%
           \expandafter\XINT_expr_done
        \or\expandafter#5%
        \else
           \expandafter#3%
           \romannumeral`&&@\csname XINT_#6_op_##2\expandafter\endcsname
        \fi
    }%
    \def#5%
    {%
        \XINT_expandableerror
        {Extra ) removed. Hit <return>, fingers crossed.}%
        \expandafter#2\romannumeral`&&@\expandafter\XINT_expr_put_op_first
        \romannumeral`&&@\XINT_expr_getop_legacy
    }%
}%
\let\XINT_expr_done\space
\xintFor #1 in {flexpr,iiexpr} \do {%
    \expandafter\XINT_def_start_and_check
    \csname XINT_#1_start\expandafter\endcsname
    \csname XINT_#1_check\expandafter\endcsname
    \csname XINT_#1_checkp\expandafter\endcsname
    \csname XINT_#1_op_-xii\expandafter\endcsname
    \csname XINT_#1_extra_)\endcsname
    {#1}%
}%
```

Here also we take some shortcuts relative to general philosophy and have no explicit exec macro.

```
\def\XINT_defop_oparen #1#2#3#4#5#6#7%
```

```
{%
    \def #1##1% op_(
    {%
        \expandafter #4\romannumeral`&&@\XINT_expr_getnext
    }%
    \def #2##1% op_)
    {%
        \expanded{\xint_noxpd{\XINT_expr_put_op_first{##1}}\expandafter}%
        \romannumeral`&&@\XINT_expr_getop
    }%
    \def #3% oparen
    {%
        \expandafter #4\romannumeral`&&@\XINT_expr_getnext
    }%
    \def #4##1% check-
    {%
        \xint_UDsignfork
            ##1{\expandafter#5\romannumeral`&&@#6}%
              -{#5##1}%
        \krof
    }%
    \def #5##1##2% checkp
    {%
        \ifcase ##1\expandafter\XINT_expr_missing_)
        \or \csname XINT_#7_op_##2\expandafter\endcsname
        \else
          \expandafter #5%
          \romannumeral`&&@\csname XINT_#7_op_##2\expandafter\endcsname
        \fi
    }%
}%
\def\XINT_expr_missing_)
   {\XINT_expandableerror
    {End of expression found, but some ) was missing there.}%
    \xint_c_ \XINT_expr_done }%
\xintFor #1 in {flexpr,iiexpr} \do {%
    \expandafter\XINT_defop_oparen
    \csname XINT_#1_op_(\expandafter\endcsname
    \csname XINT_#1_op_)\expandafter\endcsname
    \csname XINT_#1_oparen\expandafter\endcsname
    \csname XINT_#1_check-_)\expandafter\endcsname
    \csname XINT_#1_checkp_)\expandafter\endcsname
    \csname XINT_#1_op_-xii\endcsname
    {#1}%
}%
\let\XINT_expr_precedence_)\xint_c_i
\catcode`) 12
```

32.16. The comma as binary operator

New with 1.09a. Refactored at 1.4.

```
\def\XINT_defop_comma #1#2#3#4#5#6%
```

```
{%
    \def #1##1% \XINT_expr_op_,
    {%
      \expanded{\xint_noxpd{#2{##1}}\expandafter}%
      \romannumeral`&&@\expandafter#3\romannumeral`&&@\XINT_expr_getnext
    }%
    \def #2##1##2##3##4{##2##3{##1##4}}% \XINT_expr_exec_,
    \def #3##1% \XINT_expr_check-_,
    {%
      \xint_UDsignfork
        ##1{\expandafter#4\romannumeral`&&@#5}%
          -{#4##1}%
      \krof
    }%
    \def #4##1##2% \XINT_expr_checkp_,
    {%
      \ifnum ##1>\xint_c_iii
        \expandafter#4%
           \romannumeral`&&@\csname XINT_#6_op_##2\expandafter\endcsname
      \else
        \expandafter##1\expandafter##2%
      \fi
    }%
}%
\xintFor #1 in {flexpr,iiexpr} \do {%
\expandafter\XINT_defop_comma
    \csname XINT_#1_op_,\expandafter\endcsname
    \csname XINT_#1_exec_,\expandafter\endcsname
    \csname XINT_#1_check-_,\expandafter\endcsname
    \csname XINT_#1_checkp_,\expandafter\endcsname
    \csname XINT_#1_op_-xii\endcsname {#1}%
}%
\expandafter\let\csname XINT_expr_precedence_,\endcsname\xint_c_iii
```

32.17. The minus as prefix operator of variable precedence level

Inherits the precedence level of the previous infix operator, if the latter has at least the precedence level of binary + and -, i.e. currently 12.

Refactored at 1.4.

At 1.4g I belatedly observe that I have been defining architecture for op_-xvi but such operator can never be created, because there are no infix operators of precedence level 16. Perhaps in the past this was really needed? But now such 16 is precedence level of tacit multiplication which is implemented simply by the \XINT_expr_prec_tacit token, there is no macro check-_*** which would need an op_-xvi.

For the record: at least one scenario exists which creates tacit multiplication in front of a unary -, it is 2\count0 which first generates tacit multiplication then applies \number to \count0, but the operator is still *, so this triggers only \XINT_expr_op_-xiv, not -xvi.

At 1.4g we need 17 and not 18 anymore as the precedence of unary minus following power operators ^ and **. The needed \xint_c_xvii creation was added to xintkernel.sty.

```
\def\XINT_defop_minus_i #1#2#3#4#5#6#7%
{%
    \def #1% \XINT_expr_op_-<level>
```

```
    {%
      \expandafter #2\romannumeral`&&@\expandafter#3%
      \romannumeral`&&@\XINT_expr_getnext
    }%
    \def #2##1##2##3% \XINT_expr_exec_-<level>
    {%
      \expandafter ##1\expandafter ##2\expandafter
       {%
        \romannumeral`&&@\XINT:NEhook:f:one:from:one
        {\romannumeral`&&@#7##3}%
       }%
    }%
    \def #3##1% \XINT_expr_check-_-<level>
    {%
      \xint_UDsignfork
        ##1{\expandafter #4\romannumeral`&&@#1}%
          -{#4##1}%
      \krof
    }%
    \def #4##1##2% \XINT_expr_checkp_-<level>
    {%
      \ifnum ##1>#5%
        \expandafter #4%
        \romannumeral`&&@\csname XINT_#6_op_##2\expandafter\endcsname
      \else
        \expandafter ##1\expandafter ##2%
      \fi
    }%
}%
\def\XINT_defop_minus #1#2#3%
{%
    \expandafter\XINT_defop_minus_i
    \csname XINT_#1_op_-#3\expandafter\endcsname
    \csname XINT_#1_exec_-#3\expandafter\endcsname
    \csname XINT_#1_check-_-#3\expandafter\endcsname
    \csname XINT_#1_checkp_-#3\expandafter\endcsname
    \csname xint_c_#3\endcsname {#1}#2%
}%
\xintApplyInline{\XINT_defop_minus {flexpr}\calxFpOpp}{{xii}{xiv}{xvii}}%
\xintApplyInline{\XINT_defop_minus {iiexpr}\calxiiOpp}{{xii}{xiv}{xvii}}%
```

32.18. The * as Python-like «unpacking» prefix operator

New with 1.4. Prior to 1.4 the internal data structure was the one of \csname encapsulated comma separated numbers. No hierarchical structure was (easily) possible. At 1.4, we can use TeX braces because there is no detokenization to catcode 12.

```
\def\XINT_defpost_unpack#1#2#3%
{%
    \def#1##1{\expandafter#2\romannumeral`&&@\XINT_expr_getnext}%
    \def#2##1##2%
    {%
      \ifnum ##1>\xint_c_xx
```

```
        \expandafter #2%
        \romannumeral`&&@\csname XINT_#3_op_##2\expandafter\endcsname
      \else
        \expandafter##1%
         \expandafter##2\romannumeral0\expandafter\XINT:NEhook:unpack
      \fi
    }%
}%
\def\XINT:NEhook:unpack{\xint_stop_atfirstofone}%
\xintFor* #1 in {{flexpr}{iiexpr}}:
    {\expandafter\XINT_defpost_unpack\csname XINT_#1_op_0\expandafter\endcsname
                            \csname XINT_#1_until_unpack\endcsname {#1}}%
```

32.19. Infix operators

32.19.1 &&, ||, //, /:, +, −, *, /, ^, **, 'and', 'or', 'xor', and 'mod' 747
32.19.2 .., ..[, and].. for a..b and a..[b]..c syntax 749
32.19.3 <, >, ==, <=, >=, != with Python-like chaining 751
32.19.4 Support macros for .., ..[and].. 753
\xintSeq:tl:x, \xintfloatSeq:tl:x 753

1.2d adds the *** for tying via tacit multiplication, for example x/2y. Actually I don't need the _itself mechanism for ***, only a precedence.

At 1.4b we must make sure that the ! in expansion of \XINT_expr_itself_!= is of catcode 12 and not of catcode 11. This is because implementation of chaining of comparison operators proceeds via inserting the itself macro directly into upcoming token stream, whereas formerly such itself macros would be expanded only in a \csname...\endcsname context.

```
\catcode`& 12 \catcode`! 12
\xintFor* #1 in {{==}{!=}{<=}{>=}{&&}{||}{//}{/:}{..}{..[}{].}{]..}}%
    \do {\expandafter\def\csname XINT_expr_itself_#1\endcsname {#1}}%
\catcode`& 7  \catcode`! 11
```

32.19.1. &&, ||, //, /:, +, −, *, /, ^, **, 'and', 'or', 'xor', and 'mod'

At 1.4g I finally decide to enact the switch to right associativity for the power operators ^ and **.

This goes via inserting into the checkp macros not anymore the precedence chardef token (which now only serves as left precedence, inserted in the token stream) but in its place an \xint_c_<roman> token holding the right precedence. Which is also transmitted to spanned unary minus operators.

Here only levels 12, 14, and 17 are created as right precedences.

#6 and #7 got permuted and the new #7 is directly a control sequence. Also #3 and #4 are now integers which need \romannumeral.

```
\def\XINT_expr_defbin_ai #1#2#3#4#5#6#7#8%
{%
  \def #1##1% \XINT_expr_op_<op>
  {%
    \expanded{\xint_noxpd{#2{##1}}\expandafter}%
    \romannumeral`&&@\expandafter#3\romannumeral`&&@\XINT_expr_getnext
  }%
  \def #2##1##2##3##4% \XINT_expr_exec_<op>
  {%
    \expandafter##2\expandafter##3\expandafter
```

```
      {\romannumeral`&&@\XINT:NEhook:f:one:from:two
       {\romannumeral`&&@#7##1##4}}%
  }%
  \def #3##1% \XINT_expr_check-_<op>
  {%
    \xint_UDsignfork
      ##1{\expandafter#4\romannumeral`&&@#5}%
        -{#4##1}%
    \krof
  }%
  \def #4##1##2% \XINT_expr_checkp_<op>
  {%
    \ifnum ##1>#6%
      \expandafter#4%
      \romannumeral`&&@\csname XINT_#8_op_##2\expandafter\endcsname
    \else
      \expandafter ##1\expandafter ##2%
    \fi
  }%
}%
\def\XINT_expr_defbin_a #1#2#3#4#5%
{%
  \expandafter\XINT_expr_defbin_ai
  \csname XINT_#1_op_#2\expandafter\endcsname
  \csname XINT_#1_exec_#2\expandafter\endcsname
  \csname XINT_#1_check-_#2\expandafter\endcsname
  \csname XINT_#1_checkp_#2\expandafter\endcsname
  \csname XINT_#1_op_-\romannumeral\ifnum#4>12
          #4\else12\fi\expandafter\endcsname
  \csname xint_c_\romannumeral#4\endcsname
  #5%
  {#1}%
  \expandafter % done 3 times but well
  \let\csname XINT_expr_precedence_#2\expandafter\endcsname
      \csname xint_c_\romannumeral#3\endcsname
}%
\XINT_expr_defbin_a {flexpr}{||} {6} {6} \calxFpOR
\XINT_expr_defbin_a {iiexpr}{||} {6} {6} \calxOR
\catcode`& 12
\XINT_expr_defbin_a {flexpr}{&&} {8} {8} \calxFpAND
\XINT_expr_defbin_a {iiexpr}{&&} {8} {8} \calxAND
\catcode`& 7
\XINT_expr_defbin_a {flexpr}{xor}{6} {6} \calxFpXOR
\XINT_expr_defbin_a {iiexpr}{xor}{6} {6} \calxXOR
\XINT_expr_defbin_a {flexpr}{//} {14}{14}\calxFpDivFloor
\XINT_expr_defbin_a {iiexpr}{//} {14}{14}\calxiiDivFloor
\XINT_expr_defbin_a {flexpr}{/:} {14}{14}\calxFpMod
\XINT_expr_defbin_a {iiexpr}{/:} {14}{14}\calxiiMod
\XINT_expr_defbin_a {flexpr} +   {12}{12}\calxFpAdd
\XINT_expr_defbin_a {iiexpr} +   {12}{12}\calxiiAdd
\XINT_expr_defbin_a {flexpr} -   {12}{12}\calxFpSub
\XINT_expr_defbin_a {iiexpr} -   {12}{12}\calxiiSub
```

```
\XINT_expr_defbin_a {flexpr} *   {14}{14}\calxFpMul
\XINT_expr_defbin_a {iiexpr} *   {14}{14}\calxiiMul
\let\XINT_expr_prec_tacit \xint_c_xvi
\XINT_expr_defbin_a {flexpr} /   {14}{14}\calxFpDiv
\XINT_expr_defbin_a {iiexpr} /   {14}{14}\calxiiDivRound
```

At 1.4g, right associativity is implemented via a lowered right precedence here.

```
\XINT_expr_defbin_a {flexpr} ^   {18}{17}\XINTinFloatSciPow
\XINT_expr_defbin_a {iiexpr} ^   {18}{17}\calxiiPow
```

1.4g This is a trick (which was in old version of bnumexpr, I wonder why I did not have it here) but it will make error messages in case of **<token> confusing. The ^ here is of catcode 11 but it does not matter.

```
\expandafter\def\csname XINT_expr_itself_**\endcsname{^}%
\catcode`& 12
```

For this which contributes to implementing 'and', 'or', etc... see \XINT_expr_binopwrd.

```
\xintFor #1 in {and,or,xor,mod} \do
{%
   \expandafter\def\csname XINT_expr_itself_#1\endcsname {#1}%
}%
\expandafter\let\csname XINT_expr_precedence_and\expandafter\endcsname
                \csname XINT_expr_precedence_&&\endcsname
\expandafter\let\csname XINT_expr_precedence_or\expandafter\endcsname
                \csname XINT_expr_precedence_||\endcsname
\expandafter\let\csname XINT_expr_precedence_mod\expandafter\endcsname
                \csname XINT_expr_precedence_/:\endcsname
\xintFor #1 in {flexpr, iiexpr} \do
{%
   \expandafter\let\csname XINT_#1_op_and\expandafter\endcsname
                   \csname XINT_#1_op_&&\endcsname
   \expandafter\let\csname XINT_#1_op_or\expandafter\endcsname
                   \csname XINT_#1_op_||\endcsname
   \expandafter\let\csname XINT_#1_op_mod\expandafter\endcsname
                   \csname XINT_#1_op_/:\endcsname
}%
\catcode`& 7
```

32.19.2. .., ..[, and].. for a..b and a..[b]..c syntax

```
\def\XINT_expr_defbin_bi #1#2#3#4#5#6#7%
{%
  \def #1##1% \XINT_expr_op_..[
  {%
    \expanded{\xint_noxpd{#2{##1}}\expandafter}%
    \romannumeral`&&@\expandafter#3\romannumeral`&&@\XINT_expr_getnext
  }%
  \def #2##1##2##3##4% \XINT_expr_exec_..[
  {%
    ##2##3{{##1##4}}%
  }%
  \def #3##1% \XINT_expr_check-_..[
  {%
    \xint_UDsignfork
      ##1{\expandafter#4\romannumeral`&&@#5}%
```

```
        -{#4##1}%
    \krof
  }%
  \def #4##1##2% \XINT_expr_checkp_..[
  {%
    \ifnum ##1>#6%
      \expandafter#4%
      \romannumeral`&&@\csname XINT_#7_op_##2\expandafter\endcsname
    \else
      \expandafter ##1\expandafter ##2%
    \fi
  }%
}%
\def\XINT_expr_defbin_b #1%
{%
  \expandafter\XINT_expr_defbin_bi
  \csname XINT_#1_op_..[\expandafter\endcsname
  \csname XINT_#1_exec_..[\expandafter\endcsname
  \csname XINT_#1_check-_..[\expandafter\endcsname
  \csname XINT_#1_checkp_..[\expandafter\endcsname
  \csname XINT_#1_op_-xii\expandafter\endcsname
  \csname XINT_expr_precedence_..[\endcsname
  {#1}%
}%
\XINT_expr_defbin_b {flexpr}%
\XINT_expr_defbin_b {iiexpr}%
\expandafter\let\csname XINT_expr_precedence_..[\endcsname\xint_c_vi
\def\XINT_expr_defbin_ci #1#2#3#4#5#6#7#8%
{%
  \def #1##1% \XINT_expr_op_<op>
  {%
    \expanded{\xint_noxpd{#2{##1}}\expandafter}%
    \romannumeral`&&@\expandafter#3\romannumeral`&&@\XINT_expr_getnext
  }%
  \def #2##1##2##3##4% \XINT_expr_exec_<op>
  {%
    \expandafter##2\expandafter##3\expanded
    {{\XINT:NEhook:x:one:from:two#8##1##4}}%
  }%
  \def #3##1% \XINT_expr_check-_<op>
  {%
    \xint_UDsignfork
      ##1{\expandafter#4\romannumeral`&&@#5}%
        -{#4##1}%
    \krof
  }%
  \def #4##1##2% \XINT_expr_checkp_<op>
  {%
    \ifnum ##1>#6%
      \expandafter#4%
      \romannumeral`&&@\csname XINT_#7_op_##2\expandafter\endcsname
    \else
```

```
      \expandafter ##1\expandafter ##2%
    \fi
  }%
}%
\def\XINT_expr_defbin_c #1#2#3%
{%
  \expandafter\XINT_expr_defbin_ci
  \csname XINT_#1_op_#2\expandafter\endcsname
  \csname XINT_#1_exec_#2\expandafter\endcsname
  \csname XINT_#1_check-_#2\expandafter\endcsname
  \csname XINT_#1_checkp_#2\expandafter\endcsname
  \csname XINT_#1_op_-xii\expandafter\endcsname
  \csname XINT_expr_precedence_#2\endcsname
  {#1}#3%
  \expandafter\let
  \csname XINT_expr_precedence_#2\expandafter\endcsname\xint_c_vi
}%
\XINT_expr_defbin_c {flexpr} {..}\xintfloatSeq:tl:x
\XINT_expr_defbin_c {iiexpr} {..}\xintSeq:tl:x
\XINT_expr_defbin_c {flexpr}{]..}\xintfloatSeqB:tl:x
\XINT_expr_defbin_c {iiexpr}{]..}\calxiiSeqB:tl:x
```

32.19.3. <, >, ==, <=, >=, != with Python-like chaining

Added at 1.4b (2020/02/25). This is preliminary implementation of chaining of comparison operators like Python and (I think) l3fp do. I am not too happy with how many times the (second) operand (already evaluated) is fetched.

Modified at 1.4p (2026/03/01). Sadly this was broken ever since creation if used inside \xintiieval or \xintfloateval due to a macro using by oversight the parser type in the name of a token defining the precedence of the inserted &&. But this token never got defined. Anyhow we don't need another one beyond the one suitable for \xinteval. A unit test existed, *of course*, but it tested only \xinteval! Thanks to Thomas Ossig for report.

```
\def\XINT_expr_defbin_dii #1#2%
{%
  \def #1##1##2##3##4% \XINT_<expr>_exec_<op>
  {%
    \expandafter##2\expandafter##3\expandafter
      {\romannumeral`&&@\XINT:NEhook:f:one:from:two
         {\romannumeral`&&@#2##1##4}}%
  }%
}%
\def\XINT_expr_defbin_di #1#2#3#4#5#6#7#8#9%
{%
  \def #1##1% \XINT_<expr>_op_<op>
  {%
    \expanded{\xint_noxpd{#2{##1}}\expandafter}%
    \romannumeral`&&@\expandafter#7%
    \romannumeral`&&@\expandafter#3\romannumeral`&&@\XINT_expr_getnext
  }%
  \def #3##1% \XINT_<expr>_check-_<op>
  {%
    \xint_UDsignfork
```

```
      ##1{\expandafter#4\romannumeral`&&@#5}%
        -{#4##1}%
    \krof
  }%
  \def #4##1##2% \XINT_<expr>_checkp_<op>
  {%
    \ifnum ##1>#6%
      \expandafter#4%
      \romannumeral`&&@\csname XINT_#9_op_##2\expandafter\endcsname
    \else
      \expandafter ##1\expandafter ##2%
    \fi
  }%
  \let #6\xint_c_x
  \def #7##1% \XINT_expr_checkc_<op>
  {%
    \ifnum ##1=\xint_c_x\expandafter#8\fi ##1%
  }%
  \edef #8##1##2##3% \XINT_expr_execc_<op>
  {%
    \csname XINT_expr_precedence_\string&\string&\endcsname
    \expandafter\noexpand\csname XINT_expr_itself_\string&\string&\endcsname
    {##3}%
    \XINTfstop.{##3}##2%
  }%
  \XINT_expr_defbin_dii #2% \XINT_expr_exec_<op>
}%
\def\XINT_expr_defbin_d #1#2%#3%
{%
  \expandafter\XINT_expr_defbin_di
  \csname XINT_#1_op_#2\expandafter\endcsname
  \csname XINT_#1_exec_#2\expandafter\endcsname
  \csname XINT_#1_check-_#2\expandafter\endcsname
  \csname XINT_#1_checkp_#2\expandafter\endcsname
  \csname XINT_#1_op_-xii\expandafter\endcsname
  \csname XINT_expr_precedence_#2\expandafter\endcsname
  \csname XINT_#1_checkc_#2\expandafter\endcsname
  \csname XINT_#1_execc_#2\endcsname
  {#1}%#3%
}%
```

Attention that third token here is left in stream by defbin_b, then also by defbin_c and is picked up as #2 of defbin_d. Had to work around TeX accepting only 9 arguments. Why did it not start counting at #0 like all decent mathematicians do?

```
\XINT_expr_defbin_d {flexpr}<\calxFpLt% calx
\XINT_expr_defbin_d {iiexpr}<\calxiiLt
\XINT_expr_defbin_d {flexpr}>\calxFpGt% calx
\XINT_expr_defbin_d {iiexpr}>\calxiiGt
\XINT_expr_defbin_d {flexpr}{==}\calxFpEq% calx
\XINT_expr_defbin_d {iiexpr}{==}\calxiiEq
\XINT_expr_defbin_d {flexpr}{<=}\calxFpLtorEq% calx
\XINT_expr_defbin_d {iiexpr}{<=}\calxiiLtorEq
\XINT_expr_defbin_d {flexpr}{>=}\calxFpGtorEq% calx
```

```
\XINT_expr_defbin_d {iiexpr}{>=}\calxiiGtorEq
\XINT_expr_defbin_d {flexpr}{!=}\calxFpNotEq% calx
\XINT_expr_defbin_d {iiexpr}{!=}\calxiiNotEq
```

32.19.4. Support macros for .., ..[and]..

\xintSeq:tl:x, \xintfloatSeq:tl:x 753

\xintSeq:tl:x, \xintfloatSeq:tl:x a..b: the integers from ceil(a) to floor(b), decreasing if floor(b) < ceil(a), both ends included, never an empty list (a = b non-integer: ceil(a), floor(a)). Values of any of the three parsers; the ends must be TeX integers. \xintSeq:tl:x (\xinteval, \xintiieval): M items (\CALX_seq_m); \xintfloatSeq:tl:x (\xintfloateval): F items (\CALX_seq_f). b = I-: the single item ceil(a). b = I+ or I?, or a an I value: an expandable error, empty list.

ceil and floor by calxfp's \CALX_fint (sign s, |trunc| T, first omitted digit d, non-zero flag z) and its kind macros \CALX_fintk_c, \CALX_fintk_f: \CALX_seq_k writes sTs<k>sdz, i.e. s(T + a), in \numexpr. The loop carries the item builder after the two ends.

```
\def\xintSeq:tl:x {\CALX_seq\CALX_seq_m}%
\def\xintfloatSeq:tl:x {\CALX_seq\CALX_seq_f}%
\def\CALX_seq_f #1.{\romannumeral0\CALX_ffromraw{#1[0]}}%
\def\CALX_seq #1#2#3%
   {\expandafter\CALX_seq_b\romannumeral`&&@#3\xint:#1{#2}}%
\def\CALX_seq_b #1#2\xint:#3#4%
{%
    \if I#1\xint_dothis{\CALX_seq_i#2\xint:#3{#4}}\fi
    \xint_orthat{\expandafter\CALX_seq_c\the\numexpr\expandafter\CALX_seq_k
      \romannumeral0\CALX_fint#1#2\xint:\CALX_fintk_f.#3{#4}}%
}%
\def\CALX_seq_k #1#2.#3#4#5{#1#2#1#5#1#3#4}%
\def\CALX_seq_i #1\xint:#2#3%
   {\if-#1\xint_dothis{\CALX_seq_c.#2{#3}}\fi\xint_orthat\CALX_seq_err}%
\def\CALX_seq_c #1.#2#3%
   {\expandafter\CALX_seq_d\romannumeral`&&@#3\xint:#1.#2}%
\def\CALX_seq_d #1#2\xint:#3.#4%
{%
    \if I#1\xint_dothis\CALX_seq_err\fi
    \xint_orthat{\expandafter\CALX_seq_e\the\numexpr\expandafter\CALX_seq_k
      \romannumeral0\CALX_fint#1#2\xint:\CALX_fintk_c.#3.#4}%
}%
\def\CALX_seq_err{\XINT_expandableerror{a..b: invalid range.}}%
\def\CALX_seq_e #1.#2.#3%
{%
    \if\relax\detokenize{#2}\relax\xint_dothis{{#3#1.}}\fi
    \ifnum#2=#1 \xint_dothis{{#3#1.}}\fi
    \ifnum#2<#1 \xint_dothis{\CALX_seq_n #1.#2.#3}\fi
    \xint_orthat{\CALX_seq_p #1.#2.#3}%
}%
\def\CALX_seq_p #1.#2.#3%
   {{#3#1.}\ifnum#1=#2 \CALX_seq_x\fi
    \expandafter\CALX_seq_p\the\numexpr#1+\xint_c_i.#2.#3}%
\def\CALX_seq_n #1.#2.#3%
   {{#3#1.}\ifnum#1=#2 \CALX_seq_x\fi
```

```
    \expandafter\CALX_seq_n\the\numexpr#1-\xint_c_i.#2.#3}%
\def\CALX_seq_x #1#2.#3.#4{#1}%
```

a..[d]..b of \xintfloateval and \xintiieval: from a by steps d while not beyond b; d = 0: a alone; nil if b is on the wrong side of a. Exact stepping and comparison: \xintfloatSeqB:tl:x on F values (\CALX_fexact: F, M and raw A[N] exactly, A/B[N] with B > 1 rounded to P digits; \CALX_faddx, \CALX_fcmp), \calxiiSeqB:tl:x on M values (\CALX_tom, \CALX_iiadd, \CALX_iicmp_x). An I value among a, d, b: expandable error, empty list. \xinteval has its own (xintexpr.sty).

\CALX_seqb <conv><add><cmp>{a}{d}{b}: the three ends converted by <conv> (output Y\xint:), the sign of d by calxcore's \CALX_ifsgn. The loop \CALX_seqb_l {x}{d}{b}<add><cmp><order>: one comparison of x with b per item, the selectors \xint_stop_at...ofthree of <cmp> (equal, greater, less) taken in the order of \CALX_seqb_sw (d > 0: x against b) or \CALX_seqb_id (d < 0: b against x); the first step is a loop step with x = a.

```
\def\xintfloatSeqB:tl:x #1#2%
   {\CALX_seqb\CALX_fexact\CALX_faddx\CALX_seqb_fc#1{#2}}%
\def\calxiiSeqB:tl:x #1#2{\CALX_seqb\CALX_tom\CALX_iiadd\CALX_seqb_ic#1{#2}}%
\def\CALX_seqb_fc #1#2%
   {\ifcase\CALX_fcmp{#1}{#2} \expandafter\xint_stop_atfirstofthree
    \or\expandafter\xint_stop_atsecondofthree
    \else\expandafter\xint_stop_atthirdofthree\fi}%
\def\CALX_seqb_ic #1#2{\CALX_iicmp_x#1\xint:#2\xint:}%
\def\CALX_seqb #1#2#3#4#5%
   {\expandafter\CALX_seqb_a\romannumeral0#1{#5}#1#2#3{#4}}%
\def\CALX_seqb_a #1\xint:#2#3#4#5%
   {\expandafter\CALX_seqb_b\romannumeral0#2{#5}{#1}#2#3#4}%
\def\CALX_seqb_b #1\xint:#2#3#4#5#6%
   {\expandafter\CALX_seqb_c\romannumeral0#3{#6}{#1}{#2}#4#5}%
\def\CALX_seqb_c #1\xint:#2#3%
   {\CALX_seqb_d #1\xint:#2\xint:#3\xint:{#1}{#2}{#3}}%
\def\CALX_seqb_d #1#2\xint:#3#4\xint:#5#6\xint:%
{%
    \if I#1\xint_dothis\CALX_seqb_e\fi
    \if I#3\xint_dothis\CALX_seqb_e\fi
    \if I#5\xint_dothis\CALX_seqb_e\fi
    \xint_orthat\CALX_seqb_f
}%
\def\CALX_seqb_e #1#2#3#4#5{\CALX_seq_err}%
\def\CALX_seqb_f #1#2#3#4#5%
   {\romannumeral0\CALX_ifsgn{#3}{\CALX_seqb_l{#2}{#3}{#1}#4#5\CALX_seqb_id}%
    {{#2}}{\CALX_seqb_l{#2}{#3}{#1}#4#5\CALX_seqb_sw}}%
\def\CALX_seqb_l #1#2#3#4#5#6%
   {\romannumeral0#6#5{#3}{#1}{{#1}}{}%
    {{#1}\expandafter\CALX_seqb_m\romannumeral0#4{#1}{#2}\xint:
     {#2}{#3}#4#5#6}}%
\def\CALX_seqb_m #1\xint:{\CALX_seqb_l{#1}}%
\def\CALX_seqb_id #1{#1}%
\def\CALX_seqb_sw #1#2#3{#1{#3}{#2}}%
```

32.20. Square brackets [] both as a container and a Python slicer

Refactored at 1.4

The architecture allows to implement separately a «left» and a «right» precedence and this is crucial.

32.20.1 [...] as «oneple» constructor 755
32.20.2 [...] brackets and : operator for NumPy-like slicing and item indexing syntax 756
32.20.3 Macro layer implementing indexing and slicing 758

32.20.1. [...] as «oneple» constructor

In the definition of \XINT_expr_op_obracket the parameter is trash {}. The [is intercepted by the getnextfork and handled via the \xint_c_ii^v highest precedence trick to get op_obracket executed.

```
\def\XINT_expr_itself_obracket{obracket}%
\catcode`] 11 \catcode`[ 11
\def\XINT_expr_defbin_ei #1#2#3#4#5#6%
{%
    \def #1##1%
    {%
      \expandafter#3\romannumeral`&&@\XINT_expr_getnext
    }%
    \def #2##1% op_]
    {%
      \expanded{\xint_noxpd{\XINT_expr_put_op_first{{##1}}}\expandafter}%
         \romannumeral`&&@\XINT_expr_getop
    }%
    \def #3##1%  until_cbracket_a
    {%
      \xint_UDsignfork
         ##1{\expandafter#4\romannumeral`&&@#5}%  #5 = op_-xii
           -{#4##1}%
      \krof
    }%
    \def #4##1##2%  until_cbracket_b
    {%
      \ifcase ##1\expandafter\XINT_expr_missing_]
      \or \expandafter\XINT_expr_missing_]
      \or \expandafter#2%
      \else
      \expandafter #4%
        \romannumeral`&&@\csname XINT_#6_op_##2\expandafter\endcsname
      \fi
    }%
}%
\def\XINT_expr_defbin_e #1%
{%
    \expandafter\XINT_expr_defbin_ei
    \csname XINT_#1_op_obracket\expandafter\endcsname
    \csname XINT_#1_op_]\expandafter\endcsname
    \csname XINT_#1_until_cbracket_a\expandafter\endcsname
    \csname XINT_#1_until_cbracket_b\expandafter\endcsname
    \csname XINT_#1_op_-xii\endcsname
    {#1}%
}%
\XINT_expr_defbin_e {flexpr}%
\XINT_expr_defbin_e {iiexpr}%
\def\XINT_expr_missing_]
```

```
1770    {\XINT_expandableerror{Ooops, looks like we are missing a ]. Aborting!}%
1771     \xint_c_ \XINT_expr_done}%
1772 \let\XINT_expr_precedence_]\xint_c_ii
```

32.20.2. [...] brackets and : operator for NumPy-like slicing and item indexing syntax

The opening bracket [for the nutple constructor is filtered out by \XINT_expr_getnextfork and becomes «obracket» which behaves with precedence level 2. For the [..] Python slicer on the other hand, a real operator [is defined with precedence level 4 (it must be higher than precedence level of commas) on its right and maximal precedence on its left.

Important: although slicing and indexing shares many rules with Python/NumPy there are some significant differences: in particular there can not be any out-of-range error generated, slicing applies also to «oples» and not only to «nutple», and nested lists do not have to have their leaves at a constant depth. See the user manual.

Currently, NumPy-like nested (basic) slicing is implemented, i.e [a:b, c:d, N, e:f, M] type syntax with Python rules regarding negative integers. This is parsed as an expression and can arise from expansion or contain calculations.

Currently stepping, Ellipsis, and simultaneous multi-index extracting are not yet implemented.

There are some subtle things here with possibility of variables been passed by reference.

```
1773 \def\XINT_expr_defbin_fi #1#2#3#4#5#6%
1774 {%
1775     \def #1##1% \XINT_expr_op_[
1776     {%
1777         \expanded{\xint_noxpd{#2{##1}}\expandafter}%
1778         \romannumeral`&&@\expandafter#3\romannumeral`&&@\XINT_expr_getnext
1779     }%
1780     \def #2##1##2##3##4% \XINT_expr_exec_]
1781     {%
1782          \expandafter\XINT_expr_put_op_first
1783          \expanded
1784           {%
1785            {\XINT:NEhook:x:listsel\XINT_ListSel_top ##1__##4&%
1786             ({##1}\expandafter}%
1787            \expandafter
1788           }%
1789          \romannumeral`&&@\XINT_expr_getop
1790     }%
1791     \def #3##1% \XINT_expr_check-_]
1792     {%
1793       \xint_UDsignfork
1794         ##1{\expandafter#4\romannumeral`&&@#5}%
1795           -{#4##1}%
1796       \krof
1797     }%
1798     \def #4##1##2% \XINT_expr_checkp_]
1799     {%
1800       \ifcase ##1\XINT_expr_missing_]
1801         \or \XINT_expr_missing_]
1802         \or \expandafter##1\expandafter##2%
1803         \else \expandafter#4%
1804               \romannumeral`&&@\csname XINT_#6_op_##2\expandafter\endcsname
1805       \fi
1806     }%
```

```
}%
\let\XINT_expr_precedence_[ \xint_c_xx
\def\XINT_expr_defbin_f #1%
{%
    \expandafter\XINT_expr_defbin_fi
    \csname XINT_#1_op_[\expandafter\endcsname
    \csname XINT_#1_exec_]\expandafter\endcsname
    \csname XINT_#1_check-_]\expandafter\endcsname
    \csname XINT_#1_checkp_]\expandafter\endcsname
    \csname XINT_#1_op_-xii\endcsname
    {#1}%
}%
\XINT_expr_defbin_f {flexpr}%
\XINT_expr_defbin_f {iiexpr}%
\catcode`] 12 \catcode`[ 12
```

At 1.4 the getnext, scanint, scanfunc, getop chain got revisited to trigger automatic insertion of the nil variable if needed, without having in situations like here to define operators to support «[:» or «:]». And as we want to implement nested slicing à la NumPy, we would have had to handle also «:,» for example. Thus here we simply have to define the sole operator «:» and it will be some sort of inert joiner preparing a slicing spec.

```
\def\XINT_expr_defbin_gi #1#2#3#4#5#6%
{%
    \def #1##1% \XINT_expr_op_:
    {%
        \expanded{\xint_noxpd{#2{##1}}\expandafter}%
        \romannumeral`&&@\expandafter#3\romannumeral`&&@\XINT_expr_getnext
    }%
    \def #2##1##2##3##4% \XINT_expr_exec_:
    {%
       ##2##3{:##1{0};##4:_}%
    }%
    \def #3##1% \XINT_expr_check-_:
    {\xint_UDsignfork
        ##1{\expandafter#4\romannumeral`&&@#5}%
          -{#4##1}%
     \krof
    }%
    \def #4##1##2% \XINT_expr_checkp_:
    {%
      \ifnum ##1>\XINT_expr_precedence_:
        \expandafter #4\romannumeral`&&@%
                       \csname XINT_#6_op_##2\expandafter\endcsname
      \else
        \expandafter##1\expandafter##2%
      \fi
    }%
}%
\let\XINT_expr_precedence_: \xint_c_vi
\def\XINT_expr_defbin_g #1%
{%
    \expandafter\XINT_expr_defbin_gi
    \csname XINT_#1_op_:\expandafter\endcsname
```

```
    \csname XINT_#1_exec_:\expandafter\endcsname
    \csname XINT_#1_check-_:\expandafter\endcsname
    \csname XINT_#1_checkp_:\expandafter\endcsname
    \csname XINT_#1_op_-xii\endcsname {#1}%
}%
\XINT_expr_defbin_g {flexpr}%
\XINT_expr_defbin_g {iiexpr}%
```

32.20.3. Macro layer implementing indexing and slicing

xintexpr applies slicing not only to «objects» (which can be passed as arguments to functions) but also to «oples».

Our «nlists» are not necessarily regular N-dimensional arrays à la NumPy. Leaves can be at arbitrary depths. If we were handling regular «ndarrays», we could proceed a bit differently.

For the related explanations, refer to the user manual.

Notice that currently the code uses f-expandable (and not using \expanded) macros \xintApply, \xintApplyUnbraced, \xintKeep, \xintTrim, \xintNthOne from xinttools.

But the whole expansion happens inside an \expanded context, so possibly some gain could be achieved with x-expandable variants (xintexpr < 1.4 had an \xintKeep:x:csv).

I coded \xintApply:x and \xintApplyUnbraced:x in xinttools, Brief testing indicated they were perhaps a bit better for 5x5x5x5 and 15x15x15x15 arrays of 8 digits numbers and for 30x30x15 with 16 digits numbers: say 1% gain... this seems to raise to between 4% and 5% for 400x400 array of 1 digit...

Currently sticking with old macros.

```
\def\XINT_ListSel_deeper #1%
{%
    \if :#1\xint_dothis\XINT_ListSel_slice_next\fi
    \xint_orthat {\XINT_ListSel_extract_next {#1}}%
}%
\def\XINT_ListSel_slice_next #1(%
{%
    \xintApply{\XINT_ListSel_recurse{:#1}}%
}%
\def\XINT_ListSel_extract_next #1(%
{%
    \xintApplyUnbraced{\XINT_ListSel_recurse{#1}}%
}%
\def\XINT_ListSel_recurse #1#2%
{%
    \XINT_ListSel_check #2__#1({#2}\expandafter\empty\empty
}%
\def\XINT_ListSel_check{\expandafter\XINT_ListSel_check_a \string}%
\def\XINT_ListSel_check_a #1%
{%
    \if #1\bgroup\xint_dothis\XINT_ListSel_check_is_ok\fi
    \xint_orthat\XINT_ListSel_check_leaf
}%
\def\XINT_ListSel_check_leaf #1\expandafter{\expandafter}%
\def\XINT_ListSel_check_is_ok
{%
    \expandafter\XINT_ListSel_check_is_ok_a\expandafter{\string}%
}%
\def\XINT_ListSel_check_is_ok_a #1__#2%
```

```
{%
    \if :#2\xint_dothis{\XINT_ListSel_slice}\fi
    \xint_orthat {\XINT_ListSel_nthone {#2}}%
}%
\def\XINT_ListSel_top #1#2%
{%
    \if _\noexpand#2%
           \expandafter\XINT_ListSel_top_one_or_none\string#1.\else
           \expandafter\XINT_ListSel_top_at_least_two\fi
}%
\def\XINT_ListSel_top_at_least_two #1__{\XINT_ListSel_top_ople}%
\def\XINT_ListSel_top_one_or_none #1%
{%
    \if #1_\xint_dothis\XINT_ListSel_top_nil\fi
    \if #1.\xint_dothis\XINT_ListSel_top_nutple_a\fi
    \if #1\bgroup\xint_dothis\XINT_ListSel_top_nutple\fi
    \xint_orthat\XINT_ListSel_top_number
}%
\def\XINT_ListSel_top_nil #1\expandafter#2\expandafter{\fi\expandafter}%
\def\XINT_ListSel_top_nutple
{%
    \expandafter\XINT_ListSel_top_nutple_a\expandafter{\string}%
}%
\def\XINT_ListSel_top_nutple_a #1_#2#3(#4%
{%
    \fi\if :#2\xint_dothis{{\XINT_ListSel_slice #3(#4}}\fi
    \xint_orthat {\XINT_ListSel_nthone {#2}#3(#4}%
}%
\def\XINT_ListSel_top_number #1_{\fi\XINT_ListSel_top_ople}%
\def\XINT_ListSel_top_ople #1%
{%
    \if :#1\xint_dothis\XINT_ListSel_slice\fi
    \xint_orthat {\XINT_ListSel_nthone {#1}}%
}%
\def\XINT_ListSel_slice #1%
{%
    \expandafter\XINT_ListSel_slice_a \expandafter{\romannumeral0%
    \expandafter\CALX_fnum\romannumeral`&&@\xint_firstofone#1\xint:}%
}%
\def\XINT_ListSel_slice_a #1#2;#3#4%
{%
   \if _#4\expandafter\XINT_ListSel_s_b
          \else\expandafter\XINT_ListSel_slice_b\fi
   #1;#3%
}%
\def\XINT_ListSel_s_b #1#2;#3#4%
{%
    \if &#4\expandafter\XINT_ListSel_s_last\fi
    \XINT_ListSel_s_c #1{#1#2}{#4}%
}%
\def\XINT_ListSel_s_last\XINT_ListSel_s_c #1#2#3(#4%
{%
```

```
    \if-#1\expandafter\xintKeep\else\expandafter\xintTrim\fi {#2}{#4}%
}%
\def\XINT_ListSel_s_c #1#2#3(#4%
{%
    \expandafter\XINT_ListSel_deeper
    \expanded{\xint_noxpd{#3}(\expandafter}\expandafter{%
    \romannumeral0%
    \if-#1\expandafter\xintkeep\else\expandafter\xinttrim\fi {#2}{#4}}%
}%
```

\xintNthElt from xinttools (knowingly) strips one level of braces when fetching kth «item» from {v1}...{vN}. If we expand {\xintNthElt{k}{{v1}...{vN}}} (notice external braces):

if k is out of range we end up with {}
if k is in range and the kth braced item was {} we end up with {}
if k is in range and the kth braced item was {17} we end up with {17}

Problem is that individual numbers such as 17 are stored {{17}}. So we must have one more brace pair and in the first two cases we end up with {{}}. But in the first case we should end up with the empty ople {}, not the empty bracketed ople {{}}.

I have thus added \xintNthOne to xinttools which does not strip brace pair from an extracted item.

Attention: \XINT_nthonepy_a does no expansion on second argument. But here arguments are either numerical or already expanded. Normally.

```
\def\XINT_ListSel_nthone #1#2%
{%
    \if &#2\expandafter\XINT_ListSel_nthone_last\fi
    \XINT_ListSel_nthone_a {#1}{#2}%
}%
\def\XINT_ListSel_nthone_a #1#2(#3%
{%
    \expandafter\XINT_ListSel_deeper
    \expanded{\xint_noxpd{#2}(\expandafter}\expandafter{%
    \romannumeral0\expandafter\XINT_nthonepy_a\the\numexpr
    \calxListItemToNum{#1}.{#3}}%
}%
\def\XINT_ListSel_nthone_last\XINT_ListSel_nthone_a #1#2(%#3%
{%
    \romannumeral0\expandafter\XINT_nthonepy_a\the\numexpr
    \calxListItemToNum{#1}.%{#3}
}%
```

The macros here are basically f-expandable and use the f-expandable \xintKeep and \xintTrim. Prior to xint 1.4, there was here an x-expandable \xintKeep:x:csv dealing with comma separated items, for time being we make do with our f-expandable toolkit.

```
\def\XINT_ListSel_slice_b #1;#2_#3%
{%
    \if &#3\expandafter\XINT_ListSel_slice_last\fi
    \expandafter\XINT_ListSel_slice_c \expandafter{\romannumeral0%
    \expandafter\CALX_fnum\romannumeral`&&@\xint_firstofone#2\xint:};#1;{#3}%
}%
\def\XINT_ListSel_slice_last\expandafter\XINT_ListSel_slice_c #1;#2;#3(%#4
{%
    \expandafter\XINT_ListSel_slice_last_c #1;#2;%{#4}
}%
```

Modified at 1.4n (2025/09/05). Compatibility with LuaMetaTEX regarding ; not usable as \numexpr delimiter. Fortunately I had mostly used \xint: or a dot in the past throughout the code base, so not many locations needed adjustments. Here I simpy replaced all semi-colons by fullstops starting here with how \XINT_ListSel_slice_d receives arguments. Fortuntely \xint_gob_til_sc was not used around here.

```
\def\XINT_ListSel_slice_last_c #1;#2;#3%
{%
    \romannumeral0\XINT_ListSel_slice_d #2.#1.{#3}%
}%
\def\XINT_ListSel_slice_c #1;#2;#3(#4%
{%
   \expandafter\XINT_ListSel_deeper
   \expanded{\xint_noxpd{#3}(\expandafter}\expandafter{%
   \romannumeral0\XINT_ListSel_slice_d #2.#1.{#4}}%
}%
\def\XINT_ListSel_slice_d #1#2.#3#4.%
{%
    \xint_UDsignsfork
       #1#3\XINT_ListSel_N:N
        #1-\XINT_ListSel_N:P
        -#3\XINT_ListSel_P:N
         --\XINT_ListSel_P:P
    \krof #1#2.#3#4.%
}%
\def\XINT_ListSel_P:P #1.#2.#3%
{%
    \unless\ifnum #1<#2 \expandafter\xint_gob_andstop_iii\fi
    \xintkeep{#2-#1}{\xintTrim{#1}{#3}}%
}%
\def\XINT_ListSel_N:N #1.#2.#3%
{%
    \expandafter\XINT_ListSel_N:N_a
    \the\numexpr #2-#1\expandafter.\the\numexpr#1+\xintLength{#3}.{#3}%
}%
\def\XINT_ListSel_N:N_a #1.#2.#3%
{%
    \unless\ifnum #1>\xint_c_ \expandafter\xint_gob_andstop_iii\fi
    \xintkeep{#1}{\xintTrim{\ifnum#2<\xint_c_\xint_c_\else#2\fi}{#3}}%
}%
\def\XINT_ListSel_N:P #1.#2.#3%
{%
    \expandafter\XINT_ListSel_N:P_a
    \the\numexpr #1+\xintLength{#3}.#2.{#3}%
}%
\def\XINT_ListSel_N:P_a #1#2.%
   {\if -#1\expandafter\XINT_ListSel_O:P\fi\XINT_ListSel_P:P #1#2.}%
\def\XINT_ListSel_O:P\XINT_ListSel_P:P #1.{\XINT_ListSel_P:P 0.}%
\def\XINT_ListSel_P:N #1.#2.#3%
{%
    \expandafter\XINT_ListSel_P:N_a
    \the\numexpr #2+\xintLength{#3}.#1.{#3}%
}%
```

```
\def\XINT_ListSel_P:N_a #1#2.#3.%
   {\if -#1\expandafter\XINT_ListSel_P:O\fi\XINT_ListSel_P:P #3.#1#2.}%
\def\XINT_ListSel_P:O\XINT_ListSel_P:P #1.#2.{\XINT_ListSel_P:P #1.0.}%
```

32.21. Support for raw A/B[N]

Releases earlier than 1.1 required the use of braces around A/B[N] input. The [N] is now implemented directly. *BUT* this uses a delimited macro! thus N is not allowed to be itself an expression (I could add it...). \xintE, \xintiiE, and \XINTinFloatE all put #2 in a \numexpr. But attention to the fact that \numexpr stops at spaces separating digits: \the\numexpr 3 + 7 9\relax gives 109\relax !! Hence we have to be careful.

\numexpr will not handle catcode 11 digits, but adding a \detokenize will suddenly make illicit for N to rely on macro expansion.

At 1.4, [is already overloaded and it is not easy to support this. We do this by a kludge maintaining more or less former (very not efficient) way but using $ sign which is free for time being. No, finally I use the null character, should be safe enough! (I hesitated about using R with catcode 12).

As for ? operator we needed to hack into \XINT_expr_getop_b for intercepting that pseudo operator. See also \XINT_expr_scanint_c (\XINT_expr_rawxintfrac).

calx: the precedence 14 of (17): 2^3[1] is (2^3)[1]. The finisher operators \xintiieval: raw [N] goes through \CALX_iilit (integer check, M or raw), not \xintiiE.

```
\catcode0 11
\let\XINT_expr_precedence_&&@ \xint_c_xiv
\def\CALX_iiE #1{\CALX_iiE_a #1[0]\xint:}%
\def\CALX_iiE_a #1[#2]#3\xint:#4%
   {\expandafter\CALX_iiE_b\the\numexpr#2+#4.{#1}}%
\def\CALX_iiE_b #1.#2{\CALX_iilit{#2}{#1}}%
\def\CALX_rawE #1{\CALX_rawE_a #1[0]\xint:}%
\def\CALX_rawE_a #1[#2]#3\xint:#4{\xint_noxpd{#1}[\the\numexpr#2+#4\relax]}%
\def\XINT_iiexpr_op_&&@ #1#2]%
{%
    \expandafter\XINT_expr_put_op_first
    \expanded{{{\romannumeral0\CALX_iiE#1{\xint_zapspaces #2 \xint_gobble_i}}}%
    \expandafter}\romannumeral`&&@\XINT_expr_getop
}%
\def\XINT_flexpr_op_&&@ #1#2]%
{%
    \expandafter\XINT_expr_put_op_first
    \expanded{{{\calxFpE#1{\xint_zapspaces #2 \xint_gobble_i}}}%
    \expandafter}\romannumeral`&&@\XINT_expr_getop
}%
```

calx: fake operator [0] appended in \xinteval and \xintfloateval; M in \xintiieval (\CALX_mfrom raw; raw in compiled functions, see \XINT_expr_redefinemacros).

```
\catcode2 11
\def\XINT_flexpr_op_&&B #1%
{%
    \expandafter\XINT_expr_put_op_first
    \expanded{{{\xint_noxpd#1[0]}}\expandafter}%
    \romannumeral`&&@\XINT_expr_getop
}%
\def\XINT_iiexpr_op_&&B #1%
{%
```

```
    \expandafter\XINT_expr_put_op_first
    \expanded{{{\romannumeral0\CALX_mfromraw#1}}\expandafter}%
    \romannumeral`&&@\XINT_expr_getop
}%
\catcode2 12
```

calx: fake operator scanner): #2 is the exponent; [#2] appended (as 1.4p); \xintiieval: \CALX_iilit (M if the value is an integer, else raw A[N]). Raw [N] (\xintiieval operators keep the literal raw (\XINT_expr_redefinemacros).

```
\catcode3 11
\def\XINT_flexpr_op_&&C #1#2]%
{%
    \expandafter\XINT_expr_put_op_first
    \expanded{{{\xint_noxpd#1[#2]}}\expandafter}%
    \romannumeral`&&@\XINT_expr_getop
}%
\def\XINT_iiexpr_op_&&C #1#2]%
{%
    \expandafter\XINT_expr_put_op_first
    \expanded{{{\romannumeral0\CALX_iilit#1{#2}}}\expandafter}%
    \romannumeral`&&@\XINT_expr_getop
}%
\catcode3 12
\catcode0 12
```

32.22. ? as two-way and ?? as three-way «short-circuit» conditionals

Comments undergoing reconstruction.

```
\let\XINT_expr_precedence_? \xint_c_xx
\catcode`- 11
\def\XINT_flexpr_op_?{\XINT_expr_op__? \XINT_flexpr_op_-xii}%
\def\XINT_iiexpr_op_?{\XINT_expr_op__? \XINT_iiexpr_op_-xii}%
\catcode`- 12
\def\XINT_expr_op__? #1#2#3%
    {\XINT_expr_op__?_a #3!\xint_bye\XINT_expr_exec_? {#1}{#2}{#3}}%
\def\XINT_expr_op__?_a #1{\expandafter\XINT_expr_op__?_b\detokenize{#1}}%
\def\XINT_expr_op__?_b #1%
    {\if ?#1\expandafter\XINT_expr_op__?_c\else\expandafter\xint_bye\fi }%
\def\XINT_expr_op__?_c #1{\xint_gob_til_! #1\XINT_expr_op_?? !\xint_bye}%
\def\XINT_expr_op_?? !\xint_bye\xint_bye\XINT_expr_exec_?{\XINT_expr_exec_??}%
\catcode`- 11
\def\XINT_expr_exec_? #1#2%
{%
    \expandafter\XINT_expr_check-_after?\expandafter#1%
    \romannumeral`&&@\expandafter\XINT_expr_getnext\romannumeral0\CALX_ifnz#2%
}%
\def\XINT_expr_exec_?? #1#2#3%
{%
    \expandafter\XINT_expr_check-_after?\expandafter#1%
    \romannumeral`&&@\expandafter\XINT_expr_getnext\romannumeral0\CALX_ifsgn#2%
}%
\def\XINT_expr_check-_after? #1{%
\def\XINT_expr_check-_after? ##1##2%
```

```
{%
    \xint_UDsignfork
        ##2{##1}%
        #1{##2}%
    \krof
}}\expandafter\XINT_expr_check-_after?\string -%
\catcode`- 12
```

32.23. ! as postfix factorial operator

```
\let\XINT_expr_precedence_! \xint_c_xx
\def\XINT_flexpr_op_! #1%
{%
    \expandafter\XINT_expr_put_op_first
    \expanded{{\romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\calxFpFacdigits#1}}\expandafter}%
    \romannumeral`&&@\XINT_expr_getop
}%
\def\XINT_iiexpr_op_! #1%
{%
    \expandafter\XINT_expr_put_op_first
    \expanded{{\romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\calxiiFac#1}}\expandafter}%
    \romannumeral`&&@\XINT_expr_getop
}%
```

At 1.4g, fix for input "x! == y" via a fake operator !==. The ! is of catcode 11 but this does not matter here. The definition of \XINT_expr_itself_!== is required by the functioning of the scanop macros.

We don't have to worry about "x! = y" as the single-character Boolean comparison = operator has been removed from syntax. Fixing it would have required obeying space tokens when parsing operators. For "x! == y" case, obeying space tokens would not solve "x!==y" input case anyhow.

```
\expandafter
\def\csname XINT_expr_precedence_!==\expandafter\endcsname
    \csname XINT_expr_itself_!==\endcsname {\XINT_expr_precedence_! !==}%
\expandafter\def\csname XINT_expr_itself_!==\endcsname{!==}%
```

32.24. User defined variables

32.24.1 \xintdefiivar, \xintdeffloatvar 764
32.24.2 \xintunassignvar 768

32.24.1. \xintdefiivar, \xintdeffloatvar

Modified at 1.1 (2014/10/28).

Modified at 1.2p (2017/12/05). Extends \xintdefvar et al. to accept simultaneous assignments to multiple variables.

Modified at 1.3c (2018/06/17). Use \xintexprSafeCatcodes (to palliate issue with active semi-colon from Babel+French if in body of a LaTeX document).

And allow usage with both syntaxes name:=expr; or name=expr;. Also the colon may have catcode 11, 12, or 13 with no issue. Variable names may contain letters, digits, underscores, and must not start with a digit. Names starting with @ or an underscore are reserved.

- currently @, @1, @2, @3, and @4 are reserved because they have special meanings for use in iterations,
- @@, @@@, @@@@ are also reserved but are technically functions, not variables: a user may possibly define @@ as a variable name, but if it is followed by parentheses, the function interpretation will be applied (rather than the variable interpretation followed by a tacit multiplication),
- since 1.2l, the underscore _ may be used as separator of digits in long numbers. Hence a variable whose name starts with _ will not play well with the mechanism of tacit multiplication of variables by numbers: the underscore will be removed from input stream by the number scanner, thus creating an undefined or wrong variable name, or none at all if the variable name was an initial _ followed by digits.

Note that the optional argument [P] as usable with \xintfloatexpr is **not** supported by \xintdefloatvar. One must do \xintdefloatvar foo = \xintfloatexpr[16] blabla \relax; to achieve the effect.

Modified at 1.4 (2020/01/31). The expression will be fetched up to final semi-colon in a manner allowing inner semi-colons as used in the iter(), rseq(), subsm(), subsn() etc... syntax. They don't need to be hidden within a braced pair anymore.

Modified at 1.4 (2020/01/31). Automatic unpacking in case of simultaneous assignments if the expression evaluates to a nutple.
Notes (added much later on 2021/06/10 during preparation of 1.4i):

1. the code did not try to intercept illicit syntax such as \xintdefvar a,b,c:=<number>;. It blindly «unpacked» the number handling it as if it was a nutple. The extended functionality added at 1.4i requires to check for such a situation, as the syntax is not illicit anymore.

2. the code was broken in case the expression to evaluate was an ople of length 10 or more, due to a silly mistake at some point during 1.4 development which replaced some \ifnum by an \if, perhaps due to mental confusion with the fact that functions can have at most 9 arguments, but here the code is about defining variables. Anyway this got fixed as corollary to the 1.4i extension.

Modified at 1.4c (2021/02/20). One year later I realized I had broken tacit multiplication for situations such as variable(1+2). As hinted at in comments above before 1.4 release I had been doing some deep refactoring here, which I cancelled almost completely in the end... but not quite, and as a result there was a problem that some macro holding braced contents was expanded to late, once it was in old core routines of xintfrac not expecting other things than digits. I do an emergency bugfix here with some \expandafter's but I don't have the code in my brain at this time, and don't have the luxury now to invest into it. Let's hope this does not induce breakage elsewhere, and that the February 2020 1.4 did not break something else.

Modified at 1.4e (2021/05/05). Modifies \xintdefloatvar to round to the prevailing precision (formerly, any operation would induce rounding, but in case of things such as \xintdefloatvar foo:=\xintexpr 1/100!\relax; there was no automatic rounding. One could use 0+ syntax to trigger it, and for oples, some trick like \xintfloatexpr[\XINTdigits]...\relax extra wrapper.

Modified at 1.4g (2021/05/25). The \expandafter\expandafter\expandafter et al. chain which was kept by \XINT_expr_defvar_one_b for expanding only at time of use the \XINT_expr_var_foo in \XINT_expr_onliteral_foo were senseless overhead added at 1.4c. This is used only for real variables, not dummy variables or fake variables and it is simpler to have the \XINT_expr_var_foo pre-expanded. So let's use some \edef here.
The \XINT_expr_onliteral_foo is expanded as result of action of \XINT_expr_op_` (or \XINT_flexpr_op_`, \XINT_iiexpr_op_`) which itself was triggered consuming already an \XINT_expr_put_op_first, so its expansion has to produce tokens as expected after \XINT_expr_put_op_first: <precedence token><op token>{expanded value}.

Modified at 1.4i (2021/06/11). Implement extended notion of simultaneous assignments: if there are more variables than values, define the extra variables to be nil. If there are less variables than values let the last variable be defined as the ople concatenating all non reclaimed values.

If there are at least two variables, the right hand side, if it turns out to be a nutple, is (as since 1.4) automatically unpacked, then the above rules apply.

Modified at 1.4i (2021/06/11). Fix the long-standing «seq renaming bug» via a change here of the name of auxiliary macro. Previously «onliteral_<varname>» now «var*_<varname>». I hesitated with using «var_varname*» rather.

Hesitated adding \XINT_expr_letvar_one (motivation: case of simultaneous assignments leading to defining «nil» variables). Finally, no.

```
\catcode`* 11
\def\XINT_expr_defvar_one #1#2%
{%
    \XINT_global
    \expandafter\edef\csname XINT_expr_varvalue_#1\endcsname {#2}%
    \XINT_expr_defvar_one_b {#1}%
}%
\def\XINT_expr_defvar_one_b #1%
{%
    \XINT_global
    \expandafter\edef\csname XINT_expr_var_#1\endcsname
        {{\expandafter\noexpand\csname XINT_expr_varvalue_#1\endcsname}}%
    \XINT_global
    \expandafter\edef\csname XINT_expr_var*_#1\endcsname
        {\XINT_expr_prec_tacit *\csname XINT_expr_var_#1\endcsname(}%
    \ifxintverbose\xintMessage{xintexpr}{Info}%
       {Variable #1 \ifxintglobaldefs globally \fi
        defined with value \csname XINT_expr_varvalue_#1\endcsname.}%
    \fi
}%
\catcode`* 12
\catcode`~ 13
\catcode`: 12
\def\XINT_expr_defvar_getname #1:#2~%
{%
    \endgroup
    \def\XINT_defvar_tmpa{#1}\edef\XINT_defvar_tmpc{\xintCSVLength{#1}}%
}%
\def\XINT_expr_defvar #1#2%
{%
    \def\XINT_defvar_tmpa{#2}%
    \expandafter\XINT_expr_defvar_a\expanded{\xint_noxpd{{#1}}\expandafter}%
    \romannumeral\XINT_expr_fetch_to_semicolon
}%
\def\XINT_expr_defvar_a #1#2%
{%
    \xintexprRestoreCatcodes
```

Maybe SafeCatcodes was without effect because the colon and the rest are from some earlier macro definition. Give a safe definition to active colon (even if in math mode with a math active colon..).

The \XINT_expr_defvar_getname closes the group opened here.

```
    \begingroup\lccode`~`: \lowercase{\let~}\empty
    \edef\XINT_defvar_tmpa{\XINT_defvar_tmpa}%
    \edef\XINT_defvar_tmpa{\xint_zapspaces_o\XINT_defvar_tmpa}%
    \expandafter\XINT_expr_defvar_getname
                \detokenize\expandafter{\XINT_defvar_tmpa}:~%
    \ifcase\XINT_defvar_tmpc\space
      \xintMessage {xintexpr}{Error}
      {Aborting: not allowed to declare variable with empty name.}%
    \or
     \XINT_global
     \expandafter
     \edef\csname XINT_expr_varvalue_\XINT_defvar_tmpa\endcsname{#1#2\relax}%
     \XINT_expr_defvar_one_b\XINT_defvar_tmpa
    \else
     \edef\XINT_defvar_tmpb{#1#2\relax}%
     \edef\XINT_defvar_tmpd{\expandafter\xintLength\expandafter
                            {\XINT_defvar_tmpb}}%
     \ifnum\XINT_defvar_tmpd=\xint_c_i
         \oodef\XINT_defvar_tmpb{\expandafter\xint_firstofone
                       \XINT_defvar_tmpb}%
         \if0\expandafter\expandafter\expandafter\XINT_defvar_checkifnutple
                     \expandafter\string\XINT_defvar_tmpb _\xint_bye
           \odef\XINT_defvar_tmpb{\expandafter{\XINT_defvar_tmpb}}%
         \else
           \edef\XINT_defvar_tmpd{\expandafter\xintLength
                                  \expandafter{\XINT_defvar_tmpb}}%
         \fi
     \fi
     \xintAssignArray\xintCSVtoList\XINT_defvar_tmpa\to\XINT_defvar_tmpvar
     \def\XINT_defvar_tmpe{1}%
     \expandafter\XINT_expr_defvar_multiple\XINT_defvar_tmpb\relax
   \fi
}%
\def\XINT_defvar_checkifnutple#1%
{%
    \if#1_1\fi
    \if#1\bgroup1\fi
    0\xint_bye
}%
\def\XINT_expr_defvar_multiple
{%
    \ifnum\XINT_defvar_tmpe<\XINT_defvar_tmpc\space
        \expandafter\XINT_expr_defvar_multiple_one
    \else
        \expandafter\XINT_expr_defvar_multiple_last\expandafter\empty
    \fi
}%
\def\XINT_expr_defvar_multiple_one
{%
    \ifnum\XINT_defvar_tmpe>\XINT_defvar_tmpd\space
       \expandafter\XINT_expr_defvar_one
       \csname XINT_defvar_tmpvar\XINT_defvar_tmpe\endcsname{}%
```

```
        \edef\XINT_defvar_tmpe{\the\numexpr\XINT_defvar_tmpe+1}%
        \expandafter\XINT_expr_defvar_multiple
    \else
        \expandafter\XINT_expr_defvar_multiple_one_a
    \fi
}%
\def\XINT_expr_defvar_multiple_one_a #1%
{%
    \expandafter\XINT_expr_defvar_one
        \csname XINT_defvar_tmpvar\XINT_defvar_tmpe\endcsname{{#1}}%
    \edef\XINT_defvar_tmpe{\the\numexpr\XINT_defvar_tmpe+1}%
    \XINT_expr_defvar_multiple
}%
\def\XINT_expr_defvar_multiple_last #1\relax
{%
    \expandafter\XINT_expr_defvar_one
        \csname XINT_defvar_tmpvar\XINT_defvar_tmpe\endcsname{#1}%
    \xintRelaxArray\XINT_defvar_tmpvar
    \let\XINT_defvar_tmpa\empty
    \let\XINT_defvar_tmpb\empty
    \let\XINT_defvar_tmpc\empty
    \let\XINT_defvar_tmpd\empty
    \let\XINT_defvar_tmpe\empty
}%
\catcode`~ 3
\catcode`: 11
```

This SafeCatcodes is mainly in the hope that semi-colon ending the expression can still be sanitized.

Pre 1.4e definition:

```
\def\xintdefloatvar      {\xintexprSafeCatcodes\xintdefloatvar_a}
\def\xintdefloatvar_a #1={\XINT_expr_defvar\xintthebarefloateval{#1}}
```

This would keep the value (or values) with extra digits, now. If this is actually wanted one can use \xintdefvar foo:=\xintfloatexpr...\relax; syntax, but recalling that only operations trigger the rounding inside \xintfloatexpr. Some tricks are needed for no operations case if multiple or nested values. But for a single one, one can use simply the float() function.

```
\def\xintdefiivar     {\xintexprSafeCatcodes\xintdefiivar_a}%
\def\xintdefiivar_a#1={\XINT_expr_defvar\xintthebareiieval{#1}}%
\def\xintdefloatvar   {\xintexprSafeCatcodes\xintdefloatvar_a}%
\def\xintdefloatvar_a #1={\XINT_expr_defvar\xintthebareroundedfloateval{#1}}%
```

32.24.2. \xintunassignvar

Modified at 1.2e (2015/11/22).

Modified at 1.3d (2019/01/06). Embarrassingly I had for a long time a misunderstanding of \ifcsname (let's blame its documentation) and I was not aware that it chooses FALSE branch if tested control sequence has been \let to \undefined... So earlier version didn't do the right thing (and had another bug: failure to protect \.=0 from expansion).

The \ifcsname tests are done in \XINT_expr_op__ and \XINT_expr_op_`.

Modified at 1.4i (2021/06/11). Track s/onliteral/var*/ change in macro names.

```
\def\xintunassignvar #1{%
   \edef\XINT_unvar_tmpa{#1}%
```

```
    \edef\XINT_unvar_tmpa {\xint_zapspaces_o\XINT_unvar_tmpa}%
    \ifcsname XINT_expr_var_\XINT_unvar_tmpa\endcsname
        \ifnum\expandafter\xintLength\expandafter{\XINT_unvar_tmpa}=\@ne
          \expandafter\xintnewdummy\XINT_unvar_tmpa
        \else
          \XINT_global\expandafter
            \let\csname XINT_expr_varvalue_\XINT_unvar_tmpa\endcsname
                \XINT_undefined
          \XINT_global\expandafter
            \let\csname XINT_expr_var_\XINT_unvar_tmpa\endcsname
                \XINT_undefined
          \XINT_global\expandafter
            \let\csname XINT_expr_var*_\XINT_unvar_tmpa\endcsname
                \XINT_undefined
          \ifxintverbose\xintMessage {xintexpr}{Info}%
            {Variable \XINT_unvar_tmpa\space has been
             \ifxintglobaldefs globally \fi ``unassigned''.}%
          \fi
        \fi
    \else
        \xintMessage {xintexpr}{Warning}
            {Error: there was no such variable \XINT_unvar_tmpa\space
             to unassign.}%
    \fi
}%
```

32.25. Support for dummy variables

32.25.1 \xintnewdummy . . . 769
32.25.2 \xintensuredummy, \xintrestorevariable . . . 771
32.25.3 Checking (without expansion) that a symbolic expression contains correctly nested parentheses . . . 771
32.25.4 Fetching balanced expressions E1, E2 and a variable name Name from E1, Name=E2) . . . 772
32.25.5 Fetching a balanced expression delimited by a semi-colon . . . 772
32.25.6 Low-level support for omit and abort keywords, the break() function, the n++ construct and the semi-colon as used in the syntax of seq(), add(), mul(), iter(), rseq(), iterr(), rrseq(), subsm(), subsn(), ndseq(), ndmap() . . . 773
The n++ construct . . . 773
The break() function . . . 773
The omit and abort keywords . . . 773
The semi-colon . . . 774
32.25.7 Reserved dummy variables @, @1, @2, @3, @4, @@, @@(1), ..., @@@, @@@(1), ... for recursions . . . 774

32.25.1. \xintnewdummy

Comments under reconstruction.
  1.4 adds multi-letter names as usable dummy variables!

Modified at 1.4i (2021/06/11). s/onliteral/var*/ to fix the «seq renaming bug».

```
\catcode`* 11
\def\XINT_expr_makedummy #1%
{%
```

```
    \edef\XINT_tmpa{\xint_zapspaces #1 \xint_gobble_i}%
    \ifcsname XINT_expr_var_\XINT_tmpa\endcsname
      \XINT_global
      \expandafter\let\csname XINT_expr_var_\XINT_tmpa/old\expandafter\endcsname
                      \csname XINT_expr_var_\XINT_tmpa\expandafter\endcsname
    \fi
    \ifcsname XINT_expr_var*_\XINT_tmpa\endcsname
      \XINT_global
      \expandafter\let\csname XINT_expr_var*_\XINT_tmpa/old\expandafter
      \endcsname\csname XINT_expr_var*_\XINT_tmpa\expandafter\endcsname
    \fi
    \expandafter\XINT_global
    \expanded
    {\edef\expandafter\noexpand
          \csname XINT_expr_var_\XINT_tmpa\endcsname ##1\relax
          !\XINT_tmpa##2}%
       {{##2}##1\relax !\XINT_tmpa{##2}}%
    \expandafter\XINT_global
    \expanded
    {\edef\expandafter\noexpand
          \csname XINT_expr_var*_\XINT_tmpa\endcsname ##1\relax
          !\XINT_tmpa##2}%
       {\XINT_expr_prec_tacit *{##2}(##1\relax !\XINT_tmpa{##2}}%)
}%
\xintApplyUnbraced \XINT_expr_makedummy {abcdefghijklmnopqrstuvwxyz}%
\xintApplyUnbraced \XINT_expr_makedummy {ABCDEFGHIJKLMNOPQRSTUVWXYZ}%
\def\xintnewdummy #1{%
    \XINT_expr_makedummy{#1}%
    \ifxintverbose\xintMessage {xintexpr}{Info}%
       {\XINT_tmpa\space now
        \ifxintglobaldefs globally \fi usable as dummy variable.}%
    \fi
}%
\catcode`* 12
```

The nil variable was need in xint < 1.4 (with some other meaning) in places the syntax could not allow emptiness, such as ,,, and other things, but at 1.4 meaning as changed.

The other variables are new with 1.4. Don't use the None, it is tentative, and may be input as [].

Refactored at 1.4i to define them as really genuine variables, i.e. also with associated var* macros involved in tacit multiplication (even though it will be broken with nil, and with None in \xintiiexpr). No real reason, because \XINT_expr_op__ managed them fine even in absence of var* macros.

```
\XINT_expr_defvar_one{nil}{}%
\XINT_expr_defvar_one{None}{{}}% ? tentative
\XINT_expr_defvar_one{false}{{0}}% Maple, TeX
\XINT_expr_defvar_one{true}{{1}}%
\XINT_expr_defvar_one{False}{{0}}% Python
\XINT_expr_defvar_one{True}{{1}}%
\XINT_expr_defvar_one{NaN}{{I?}}%
\XINT_expr_defvar_one{Infinity}{{I+}}%
```

32.25.2. \xintensuredummy, \xintrestorevariable

1.3e \xintensuredummy differs from \xintnewdummy only in the informational message... Attention that this is not meant to be nested.

1.4 fixes that the message mentioned non-existent \xintrestoredummy (real name was \xintrestorelettervar and renames the latter to \xintrestorevariable as it applies also to multi-letter names.

```
\def\xintensuredummy #1{%
    \XINT_expr_makedummy{#1}%
    \ifxintverbose\xintMessage {xintexpr}{Info}%
       {\XINT_tmpa\space now
        \ifxintglobaldefs globally \fi usable as dummy variable.&&J
        Issue \string\xintrestorevariable{\XINT_tmpa}
        to restore former meaning.}%
    \fi
}%
\def\xintrestorevariablesilently #1{%
   \edef\XINT_tmpa{\xint_zapspaces #1 \xint_gobble_i}%
   \ifcsname XINT_expr_var_\XINT_tmpa/old\endcsname
     \XINT_global
     \expandafter\let\csname XINT_expr_var_\XINT_tmpa\expandafter\endcsname
                     \csname XINT_expr_var_\XINT_tmpa/old\expandafter\endcsname
   \fi
   \ifcsname XINT_expr_var*_\XINT_tmpa/old\endcsname
     \XINT_global
     \expandafter\let\csname XINT_expr_var*_\XINT_tmpa\expandafter\endcsname
                     \csname XINT_expr_var*_\XINT_tmpa/old\expandafter\endcsname
   \fi
}%
\def\xintrestorevariable #1{%
    \xintrestorevariablesilently {#1}%
    \ifxintverbose\xintMessage {xintexpr}{Info}%
       {\XINT_tmpa\space
        \ifxintglobaldefs globally \fi restored to its earlier status, if any.}%
    \fi
}%
```

32.25.3. Checking (without expansion) that a symbolic expression contains correctly nested parentheses

Expands to \xint_c_mone in case a closing) had no opening (matching it, to \@ne if opening (had no closing) matching it, to \z@ if expression was balanced. Call it as:

\XINT_isbalanced_a \relax #1(\xint_bye)\xint_bye

This is legacy f-expandable code not using \expanded even at 1.4.

```
\def\XINT_isbalanced_a #1({\XINT_isbalanced_b #1)\xint_bye }%
\def\XINT_isbalanced_b #1)#2%
   {\xint_bye #2\XINT_isbalanced_c\xint_bye\XINT_isbalanced_error }%
```

if #2 is not \xint_bye, a) was found, but there was no (. Hence error -> -1

```
\def\XINT_isbalanced_error #1)\xint_bye {\xint_c_mone}%
```

#2 was \xint_bye, was there a) in original #1?

```
\def\XINT_isbalanced_c\xint_bye\XINT_isbalanced_error #1%
    {\xint_bye #1\XINT_isbalanced_yes\xint_bye\XINT_isbalanced_d #1}%
```

#1 is \xint_bye, there was never (nor) in original #1, hence OK.

```
\def\XINT_isbalanced_yes\xint_bye\XINT_isbalanced_d\xint_bye )\xint_bye
    {\xint_c_ }%
```

#1 is not \xint_bye, there was indeed a (in original #1. We check if we see a). If we do, we then loop until no (nor) is to be found.

```
\def\XINT_isbalanced_d #1)#2%
   {\xint_bye #2\XINT_isbalanced_no\xint_bye\XINT_isbalanced_a #1#2}%
```

#2 was \xint_bye, we did not find a closing) in original #1. Error.

```
\def\XINT_isbalanced_no\xint_bye #1\xint_bye\xint_bye {\xint_c_i }%
```

32.25.4. Fetching balanced expressions E1, E2 and a variable name Name from E1, Name=E2)

Multi-letter dummy variables added at 1.4.

```
\def\XINT_expr_fetch_E_comma_V_equal_E_a #1#2,%
{%
    \ifcase\XINT_isbalanced_a \relax #1#2(\xint_bye)\xint_bye
           \expandafter\XINT_expr_fetch_E_comma_V_equal_E_c
        \or\expandafter\XINT_expr_fetch_E_comma_V_equal_E_b
      \else\expandafter\xintError:noopening
    \fi {#1#2},%
}%
\def\XINT_expr_fetch_E_comma_V_equal_E_b #1,%
   {\XINT_expr_fetch_E_comma_V_equal_E_a {#1,}}%
\def\XINT_expr_fetch_E_comma_V_equal_E_c #1,#2#3=%
{%
    \expandafter\XINT_expr_fetch_E_comma_V_equal_E_d\expandafter
    {\expanded{{\xint_zapspaces #2#3 \xint_gobble_i}}{#1}}{}%
}%
\def\XINT_expr_fetch_E_comma_V_equal_E_d #1#2#3)%
{%
    \ifcase\XINT_isbalanced_a \relax #2#3(\xint_bye)\xint_bye
        \or\expandafter\XINT_expr_fetch_E_comma_V_equal_E_e
      \else\expandafter\xintError:noopening
    \fi
    {#1}{#2#3}%
}%
\def\XINT_expr_fetch_E_comma_V_equal_E_e #1#2%
   {\XINT_expr_fetch_E_comma_V_equal_E_d {#1}{#2)}}%
```

32.25.5. Fetching a balanced expression delimited by a semi-colon

1.4. For subsn() leaner syntax of nested substitutions.

Will also serve to \xintdeffunc, to not have to hide inner semi-colons in for example an iter() from \xintdeffunc.

Adding brace removal protection for no serious reason, anyhow the xintexpr parsers always removes braces when moving forward, but well.

Trigger by \romannumeral\XINT_expr_fetch_to_semicolon upfront.

```
\def\XINT_expr_fetch_to_semicolon {\XINT_expr_fetch_to_semicolon_a {}\empty}%
\def\XINT_expr_fetch_to_semicolon_a #1#2;%
{%
    \ifcase\XINT_isbalanced_a \relax #1#2(\xint_bye)\xint_bye
```

```
            \xint_dothis{\expandafter\XINT_expr_fetch_to_semicolon_c}%
          \or\xint_dothis{\expandafter\XINT_expr_fetch_to_semicolon_b}%
        \else\expandafter\xintError:noopening
     \fi\xint_orthat{}\expandafter{#2}{#1}%
}%
\def\XINT_expr_fetch_to_semicolon_b #1#2%
   {\XINT_expr_fetch_to_semicolon_a {#2#1;}\empty}%
\def\XINT_expr_fetch_to_semicolon_c #1#2{\xint_c_{#2#1}}%
```

32.25.6. Low-level support for omit and abort keywords, the break() function, the n++ construct and the semi-colon as used in the syntax of seq(), add(), mul(), iter(), rseq(), iterr(), rrseq(), subsm(), subsn(), ndseq(), ndmap()

There is some clever play simply based on setting suitable precedence levels combined with special meanings given to op macros.

The special !? internal operator is a helper for omit and abort keywords in list generators. Prior to 1.4 support for +[, *[, ...,]+,]*, had some elements here.

The n++ construct 1.1 2014/10/29 did \expandafter\.=+\xintiCeil which transformed it into \romannumeral0\xinticeil, which seems a bit weird. This exploited the fact that dummy variables macros could back then pick braced material (which in the case at hand here ended being {\romannumeral0\xinticeil...} and were submitted to two expansions. The result of this was to provide a not value which got expanded only in the first loop of the :_A and following macros of seq, iter, rseq, etc...

Anyhow with 1.2c I have changed the implementation of dummy variables which now need to fetch a single locked token, which they do not expand.

The \xintiCeil appears a bit dispendious, but I need the starting value in a \numexpr compatible form in the iteration loops.

```
\expandafter\def\csname XINT_expr_itself_++\endcsname {++}%
\expandafter\def\csname XINT_expr_itself_++)\endcsname {++)}%
\expandafter\let\csname XINT_expr_precedence_++)\endcsname \xint_c_i
\xintFor #1 in {expr,flexpr,iiexpr} \do {%
    \expandafter\def\csname XINT_#1_op_++)\endcsname ##1##2\relax
     {\expandafter\XINT_expr_foundend
                  \expanded{{+{\XINT:NEhook:f:one:from:one:direct%
                               \calxCeilToNum##1}}}%
     }%
}%
```

The break() function break is a true function, the parsing via expansion of the enclosed material proceeds via _oparen macros as with any other function.

```
\catcode`? 3
\def\XINT_expr_func_break #1#2#3{#1#2{?#3}}%
\catcode`? 11
\let\XINT_flexpr_func_break \XINT_expr_func_break
\let\XINT_iiexpr_func_break \XINT_expr_func_break
```

The omit and abort keywords Comments are currently undergoing reconstruction.

The mechanism is somewhat complex. The operator !? will fetch a dummy value ! or ^ which is then recognized int the loops implementing the various seq etc... construct using dummy variables and implement omit and abort.

In May 2021 I realized that the January 2020 1.4 had broken omit and abort if used inside a subs(). The definition

```
\edef\XINT_expr_var_omit  #1\relax !{1\string !?!\relax !}
```

conflicted with the 1.4 refactoring of «subs» and similar things which had replaced formerly clean-up macros (of ! and what's next, as in now defunct `\def\XINT_expr_subx:_end #1!#2#3{#1}` which was involved in subs mechanism, and by the way would be incompatible with multi-letter dummy variables) by usage of an `\iffalse` as in "`\relax\iffalse\relax !`" to delimite a sub-expression, which was supposed to be clever (the "`\relax !`" being delimiter for dummy variables).

This `\iffalse` from subs mechanism ended up being gobbled by omit/abort thus inducing breakage.

Grabbing `\relax #2!` would be a fix but looks a bit dangerous, as there can be a subexpression after the omit or abort bringing its own `\relax`, although this is very very unlikely.

I considered to modify the dummy variables delimiter from `\relax !` to `\xint_Bye !` for example but got afraid from the ramifications, as all structures handling dummy variables would have needed refactoring.

So finally things here remain unchanged and the refactoring to fix this breakage was done in `\XINT_allexpr_subsx` (and also subsm). Done at 1.4h. See `\XINT_allexpr_subsx` for comments.

```
\edef\XINT_expr_var_omit  #1\relax !{1\string !?!\relax !}%
\edef\XINT_expr_var_abort #1\relax !{1\string !?^\relax !}%
\def\XINT_expr_itself_!? {!?}%
\def\XINT_expr_op_!? #1#2\relax{\XINT_expr_foundend{#2}}%
\let\XINT_iiexpr_op_!? \XINT_expr_op_!?
\let\XINT_flexpr_op_!? \XINT_expr_op_!?
\let\XINT_expr_precedence_!? \xint_c_iv
```

The semi-colon Obsolete comments undergoing re-construction

```
\xintFor #1 in {expr,flexpr,iiexpr} \do {%
    \expandafter\def\csname XINT_#1_op_;\endcsname {\xint_c_i ;}%
}%
\expandafter\let\csname XINT_expr_precedence_;\endcsname\xint_c_i
\expandafter\def\csname XINT_expr_itself_;)\endcsname {)}%
\expandafter\let\csname XINT_expr_precedence_;)\endcsname\xint_c_i
```

32.25.7. Reserved dummy variables @, @1, @2, @3, @4, @@, @@(1), ..., @@@, @@@(1), ... for recursions

Comments currently under reconstruction.

1.4 breaking change: @ and @1 behave differently and one can not use @ in place of @1 in iterr() and rrseq(). Formerly @ and @1 had the same definition.

Brace stripping in `\XINT_expr_func_@@` is prevented by some ending 0 or other token see iterr() and rrseq() code.

For the record, the ~ and ? have catcode 3 in this code.

```
\catcode`* 11
\def\XINT_expr_var_@ #1~#2{{#2}#1~{#2}}%
\def\XINT_expr_var*_@ #1~#2{\XINT_expr_prec_tacit *{#2}(#1~{#2}}%
\expandafter
\def\csname XINT_expr_var_@1\endcsname #1~#2{{{#2}}#1~{#2}}%
\expandafter
\def\csname XINT_expr_var_@2\endcsname #1~#2#3{{{#3}}#1~{#2}{#3}}%
\expandafter
\def\csname XINT_expr_var_@3\endcsname #1~#2#3#4{{{#4}}#1~{#2}{#3}{#4}}%
\expandafter
\def\csname XINT_expr_var_@4\endcsname #1~#2#3#4#5{{{#5}}#1~{#2}{#3}{#4}{#5}}%
```

```
\expandafter\def\csname XINT_expr_var*_@1\endcsname #1~#2%
                {\XINT_expr_prec_tacit *{{#2}}(#1~{#2}}%
\expandafter\def\csname XINT_expr_var*_@2\endcsname #1~#2#3%
                {\XINT_expr_prec_tacit *{{#3}}(#1~{#2}{#3}}%
\expandafter\def\csname XINT_expr_var*_@3\endcsname #1~#2#3#4%
                {\XINT_expr_prec_tacit *{{#4}}(#1~{#2}{#3}{#4}}%
\expandafter\def\csname XINT_expr_var*_@4\endcsname #1~#2#3#4#5%
                {\XINT_expr_prec_tacit *{{#5}}(#1~{#2}{#3}{#4}{#5}}%
\catcode`* 12
\catcode`? 3
\def\XINT_expr_func_@@ #1#2#3#4~#5?%
{%
   \expandafter#1\expandafter#2\expandafter{\expandafter{%
         \romannumeral0\xintntheltnoexpand{\calxListItemToNum#3}{#5}}}#4~#5?%
}%
\def\XINT_expr_func_@@@ #1#2#3#4~#5~#6?%
{%
   \expandafter#1\expandafter#2\expandafter{\expandafter{%
         \romannumeral0\xintntheltnoexpand{\calxListItemToNum#3}{#6}}}%
         #4~#5~#6?%
}%
\def\XINT_expr_func_@@@@ #1#2#3#4~#5~#6~#7?%
{%
   \expandafter#1\expandafter#2\expandafter{\expandafter{%
         \romannumeral0\xintntheltnoexpand{\calxListItemToNum#3}{#7}}}%
         #4~#5~#6~#7?%
}%
\let\XINT_flexpr_func_@@\XINT_expr_func_@@
\let\XINT_flexpr_func_@@@\XINT_expr_func_@@@
\let\XINT_flexpr_func_@@@@\XINT_expr_func_@@@@
\def\XINT_iiexpr_func_@@ #1#2#3#4~#5?%
{%
   \expandafter#1\expandafter#2\expandafter{\expandafter{%
         \romannumeral0\xintntheltnoexpand{\CALX_mr#3}{#5}}}#4~#5?%
}%
\def\XINT_iiexpr_func_@@@ #1#2#3#4~#5~#6?%
{%
   \expandafter#1\expandafter#2\expandafter{\expandafter{%
         \romannumeral0\xintntheltnoexpand{\CALX_mr#3}{#6}}}#4~#5~#6?%
}%
\def\XINT_iiexpr_func_@@@@ #1#2#3#4~#5~#6~#7?%
{%
   \expandafter#1\expandafter#2\expandafter{\expandafter{%
         \romannumeral0\xintntheltnoexpand{\CALX_mr#3}{#7}}}#4~#5~#6~#7?%
}%
\catcode`? 11
```

32.26. Pseudo-functions involving dummy variables and generating scalars or sequences

32.26.1 Comments 776
32.26.2 subs(): substitution of one variable 777

32.26.3 subsm(): simultaneous independent substitutions . . . 778
32.26.4 subsn(): leaner syntax for nesting (possibly dependent) substitutions . . . 780
32.26.5 seq(): sequences from assigning values to a dummy variable . . . 781
32.26.6 iter() . . . 782
32.26.7 add(), mul() . . . 784
32.26.8 rseq() . . . 785
32.26.9 iterr() . . . 786
32.26.10 rrseq() . . . 787

32.26.1. Comments

Comments added 2020/01/16.

The mechanism for «seq» is the following. When the parser encounters «seq», which means it parsed these letters and encountered (from expansion) an opening parenthesis, the \XINT_expr_func mechanism triggers the «`» operator which realizes that «seq» is a pseudo-function (there is no _func_seq) and thus spans the \XINT_expr_onliteral_seq macro (currently this means however that the knowledge of which parser we are in is lost, see comments of \XINT_expr_op_` code). The latter will use delimited macros and parenthesis check to fetch (without any expansion), the symbolic expression ExprSeq to evaluate, the Name (now possibly multi-letter) of the variable and the expression ExprValues to evaluate which will give the values to assign to the dummy variable Name. It then positions upstream ExprValues suitably terminated (see next) and after it {{Name}{ExprSeq}}. Then it inserts a second call to the «`» operator with now «seqx» as argument hence the appropriate «{,fl,ii}expr_func_seqx» macros gets executed. The general way function macros work is that first all their arguments are evaluated via a call not to \xintbare{,float,ii}eval but to the suitable \XINT_{expr,flexpr,iiexpr}_oparen core macro which does almost same excepts it expects a final closing parenthesis (of course allowing nested parenthesis in-between) and stops there. Here, this closing parenthesis got positioned deliberately with a \relax after it, so the parser, which always after having gathered a value looks ahead to find the next operator, thinks it has hit the end of the expression and as result inserts a \xint_c_ (i.e. \z@) token for precedence level and a dummy \relax token (place-holder for a non-existing operator). Generally speaking «func_foo» macros expect to be executed with three parameters #1#2#3, #1 = precedence, #2 = operator, #3 = values (call it «args») i.e. the fully evaluated list of all its arguments. The special «func_seqx» and cousins know that the first two tokens are trash and they now proceed forward, having thus lying before them upstream the values to loop over, now fully evaluated, and {{Name}{ExprSeq}}. It then positions appropriately ExprSeq inside a sub-expression and after it, following suitable delimiter, Name and the evaluated values to assign to Name.

Dummy variables are essentially simply delimited macros where the delimiter is the variable name preceded by a \relax token and a catcode 11 exclamation point. Thus the various «subsx», «seqx», «iterx» position the tokens appropriately and launch suitable loops.

All of this nests well, inner «seq»'s (or more often in practice «subs»'s) being allowed to refer to the dummy variables used by outer «seq»'s because the outer «seq»'s have the values to assign to their variables evaluated first and their ExprSeq evaluated last. For inner dummy variables to be able to refer to outer dummy variables the author must be careful of course to not use in the implementation braces { and } which would break dummy variables to fetch values beyond the closing brace.

The above «seq» mechanism was done around June 15-25th 2014 at the time of the transition from 1.09n to 1.1 but already in October 2014 I made a note that I had a hard time to understand it again:

« [START OF YEAR 2014 COMMENTS]

All of seq, add, mul, rseq, etc... (actually all of the extensive changes from xintexpr 1.09n to 1.1) was done around June 15-25th 2014, but the problem is that I did not document the code enough, and I had a hard time understanding in October what I had done in June. Despite the lesson, again being short on time, I do not document enough my current understanding of the innards of the beast...

I added subs, and iter in October (also the [:n], [n:] list extractors), proving I did at least understand a bit (or rather could imitate) my earlier code (but don't ask me to explain \xintNewExpr !)

The \XINT_expr_fetch_E_comma_V_equal_E_a parses: "expression, variable=list)" (when it is called the opening (has been swallowed, and it looks for the ending one.) Both expression and list may themselves contain parentheses and commas, we allow nesting. For example "x^2,x=1..10)", at the end of seq_a we have {variable{expression}}{list}, in this example {x{x^2}}{1..10}, or more complicated "seq(add(y,y=1..x),x=1..10)" will work too. The variable is a single lowercase Latin letter.

The complications with \xint_c_ii^v in seq_f is for the recurrent thing that we don't know in what type of expressions we are, hence we must move back up, with some loss of efficiency (superfluous check for minus sign, etc...). But the code manages simultaneously expr, flexpr and iiexpr.

[END OF YEAR 2014 OLD COMMENTS]»

On Jeudi 16 janvier 2020 à 15:13:32 I finally did the documentation as above.

The case of «iter», «rseq», «iterr», «rrseq» differs slightly because the initial values need evaluation. This is done by genuine functions \XINT_<parser>_func_iter etc... (there was no \XINT_<parser>_func_seq). The trick is via the semi-colon ; which is a genuine operator having the precedence of a closing parenthesis and whose action is only to stop expansion. Thus this first step of gathering the initial values is done as part of the reguler expansion job of the parser not using delimited macros and the ; can be hidden in braces {;} because the three parsers when moving forward remove one level of braces always. Thus \XINT_<parser>_func_seq simply hand over to \XINT_allexpr_iter which will then trigger the fetching without expansion of ExprIter, Name=ExprValues as described previously for «seq».

With 1.4, multi-letter names for dummy variables are allowed.

Also there is the additional 1.4 ambition to make the whole thing parsable by \xintNewExpr/\xintdeffunc. This is done by checking if all is numerical, because the omit, abort and break() mechanisms have no translation into macros, and the only solution for symbolic material is to simply keep it as is, so that expansion will again activate the xintexpr parsers. At 1.4 this approach is fine although the initial goals of \xintNewExpr/\xintdeffunc was to completely replace the parsers (whose storage method hit the string pool formerly) by macros. Now that 1.4 does not impact the string pool we can make \xintdeffunc much more powerful but it will not be a construct using only xintfrac macros, it will still be partially the \xintexpr etc... parsers in such cases.

Got simpler with 1.2c as now the dummy variable fetches an already encapsulated value, which is anyhow the form in which we get it.

Refactored at 1.4 using \expanded rather than \csname.

And support for multi-letter variables, which means function declarations can now use multi-letter variables !

Modified at 2.0a1 (2026/11/01). Per parser "onliteral" macros for matters UNRELATED with subs() et al. Here defining the \xinteval interface. The needed \let done below.

32.26.2. subs(): substitution of one variable

```
\def\XINT_expr_onliteral_subs
{%
    \expandafter\XINT_allexpr_subs_f
    \romannumeral`&&@\XINT_expr_fetch_E_comma_V_equal_E_a {}%
}%
\def\XINT_allexpr_subs_f #1#2{\xint_c_ii^v `{subsx}#2)\relax #1}%
\def\XINT_expr_func_subsx   #1#2{\XINT_allexpr_subsx \xintbareeval }%
\def\XINT_flexpr_func_subsx #1#2{\XINT_allexpr_subsx \xintbarefloateval}%
\def\XINT_iiexpr_func_subsx #1#2{\XINT_allexpr_subsx \xintbareiieval }%
```

#2 is the value to assign to the dummy variable #3 is the dummy variable name (possibly multi-letter), #4 is the expression to evaluate

1.4 was doing something clever to get rid of the ! and tokens following it, via an \iffalse...\fi which erased them and propagated the expansion to trigger the getopt:

```
\expanded\bgroup\romannumeral0#1#4\relax \iffalse\relax !#3{#2}{\fi\expandafter}
```

But sadly, with a delay of more than one year later (right after having released 1.4g) I realized that this had broken omit and abort if inside a subs. As omit and abort would clean all up to \relax !, this meant here swallowing in particular the above \iffalse, leaving a dangling \fi. I had the files which show this bug already at time of 1.4 release but did not compile them, and they were not included in my test suite.

I hesitated with modifying the delimiter from "\relax !<varname>" (catcode 11 !) to "\relax \xint_Bye<varname>" for the dummy variables which would have allowed some trickery using \xint_Bye...\xint_bye clean-up but got afraid from the breakage potential of such refactoring with many induced changes.

A variant like this:

```
\def\XINT_allexpr_subsx #1#2#3#4
{
    \expandafter\XINT_expr_clean_and_put_op_first
    \expanded
    {\romannumeral0#1#4\relax !#3{#2}\xint:\expandafter}\romannumeral`&&\XINT_expr_getop
}
\def\XINT_expr_clean_and_put_op_first #1#2\xint:#3#4{#3#4{#1}}
```

breaks nesting: the braces make variables encountered in #4 unable to match their definition. This would work:

```
\def\XINT_allexpr_subsx #1#2#3#4
{
    \expandafter\XINT_allexpr_subsx_clean\romannumeral0#1#4\relax !#3{#2}\xint:
}
\def\XINT_allexpr_subsx_clean #1#2\xint:
{
    \expandafter\XINT_expr_put_op_first
    \expanded{\xint_noxpd{{#1}}\expandafter}\romannumeral`&&@\XINT_expr_getop
}
```

(not tested).

But in the end I decided to simply fix the first envisioned code above. This accepts expansion of supposedly inert #3{#2}. There is again the \iffalse but it is moved to the right. This change limits possibly hacky future developments. Done at 1.4h (2021/01/27).

No need for the \expandafter's from \XINT_expr_put_op_first in \XINT_expr_clean_and_put_op_first.

```
\def\XINT_allexpr_subsx #1#2#3#4%
{%
    \expandafter\XINT_expr_clean_and_put_op_first
    \expanded
    \bgroup\romannumeral0#1#4\relax !#3{#2}\xint:\iffalse{\fi\expandafter}%
    \romannumeral`&&@\XINT_expr_getop
}%
\def\XINT_expr_clean_and_put_op_first #1#2\xint:#3#4{#3#4{#1}}%
```

32.26.3. subsm(): simultaneous independent substitutions

New with 1.4. Globally the var1=expr1; var2=expr2; var2=expr3;... part can arise from expansion, except that once a semi-colon has been found (from expansion) the varK= thing following it must be

there. And as for subs() the final parenthesis must be there from the start.

```
\def\XINT_expr_onliteral_subsm
{%
    \expandafter\XINT_allexpr_subsm_f
    \romannumeral`&&@\XINT_expr_fetch_E_comma_V_equal_E_a {}%
}%
\def\XINT_allexpr_subsm_f #1#2{\xint_c_ii^v `{subsmx}#2)\relax #1}%
\def\XINT_expr_func_subsmx
{%
    \expandafter\XINT_allexpr_subsmx\expandafter\xintbareeval
    \expanded\bgroup{\iffalse}\fi\XINT_allexpr_subsm_A\XINT_expr_oparen
}%
\def\XINT_flexpr_func_subsmx
{%
    \expandafter\XINT_allexpr_subsmx\expandafter\xintbarefloateval
    \expanded\bgroup{\iffalse}\fi\XINT_allexpr_subsm_A\XINT_flexpr_oparen
}%
\def\XINT_iiexpr_func_subsmx
{%
    \expandafter\XINT_allexpr_subsmx\expandafter\xintbareiieval
    \expanded\bgroup{\iffalse}\fi\XINT_allexpr_subsm_A\XINT_iiexpr_oparen
}%
\def\XINT_allexpr_subsm_A #1#2#3%
{%
    \ifx#2\xint_c_
       \expandafter\XINT_allexpr_subsm_done
    \else
       \expandafter\XINT_allexpr_subsm_B
    \fi #1%
}%
\def\XINT_allexpr_subsm_B #1#2#3#4=%
{%
    {#2}\relax !\xint_zapspaces#3#4 \xint_gobble_i
    \expandafter\XINT_allexpr_subsm_A\expandafter#1\romannumeral`&&@#1%
}%
```

```
   #1 = \xintbareeval,  or \xintbarefloateval or \xintbareiieval
   #2 = evaluation of last variable assignment
```

```
\def\XINT_allexpr_subsm_done #1#2{{#2}\iffalse{{\fi}}}%
```

```
   #1 = \xintbareeval or \xintbarefloateval or \xintbareiieval
   #2 = {value1}\relax !var2{value2}....\relax !varN{valueN} (value's may be oples)
   #3 = {var1}
   #4 = the expression to evaluate
```

Refactored at 1.4h as for \XINT_allexpr_subsx, see comments there related to the omit/abort conundrum.

```
\def\XINT_allexpr_subsmx #1#2#3#4%
{%
    \expandafter\XINT_expr_clean_and_put_op_first
    \expanded
    \bgroup\romannumeral0#1#4\relax !#3#2\xint:\iffalse{\fi\expandafter}%
    \romannumeral`&&@\XINT_expr_getop
}%
```

32.26.4. subsn(): leaner syntax for nesting (possibly dependent) substitutions

New with 1.4. 2020/01/24

```
\def\XINT_expr_onliteral_subsn
{%
    \expandafter\XINT_allexpr_subsn_f
    \romannumeral`&&@\XINT_expr_fetch_E_comma_V_equal_E_a {}%
}%
\def\XINT_allexpr_subsn_f #1{\XINT_allexpr_subsn_g #1}%
```

```
   #1 = Name1
   #2 = Expression in all variables which is to evaluate
   #3 = all the stuff after Name1 =  and up to final parenthesis
```

This one needed no reactoring at 1.4h to fix the omit/abort problem, as there was no \iffalse..\fi clean-up: the clean-up is done directly via \XINT_allexpr_subsnx_J.

I only added usage of \XINT_expr_put_op_first_noexpand. There may be other locations where it could be used, but I can't afford now reviewing usage. For next release after 1.4h bugfix.

```
\def\XINT_allexpr_subsn_g #1#2#3%
{%
    \expandafter\XINT_allexpr_subsn_h
    \expanded\bgroup{\iffalse}\fi\expandafter\XINT_allexpr_subsn_B
    \romannumeral\XINT_expr_fetch_to_semicolon #1=#3;\hbox=;;^{#2}%
}%
\def\XINT_allexpr_subsn_B #1{\XINT_allexpr_subsn_C #1\vbox}%
\def\XINT_allexpr_subsn_C #1#2=#3\vbox
{%
    \ifx\hbox#1\iffalse{{\fi}\expandafter}\else
    {{\xint_zapspaces #1#2 \xint_gobble_i}};\xint_noxpd{{{#3}}}%
    \expandafter\XINT_allexpr_subsn_B
    \romannumeral\expandafter\XINT_expr_fetch_to_semicolon\fi
}%
```

Modified at 1.4p (2026/03/01).

I used here formerly \xintundefined as a delimiter token, perhaps also initially as debugging helper in case expansion went beyond the expected, but elsewhere in the code base there is \XINT_undefined, but both feel inappopriate. The problem attow is that I wrote the code many years ago and I can not be fully certain using \xint: may not cause a problem in case this delimiter could appear earlier, but I think not, so let's go for \xint:.

```
\def\XINT_allexpr_subsn_h
{%
    \xint_c_ii^v `{subsnx}\romannumeral0\xintreverseorder
}%
\def\XINT_expr_func_subsnx #1#2#3#4#5;#6%
{%
    \xint_gob_til_^ #6\XINT_allexpr_subsnx_H ^%
    \expandafter\XINT_allexpr_subsnx\expandafter
    \xintbareeval\romannumeral0\xintbareeval #5\relax !#4{#3}\xint:
    {\relax !#4{#3}\relax !#6}%
}%
\def\XINT_iiexpr_func_subsnx #1#2#3#4#5;#6%
{%
    \xint_gob_til_^ #6\XINT_allexpr_subsnx_H ^%
    \expandafter\XINT_allexpr_subsnx\expandafter
    \xintbareiieval\romannumeral0\xintbareiieval #5\relax !#4{#3}\xint:
```

```
    {\relax !#4{#3}\relax !#6}%
}%
\def\XINT_flexpr_func_subsnx #1#2#3#4#5;#6%
{%
    \xint_gob_til_^ #6\XINT_allexpr_subsnx_H ^%
    \expandafter\XINT_allexpr_subsnx\expandafter
    \xintbarefloateval\romannumeral0\xintbarefloateval #5\relax !#4{#3}\xint:
    {\relax !#4{#3}\relax !#6}%
}%
\def\XINT_allexpr_subsnx #1#2!#3\xint:#4#5;#6%
{%
    \xint_gob_til_^ #6\XINT_allexpr_subsnx_I ^%
    \expandafter\XINT_allexpr_subsnx\expandafter
    #1\romannumeral0#1#5\relax !#4{#2}\xint:
    {\relax !#4{#2}\relax !#6}%
}%
\def\XINT_allexpr_subsnx_H ^#1\romannumeral0#2#3!#4\xint:#5#6%
{%
    \expandafter\XINT_allexpr_subsnx_J\romannumeral0#2#6#5%
}%
\def\XINT_allexpr_subsnx_I ^#1\romannumeral0#2#3\xint:#4#5%
{%
    \expandafter\XINT_allexpr_subsnx_J\romannumeral0#2#5#4%
}%
\def\XINT_allexpr_subsnx_J #1#2^%
{%
    \expandafter\XINT_expr_put_op_first_noexpand
    \expanded{\xint_noxpd{{#1}}\expandafter}\romannumeral`&&@\XINT_expr_getop
}%
\def\XINT_expr_put_op_first_noexpand#1#2#3{#2#3{#1}}%
```

32.26.5. seq(): sequences from assigning values to a dummy variable

In seq_f, the #2 is the ExprValues expression which needs evaluation to provide the values to the dummy variable and #1 is {Name}{ExprSeq} where Name is the name of dummy variable and {ExprSeq} the expression which will have to be evaluated.

```
\def\XINT_allexpr_seq_f #1#2{\xint_c_ii^v `{seqx}#2)\relax #1}%
\def\XINT_expr_onliteral_seq
 {\expandafter\XINT_allexpr_seq_f
  \romannumeral`&&@\XINT_expr_fetch_E_comma_V_equal_E_a {}}%
\def\XINT_expr_func_seqx   #1#2%
 {\XINT:NEhook:seqx\XINT_allexpr_seqx\xintbareeval }%
\def\XINT_flexpr_func_seqx #1#2%
 {\XINT:NEhook:seqx\XINT_allexpr_seqx\xintbarefloateval}%
\def\XINT_iiexpr_func_seqx #1#2%
 {\XINT:NEhook:seqx\XINT_allexpr_seqx\xintbareiieval }%
\def\XINT_allexpr_seqx #1#2#3#4%
{%
    \expandafter\XINT_expr_put_op_first
    \expanded \bgroup {\iffalse}\fi\XINT_expr_seq:_b {#1#4\relax !#3}#2^%
    \XINT_expr_cb_and_getop
}%
```

```
\def\XINT_expr_cb_and_getop{\iffalse{\fi\expandafter}%
    \romannumeral`&&@\XINT_expr_getop}%
```

Comments undergoing reconstruction.

```
\catcode`? 3
\def\XINT_expr_seq:_b #1#2%
{%
    \ifx +#2\xint_dothis\XINT_expr_seq:_Ca\fi
    \ifx !#2!\xint_dothis\XINT_expr_seq:_noop\fi
    \ifx ^#2\xint_dothis\XINT_expr_seq:_end\fi
    \xint_orthat{\XINT_expr_seq:_c}{#2}{#1}%
}%
\def\XINT_expr_seq:_noop #1{\XINT_expr_seq:_b }%
\def\XINT_expr_seq:_end  #1#2{\iffalse{\fi}}%
\def\XINT_expr_seq:_c #1#2%
   {\expandafter\XINT_expr_seq:_d\romannumeral0#2{{#1}}{#2}}%
\def\XINT_expr_seq:_d #1{\ifx ^#1\xint_dothis\XINT_expr_seq:_abort\fi
                         \ifx ?#1\xint_dothis\XINT_expr_seq:_break\fi
                         \ifx !#1\xint_dothis\XINT_expr_seq:_omit\fi
                         \xint_orthat{\XINT_expr_seq:_goon {#1}}}%
\def\XINT_expr_seq:_abort #1!#2^{\iffalse{\fi}}%
\def\XINT_expr_seq:_break #1!#2^{#1\iffalse{\fi}}%
\def\XINT_expr_seq:_omit  #1!#2#{\expandafter\XINT_expr_seq:_b\xint_gobble_i}%
\def\XINT_expr_seq:_goon  #1!#2#%
   {#1\expandafter\XINT_expr_seq:_b\xint_gobble_i}%
\def\XINT_expr_seq:_Ca #1#2#3{\XINT_expr_seq:_Cc#3.{#2}}%
\def\XINT_expr_seq:_Cb #1%
  {\expandafter\XINT_expr_seq:_Cc\the\numexpr#1+\xint_c_i.}%
\def\XINT_expr_seq:_Cc #1.#2%
  {\expandafter\XINT_expr_seq:_D\romannumeral0#2{{#1[0]}}{#1}{#2}}%
\def\XINT_expr_seq:_D #1{\ifx ^#1\xint_dothis\XINT_expr_seq:_abort\fi
                         \ifx ?#1\xint_dothis\XINT_expr_seq:_break\fi
                         \ifx !#1\xint_dothis\XINT_expr_seq:_Omit\fi
                         \xint_orthat{\XINT_expr_seq:_Goon {#1}}}%
\def\XINT_expr_seq:_Omit #1!#2#{\expandafter\XINT_expr_seq:_Cb\xint_gobble_i}%
\def\XINT_expr_seq:_Goon #1!#2#{#1\expandafter\XINT_expr_seq:_Cb\xint_gobble_i}%
```

32.26.6. iter()

Prior to 1.2g, the iter keyword was what is now called iterr, analogous with rrseq. Somehow I forgot an iter functioning like rseq with the sole difference of printing only the last iteration. Both rseq and iter work well with list selectors, as @ refers to the whole comma separated sequence of the initial values. I have thus deliberately done the backwards incompatible renaming of iter to iterr, and the new iter.

To understand the tokens which are presented to \XINT_allexpr_iter it is needed to check elsewhere in the source code how the ; hack is done.

The #2 in \XINT_allexpr_iter is \xint_c_i from the ; hack. Formerly (xint < 1.4) there was no such token. The change is motivated to using ; also in subsm() syntax.

```
\def\XINT_expr_func_iter   {\XINT_allexpr_iter \xintbareeval      }%
\def\XINT_flexpr_func_iter {\XINT_allexpr_iter \xintbarefloateval }%
\def\XINT_iiexpr_func_iter {\XINT_allexpr_iter \xintbareiieval    }%
\def\XINT_allexpr_iter #1#2#3#4%
{%
```

```
    \expandafter\XINT_expr_iterx
    \expandafter#1\expanded{\xint_noxpd{{#4}}\expandafter}%
    \romannumeral`&&@\XINT_expr_fetch_E_comma_V_equal_E_a {}%
}%
\def\XINT_expr_iterx #1#2#3#4%
{%
    \XINT:NEhook:iter\XINT_expr_itery\romannumeral0#1(#4)\relax {#2}#3#1%
}%
\def\XINT_expr_itery #1#2#3#4#5%
{%
    \expandafter\XINT_expr_put_op_first
    \expanded \bgroup {\iffalse}\fi
    \XINT_expr_iter:_b {#5#4\relax !#3}#1^~{#2}\XINT_expr_cb_and_getop
}%
\def\XINT_expr_iter:_b #1#2%
{%
    \ifx +#2\xint_dothis\XINT_expr_iter:_Ca\fi
    \ifx !#2!\xint_dothis\XINT_expr_iter:_noop\fi
    \ifx ^#2\xint_dothis\XINT_expr_iter:_end\fi
    \xint_orthat{\XINT_expr_iter:_c}{#2}{#1}%
}%
\def\XINT_expr_iter:_noop #1{\XINT_expr_iter:_b }%
\def\XINT_expr_iter:_end  #1#2~#3{#3\iffalse{\fi}}%
\def\XINT_expr_iter:_c #1#2%
   {\expandafter\XINT_expr_iter:_d\romannumeral0#2{{#1}}{#2}}%
\def\XINT_expr_iter:_d #1{\ifx ^#1\xint_dothis\XINT_expr_iter:_abort\fi
                          \ifx ?#1\xint_dothis\XINT_expr_iter:_break\fi
                          \ifx !#1\xint_dothis\XINT_expr_iter:_omit\fi
                          \xint_orthat{\XINT_expr_iter:_goon {#1}}}%
\def\XINT_expr_iter:_abort #1!#2^~#3{#3\iffalse{\fi}}%
\def\XINT_expr_iter:_break #1!#2^~#3{#1\iffalse{\fi}}%
\def\XINT_expr_iter:_omit  #1!#2#{%
   \expandafter\XINT_expr_iter:_b\xint_gobble_i
}%
\def\XINT_expr_iter:_goon  #1!#2#{\XINT_expr_iter:_goon_a {#1}}%
\def\XINT_expr_iter:_goon_a #1#2#3~#4{\XINT_expr_iter:_b #3~{#1}}%
\def\XINT_expr_iter:_Ca #1#2#3{\XINT_expr_iter:_Cc#3.{#2}}%
\def\XINT_expr_iter:_Cb #1%
  {\expandafter\XINT_expr_iter:_Cc\the\numexpr#1+\xint_c_i.}%
\def\XINT_expr_iter:_Cc #1.#2%
  {\expandafter\XINT_expr_iter:_D\romannumeral0#2{{#1[0]}}{#1}{#2}}%
\def\XINT_expr_iter:_D #1{\ifx ^#1\xint_dothis\XINT_expr_iter:_abort\fi
                          \ifx ?#1\xint_dothis\XINT_expr_iter:_break\fi
                          \ifx !#1\xint_dothis\XINT_expr_iter:_Omit\fi
                          \xint_orthat{\XINT_expr_iter:_Goon {#1}}}%
\def\XINT_expr_iter:_Omit  #1!#2#{%
  \expandafter\XINT_expr_iter:_Cb\xint_gobble_i
}%
\def\XINT_expr_iter:_Goon  #1!#2#{\XINT_expr_iter:_Goon_a {#1}}%
\def\XINT_expr_iter:_Goon_a #1#2#3~#4{\XINT_expr_iter:_Cb #3~{#1}}%
```

32.26.7. add(), mul()

Comments under reconstruction.

These were a bit anomalous as they did not implement omit and abort keyword and the break() function (and per force then neither the n++ syntax).

At 1.4 they are simply mapped to using adequately iter(). Thus, there is small loss in efficiency, but supporting omit, abort and break is important. Using dedicated macros here would have caused also slight efficiency drop. Simpler to remove the old approach.

```
\def\XINT_expr_onliteral_add
 {\expandafter\XINT_allexpr_add_f
  \romannumeral`&&@\XINT_expr_fetch_E_comma_V_equal_E_a {}}%
\def\XINT_allexpr_add_f #1#2{\xint_c_ii^v `{opx}#2)\relax #1{+}{0[0]}}%
\def\XINT_expr_onliteral_mul
 {\expandafter\XINT_allexpr_mul_f
  \romannumeral`&&@\XINT_expr_fetch_E_comma_V_equal_E_a {}}%
\def\XINT_allexpr_mul_f #1#2{\xint_c_ii^v `{opx}#2)\relax #1{*}{1[0]}}%
\def\XINT_expr_func_opx
  {\XINT:NEhook:opx \XINT_allexpr_opx \xintbareeval     }%
\def\XINT_flexpr_func_opx
  {\XINT:NEhook:opx \XINT_allexpr_opx \xintbarefloateval}%
\def\XINT_iiexpr_func_opx
  {\XINT:NEhook:opx \XINT_allexpr_opx \xintbareiieval   }%
```

1.4a In case of usage of omit (did I not test it? obviously I didn't as neither omit nor abort could work; and break neither), 1.4 code using (#6) syntax caused a (somewhat misleading) «missing)» error message which originated in the #6. This is non-obvious problem (perhaps explained why prior to 1.4 I had not added support for omit and break() to add() and mul()...

Allowing () is not enough as it would have to be 0 or 1 depending on whether we are using add() or mul(). Hence the somewhat complicated detour (relying on precise way var_omit and var_abort work) via \XINT_allexpr_opx_ifnotomitted.

\break() has special meaning here as it is used as last operand, not as last value. The code is very unsatisfactory and inefficient but this is hotfix for 1.4a.

```
\def\XINT_allexpr_opx #1#2#3#4#5#6#7#8%
{%
    \expandafter\XINT_expr_put_op_first
    \expanded \bgroup {\iffalse}\fi
    \XINT_expr_iter:_b {#1%
    \expandafter\XINT_allexpr_opx_ifnotomitted
    \romannumeral0#1#6\relax#7@\relax !#5}#4^~{{#8}}\XINT_expr_cb_and_getop
}%
\def\XINT_allexpr_opx_ifnotomitted #1%
{%
    \ifx !#1\xint_dothis{@\relax}\fi
    \ifx ^#1\xint_dothis{\XINTfstop. ^\relax}\fi
    \if ?\xintFirstItem{#1}\xint_dothis{\XINT_allexpr_opx_break{#1}}\fi
    \xint_orthat{\XINTfstop.{#1}}%
}%
\def\XINT_allexpr_opx_break #1#2\relax
{%
    break%
    (\expandafter\XINTfstop\expandafter.\expandafter{\xint_gobble_i#1}#2)\relax
}%
```

32.26.8. rseq()

When func_rseq has its turn, initial segment has been scanned by oparen, the ; mimicking the rôle of a closing parenthesis, and stopping further expansion (and leaving a \xint_c_i left-over token since 1.4). The ; is discovered during standard parsing mode, it may be for example {;} or arise from expansion as rseq does not use a delimited macro to locate it.

```
\def\XINT_expr_func_rseq   {\XINT_allexpr_rseq \xintbareeval      }%
\def\XINT_flexpr_func_rseq {\XINT_allexpr_rseq \xintbarefloateval }%
\def\XINT_iiexpr_func_rseq {\XINT_allexpr_rseq \xintbareiieval    }%
\def\XINT_allexpr_rseq #1#2#3#4%
{%
    \expandafter\XINT_expr_rseqx
    \expandafter #1\expanded{\xint_noxpd{{#4}}\expandafter}%
    \romannumeral`&&@\XINT_expr_fetch_E_comma_V_equal_E_a {}%
}%
\def\XINT_expr_rseqx #1#2#3#4%
{%
    \XINT:NEhook:rseq \XINT_expr_rseqy\romannumeral0#1(#4)\relax {#2}#3#1%
}%
\def\XINT_expr_rseqy #1#2#3#4#5%
{%
    \expandafter\XINT_expr_put_op_first
    \expanded \bgroup {\iffalse}\fi
    #2%
    \XINT_expr_rseq:_b {#5#4\relax !#3}#1^~{#2}\XINT_expr_cb_and_getop
}%
\def\XINT_expr_rseq:_b #1#2%
{%
    \ifx +#2\xint_dothis\XINT_expr_rseq:_Ca\fi
    \ifx !#2!\xint_dothis\XINT_expr_rseq:_noop\fi
    \ifx ^#2\xint_dothis\XINT_expr_rseq:_end\fi
    \xint_orthat{\XINT_expr_rseq:_c}{#2}{#1}%
}%
\def\XINT_expr_rseq:_noop #1{\XINT_expr_rseq:_b }%
\def\XINT_expr_rseq:_end  #1#2~#3{\iffalse{\fi}}%
\def\XINT_expr_rseq:_c #1#2%
   {\expandafter\XINT_expr_rseq:_d\romannumeral0#2{{#1}}{#2}}%
\def\XINT_expr_rseq:_d #1{\ifx ^#1\xint_dothis\XINT_expr_rseq:_abort\fi
                          \ifx ?#1\xint_dothis\XINT_expr_rseq:_break\fi
                          \ifx !#1\xint_dothis\XINT_expr_rseq:_omit\fi
                          \xint_orthat{\XINT_expr_rseq:_goon {#1}}}%
\def\XINT_expr_rseq:_abort #1!#2^~#3{\iffalse{\fi}}%
\def\XINT_expr_rseq:_break #1!#2^~#3{#1\iffalse{\fi}}%
\def\XINT_expr_rseq:_omit  #1!#2#{%
  \expandafter\XINT_expr_rseq:_b\xint_gobble_i
}%
\def\XINT_expr_rseq:_goon  #1!#2#{\XINT_expr_rseq:_goon_a {#1}}%
\def\XINT_expr_rseq:_goon_a #1#2#3~#4{#1\XINT_expr_rseq:_b #3~{#1}}%
\def\XINT_expr_rseq:_Ca #1#2#3{\XINT_expr_rseq:_Cc#3.{#2}}%
\def\XINT_expr_rseq:_Cb #1%
  {\expandafter\XINT_expr_rseq:_Cc\the\numexpr#1+\xint_c_i.}%
\def\XINT_expr_rseq:_Cc #1.#2%
  {\expandafter\XINT_expr_rseq:_D\romannumeral0#2{{#1[0]}}{#1}{#2}}%
```

```
\def\XINT_expr_rseq:_D #1{\ifx ^#1\xint_dothis\XINT_expr_rseq:_abort\fi
                          \ifx ?#1\xint_dothis\XINT_expr_rseq:_break\fi
                          \ifx !#1\xint_dothis\XINT_expr_rseq:_Omit\fi
                          \xint_orthat{\XINT_expr_rseq:_Goon {#1}}}%
\def\XINT_expr_rseq:_Omit  #1!#2#{%
  \expandafter\XINT_expr_rseq:_Cb\xint_gobble_i
}%
\def\XINT_expr_rseq:_Goon  #1!#2#{\XINT_expr_rseq:_Goon_a {#1}}%
\def\XINT_expr_rseq:_Goon_a #1#2#3~#4{#1\XINT_expr_rseq:_Cb #3~{#1}}%
```

32.26.9. iterr()

ATTENTION! at 1.4 the @ and @1 are not synonymous anymore. One *must* use @1 in iterr() context.

```
\def\XINT_expr_func_iterr   {\XINT_allexpr_iterr \xintbareeval      }%
\def\XINT_flexpr_func_iterr {\XINT_allexpr_iterr \xintbarefloateval }%
\def\XINT_iiexpr_func_iterr {\XINT_allexpr_iterr \xintbareiieval    }%
\def\XINT_allexpr_iterr #1#2#3#4%
{%
    \expandafter\XINT_expr_iterrx
    \expandafter #1\expanded{{\xintRevWithBraces{#4}}\expandafter}%
    \romannumeral`&&@\XINT_expr_fetch_E_comma_V_equal_E_a {}%
}%
\def\XINT_expr_iterrx #1#2#3#4%
{%
    \XINT:NEhook:iterr\XINT_expr_iterry\romannumeral0#1(#4)\relax {#2}#3#1%
}%
\def\XINT_expr_iterry #1#2#3#4#5%
{%
    \expandafter\XINT_expr_put_op_first
    \expanded \bgroup {\iffalse}\fi
    \XINT_expr_iterr:_b {#5#4\relax !#3}#1^~#20?\XINT_expr_cb_and_getop
}%
\def\XINT_expr_iterr:_b #1#2%
{%
    \ifx +#2\xint_dothis\XINT_expr_iterr:_Ca\fi
    \ifx !#2!\xint_dothis\XINT_expr_iterr:_noop\fi
    \ifx ^#2\xint_dothis\XINT_expr_iterr:_end\fi
    \xint_orthat{\XINT_expr_iterr:_c}{#2}{#1}%
}%
\def\XINT_expr_iterr:_noop #1{\XINT_expr_iterr:_b }%
\def\XINT_expr_iterr:_end  #1#2~#3#4?{{#3}\iffalse{\fi}}%
\def\XINT_expr_iterr:_c #1#2%
  {\expandafter\XINT_expr_iterr:_d\romannumeral0#2{{#1}}{#2}}%
\def\XINT_expr_iterr:_d #1{\ifx ^#1\xint_dothis\XINT_expr_iterr:_abort\fi
                           \ifx ?#1\xint_dothis\XINT_expr_iterr:_break\fi
                           \ifx !#1\xint_dothis\XINT_expr_iterr:_omit\fi
                           \xint_orthat{\XINT_expr_iterr:_goon {#1}}}%
\def\XINT_expr_iterr:_abort #1!#2^~#3?{\iffalse{\fi}}%
\def\XINT_expr_iterr:_break #1!#2^~#3?{#1\iffalse{\fi}}%
\def\XINT_expr_iterr:_omit  #1!#2#{%
  \expandafter\XINT_expr_iterr:_b\xint_gobble_i
}%
```

```
\def\XINT_expr_iterr:_goon  #1!#2#{\XINT_expr_iterr:_goon_a{#1}}%
\def\XINT_expr_iterr:_goon_a  #1#2#3~#4?%
{%
    \expandafter\XINT_expr_iterr:_b
    \expanded{\xint_noxpd{#3~}\xintTrim{-2}{#1#4}}0?%
}%
\def\XINT_expr_iterr:_Ca #1#2#3{\XINT_expr_iterr:_Cc#3.{#2}}%
\def\XINT_expr_iterr:_Cb #1%
  {\expandafter\XINT_expr_iterr:_Cc\the\numexpr#1+\xint_c_i.}%
\def\XINT_expr_iterr:_Cc #1.#2%
   {\expandafter\XINT_expr_iterr:_D\romannumeral0#2{{#1[0]}}{#1}{#2}}%
\def\XINT_expr_iterr:_D #1{\ifx ^#1\xint_dothis\XINT_expr_iterr:_abort\fi
                           \ifx ?#1\xint_dothis\XINT_expr_iterr:_break\fi
                           \ifx !#1\xint_dothis\XINT_expr_iterr:_Omit\fi
                           \xint_orthat{\XINT_expr_iterr:_Goon {#1}}}%
\def\XINT_expr_iterr:_Omit #1!#2#{%
  \expandafter\XINT_expr_iterr:_Cb\xint_gobble_i
}%
\def\XINT_expr_iterr:_Goon  #1!#2#{\XINT_expr_iterr:_Goon_a{#1}}%
\def\XINT_expr_iterr:_Goon_a  #1#2#3~#4?%
{%
    \expandafter\XINT_expr_iterr:_Cb
    \expanded{\xint_noxpd{#3~}\xintTrim{-2}{#1#4}}0?%
}%
```

32.26.10. rrseq()

When func_rrseq has its turn, initial segment has been scanned by oparen, the ; mimicking the rôle of a closing parenthesis, and stopping further expansion. #2 = \xint_c_i and #3 are left-over trash.

```
\def\XINT_expr_func_rrseq   {\XINT_allexpr_rrseq \xintbareeval      }%
\def\XINT_flexpr_func_rrseq {\XINT_allexpr_rrseq \xintbarefloateval }%
\def\XINT_iiexpr_func_rrseq {\XINT_allexpr_rrseq \xintbareiieval    }%
\def\XINT_allexpr_rrseq #1#2#3#4%
{%
    \expandafter\XINT_expr_rrseqx\expandafter#1\expanded
       {\xint_noxpd{{#4}}{\xintRevWithBraces{#4}}\expandafter}%
    \romannumeral`&&@\XINT_expr_fetch_E_comma_V_equal_E_a {}%
}%
\def\XINT_expr_rrseqx #1#2#3#4#5%
{%
    \XINT:NEhook:rrseq\XINT_expr_rrseqy\romannumeral0#1(#5)\relax {#2}{#3}#4#1%
}%
\def\XINT_expr_rrseqy #1#2#3#4#5#6%
{%
    \expandafter\XINT_expr_put_op_first
    \expanded \bgroup {\iffalse}\fi
    #2\XINT_expr_rrseq:_b {#6#5\relax !#4}#1^~#30?\XINT_expr_cb_and_getop
}%
\def\XINT_expr_rrseq:_b #1#2%
{%
    \ifx +#2\xint_dothis\XINT_expr_rrseq:_Ca\fi
```

```
    \ifx !#2!\xint_dothis\XINT_expr_rrseq:_noop\fi
    \ifx ^#2\xint_dothis\XINT_expr_rrseq:_end\fi
    \xint_orthat{\XINT_expr_rrseq:_c}{#2}{#1}%
}%
\def\XINT_expr_rrseq:_noop #1{\XINT_expr_rrseq:_b }%
\def\XINT_expr_rrseq:_end  #1#2~#3?{\iffalse{\fi}}%
\def\XINT_expr_rrseq:_c #1#2%
  {\expandafter\XINT_expr_rrseq:_d\romannumeral0#2{{#1}}{#2}}%
\def\XINT_expr_rrseq:_d #1{\ifx ^#1\xint_dothis\XINT_expr_rrseq:_abort\fi
                           \ifx ?#1\xint_dothis\XINT_expr_rrseq:_break\fi
                           \ifx !#1\xint_dothis\XINT_expr_rrseq:_omit\fi
                           \xint_orthat{\XINT_expr_rrseq:_goon {#1}}}%
\def\XINT_expr_rrseq:_abort #1!#2^~#3?{\iffalse{\fi}}%
\def\XINT_expr_rrseq:_break #1!#2^~#3?{#1\iffalse{\fi}}%
\def\XINT_expr_rrseq:_omit  #1!#2#{%
  \expandafter\XINT_expr_rrseq:_b\xint_gobble_i
}%
\def\XINT_expr_rrseq:_goon  #1!#2#{\XINT_expr_rrseq:_goon_a {#1}}%
\def\XINT_expr_rrseq:_goon_a  #1#2#3~#4?%
{%
    #1\expandafter\XINT_expr_rrseq:_b
    \expanded{\xint_noxpd{#3~}\xintTrim{-2}{#1#4}}0?%
}%
\def\XINT_expr_rrseq:_Ca #1#2#3{\XINT_expr_rrseq:_Cc#3.{#2}}%
\def\XINT_expr_rrseq:_Cb #1%
  {\expandafter\XINT_expr_rrseq:_Cc\the\numexpr#1+\xint_c_i.}%
\def\XINT_expr_rrseq:_Cc #1.#2%
   {\expandafter\XINT_expr_rrseq:_D\romannumeral0#2{{#1[0]}}{#1}{#2}}%
\def\XINT_expr_rrseq:_D #1{\ifx ^#1\xint_dothis\XINT_expr_rrseq:_abort\fi
                           \ifx ?#1\xint_dothis\XINT_expr_rrseq:_break\fi
                           \ifx !#1\xint_dothis\XINT_expr_rrseq:_Omit\fi
                           \xint_orthat{\XINT_expr_rrseq:_Goon {#1}}}%
\def\XINT_expr_rrseq:_Omit  #1!#2#{%
  \expandafter\XINT_expr_rrseq:_Cb\xint_gobble_i
}%
\def\XINT_expr_rrseq:_Goon  #1!#2#{\XINT_expr_rrseq:_Goon_a {#1}}%
\def\XINT_expr_rrseq:_Goon_a  #1#2#3~#4?%
{%
    #1\expandafter\XINT_expr_rrseq:_Cb
      \expanded{\xint_noxpd{#3~}\xintTrim{-2}{#1#4}}0?%
}%
\catcode`? 11
```

32.27. Pseudo-functions related to N-dimensional hypercubic lists

32.27.1. ndseq()

New with 1.4. 2020/01/23. It is derived from subsm() but instead of evaluating one expression according to one value per variable, it constructs a nested bracketed seq... this means the expression is parsed each time ! Anyway, proof of concept. Nota Bene : omit, abort, break() work !

```
\def\XINT_expr_onliteral_ndseq
{%
```

```
    \expandafter\XINT_allexpr_ndseq_f
    \romannumeral`&&@\XINT_expr_fetch_E_comma_V_equal_E_a {}%
}%
\def\XINT_allexpr_ndseq_f #1#2{\xint_c_ii^v `{ndseqx}#2)\relax #1}%
\def\XINT_expr_func_ndseqx
{%
    \expandafter\XINT_allexpr_ndseqx\expandafter\xintbareeval
    \expandafter{\romannumeral0\expandafter\xint_gobble_i\string}%
    \expandafter\xintrevwithbraces
    \expanded\bgroup{\iffalse}\fi\XINT_allexpr_ndseq_A\XINT_expr_oparen
}%
\def\XINT_flexpr_func_ndseqx
{%
    \expandafter\XINT_allexpr_ndseqx\expandafter\xintbarefloateval
    \expandafter{\romannumeral0\expandafter\xint_gobble_i\string}%
    \expandafter\xintrevwithbraces
    \expanded\bgroup{\iffalse}\fi\XINT_allexpr_ndseq_A\XINT_flexpr_oparen
}%
\def\XINT_iiexpr_func_ndseqx
{%
    \expandafter\XINT_allexpr_ndseqx\expandafter\xintbareiieval
    \expandafter{\romannumeral0\expandafter\xint_gobble_i\string}%
    \expandafter\xintrevwithbraces
    \expanded\bgroup{\iffalse}\fi\XINT_allexpr_ndseq_A\XINT_iiexpr_oparen
}%
\def\XINT_allexpr_ndseq_A #1#2#3%
{%
    \ifx#2\xint_c_
       \expandafter\XINT_allexpr_ndseq_C
    \else
       \expandafter\XINT_allexpr_ndseq_B
    \fi #1%
}%
\def\XINT_allexpr_ndseq_B #1#2#3#4=%
{%
    {#2}{\xint_zapspaces#3#4 \xint_gobble_i}%
    \expandafter\XINT_allexpr_ndseq_A\expandafter#1\romannumeral`&&@#1%
}%
```

#1 = \xintbareeval, or \xintbarefloateval or \xintbareiieval #2 = values for last coordinate

```
\def\XINT_allexpr_ndseq_C #1#2{{#2}\iffalse{{{\fi}}}}%
```

#1 = \xintbareeval or \xintbarefloateval or \xintbareiieval #2 = {valuesN}...{values2}{var2}{values1} #3 = {var1} #4 = the expression to evaluate

```
\def\XINT_allexpr_ndseqx #1#2#3#4%
{%
    \expandafter\XINT_expr_put_op_first
    \expanded
    \bgroup
      \romannumeral0#1\empty
      \expanded{\xintReplicate{\xintLength{{#3}#2}/2}{[seq(}%
                \xint_noxpd{#4}%
                \XINT_allexpr_ndseqx_a #2{#3}^^%
```

```
                }%
      \relax
    \iffalse{\fi\expandafter}\romannumeral`&&@\XINT_expr_getop
}%
\def\XINT_allexpr_ndseqx_a #1#2%
{%
    \xint_gob_til_^ #1\XINT_allexpr_ndseqx_e ^%
    \xint_noxpd{,#2=\XINTfstop.{#1})]}\XINT_allexpr_ndseqx_a
}%
\def\XINT_allexpr_ndseqx_e ^#1\XINT_allexpr_ndseqx_a{}%
```

32.27.2. ndmap()

New with 1.4. 2020/01/24.

```
\def\XINT_expr_onliteral_ndmap #1,{\xint_c_ii^v `{ndmapx}\XINTfstop.{#1};}%
\def\XINT_expr_func_ndmapx #1#2#3%
{%
    \expandafter\XINT_allexpr_ndmapx
    \csname XINT_expr_func_\xint_zapspaces #3 \xint_gobble_i\endcsname
    \XINT_expr_oparen
}%
\def\XINT_flexpr_func_ndmapx #1#2#3%
{%
    \expandafter\XINT_allexpr_ndmapx
    \csname XINT_flexpr_func_\xint_zapspaces #3 \xint_gobble_i\endcsname
    \XINT_flexpr_oparen
}%
\def\XINT_iiexpr_func_ndmapx #1#2#3%
{%
    \expandafter\XINT_allexpr_ndmapx
    \csname XINT_iiexpr_func_\xint_zapspaces #3 \xint_gobble_i\endcsname
    \XINT_iiexpr_oparen
}%
\def\XINT_allexpr_ndmapx #1#2%
{%
    \expandafter\XINT_expr_put_op_first
    \expanded\bgroup{\iffalse}\fi
    \expanded
      {\noexpand\XINT:NEhook:x:ndmapx
       \noexpand\XINT_allexpr_ndmapx_a
       \noexpand#1{}\expandafter}%
    \expanded\bgroup\expandafter\XINT_allexpr_ndmap_A
                    \expandafter#2\romannumeral`&&@#2%
}%
\def\XINT_allexpr_ndmap_A #1#2#3%
{%
    \ifx#3;%
       \expandafter\XINT_allexpr_ndmap_B
    \else
       \xint_afterfi{\XINT_allexpr_ndmap_C#2#3}%
    \fi #1%
}%
```

```
\def\XINT_allexpr_ndmap_B #1#2%
{%
    {#2}\expandafter\XINT_allexpr_ndmap_A\expandafter#1\romannumeral`&&@#1%
}%
\def\XINT_allexpr_ndmap_C #1#2#3#4%
{%
    {#4}^\relax\iffalse{{{\fi}}}#1#2%
}%
\def\XINT_allexpr_ndmapx_a #1#2#3%
{%
    \xint_gob_til_^ #3\XINT_allexpr_ndmapx_l ^%
    \XINT_allexpr_ndmapx_b #1{#2}{#3}%
}%
\def\XINT_allexpr_ndmapx_l ^#1\XINT_allexpr_ndmapx_b #2#3#4\relax
{%
    #2\empty\xint_firstofone{#3}%
}%
\def\XINT_allexpr_ndmapx_b #1#2#3#4\relax
{%
    {\iffalse}\fi\XINT_allexpr_ndmapx_c {#4\relax}#1{#2}#3^%
}%
\def\XINT_allexpr_ndmapx_c #1#2#3#4%
{%
    \xint_gob_til_^ #4\XINT_allexpr_ndmapx_e ^%
    \XINT_allexpr_ndmapx_a #2{#3{#4}}#1%
    \XINT_allexpr_ndmapx_c  {#1}#2{#3}%
}%
\def\XINT_allexpr_ndmapx_e ^#1\XINT_allexpr_ndmapx_c
   {\iffalse{\fi}\xint_gobble_iii}%
```

32.27.3. ndfillraw()

New with 1.4. 2020/01/24. J'hésite à autoriser un #1 quelconque, ou plutôt à le wrapper dans un \xintbareval. Mais il faut alors distinguer les trois. De toute façon les variables ne marcheraient pas donc j'hésite à mettre un wrapper automatique. Mais ce n'est pas bien d'autoriser l'injection de choses quelconques.

Pour des choses comme ndfillraw(\xintRandomBit,[10,10]).

Je n'aime pas le nom !. Le changer. ndconst? Surtout je n'aime pas que dans le premier argument il faut rajouter explicitement si nécessaire \xintiiexpr wrap.

```
\def\XINT_expr_onliteral_ndfillraw #1,%
  {\xint_c_ii^v `{ndfillrawx}\XINTfstop.{{#1}},}%
\def\XINT_expr_func_ndfillrawx #1#2#3%
{%
    \expandafter#1\expandafter#2%
    \expanded{{{\XINT_allexpr_ndfillrawx_a #3}}}%
}%
\let\XINT_iiexpr_func_ndfillrawx\XINT_expr_func_ndfillrawx
\let\XINT_flexpr_func_ndfillrawx\XINT_expr_func_ndfillrawx
\def\XINT_allexpr_ndfillrawx_a #1#2%
{%
    \expandafter\XINT_allexpr_ndfillrawx_b
    \romannumeral0\xintApply{\calxListItemToNum}{#2}^\relax {#1}%
```

```
}%
\def\XINT_allexpr_ndfillrawx_b #1#2\relax#3%
{%
    \xint_gob_til_^ #1\XINT_allexpr_ndfillrawx_c ^%
    \xintReplicate{#1}{{\XINT_allexpr_ndfillrawx_b #2\relax {#3}}}%
}%
\def\XINT_allexpr_ndfillrawx_c ^\xintReplicate #1#2%
{%
    \expandafter\XINT_allexpr_ndfillrawx_d\xint_firstofone #2%
}%
\def\XINT_allexpr_ndfillrawx_d\XINT_allexpr_ndfillrawx_b \relax #1{#1}%
```

32.27.4. Making them known to the other parsers

Modified at 2.0a1 (2026/11/01). The "onliteral" handlers are now per-parser for reasons unrelated to the above constructs. Very poorly documented, see next section.

```
\xintFor* #1 in
{{subs}{subsm}{subsn}{seq}{add}{mul}{ndseq}{ndmap}{ndfillraw}}\do{%
    \expandafter\let\csname XINT_flexpr_onliteral_#1\expandafter\endcsname
                    \csname XINT_expr_onliteral_#1\endcsname
    \expandafter\let\csname XINT_iiexpr_onliteral_#1\expandafter\endcsname
                    \csname XINT_expr_onliteral_#1\endcsname
}%
```

32.28. Other pseudo-functions: bool(), togl(), protect(), qraw(), qint(), qfrac(), qfloat(), qrand(), random(), rbit()

bool, togl and protect use delimited macros. They are not true functions, they turn off the parser to gather their "variable".

Modified at 1.2 (2015/10/10). Adds qint(), qfrac(), qfloat().

Modified at 1.3c (2018/06/17). Adds qraw(). Useful to limit impact on TEX memory from abuse of \csname's storage when generating many comma separated values from a loop.

Modified at 1.3e (2019/04/05). qfloat() keeps a short mantissa if possible.

They allow the user to hand over quickly a big number to the parser, spaces not immediately removed but should be harmless in general. The qraw() does no post-processing at all apart complete expansion, useful for comma-separated values, but must be obedient to (non really documented) expected format. Each uses a delimited macro, the closing parenthesis can not emerge from expansion.

1.3b. random(), qrand() Function-like syntax but with no argument currently, so let's use fast parsing which requires though the closing parenthesis to be explicit.

Attention that qraw() which pre-supposes knowledge of internal storage model is fragile and may break at any release.

1.4 adds rbit(). Short for random bit.

Modified at 2.0a1 (2026/11/01). Now needs \let to make "onliteral" handlers known to other parsers.

```
\def\XINT_expr_onliteral_bool #1)%
    {\expandafter\XINT_expr_put_op_first
     \expanded{{{\xintBool{#1}[0]}}\expandafter
     }\romannumeral`&&@\XINT_expr_getop}%
\def\XINT_expr_onliteral_togl #1)%
    {\expandafter\XINT_expr_put_op_first
```

```
      \expanded{{{\xintToggle{#1}[0]}}\expandafter
      }\romannumeral`&&@\XINT_expr_getop}%
\def\XINT_expr_onliteral_protect #1)%
    {\expandafter\XINT_expr_put_op_first
      \expanded{{{\detokenize{#1}}}\expandafter
      }\romannumeral`&&@\XINT_expr_getop}%
\def\XINT_expr_onliteral_qint #1)%
    {\expandafter\XINT_expr_put_op_first
      \expanded{{{\xintiNum{#1}[0]}}\expandafter
      }\romannumeral`&&@\XINT_expr_getop}%
\xintFor* #1 in {{bool}{togl}{protect}{qint}}\do{%
    \expandafter\let\csname XINT_flexpr_onliteral_#1\expandafter\endcsname
                    \csname XINT_expr_onliteral_#1\endcsname
    \expandafter\let\csname XINT_iiexpr_onliteral_#1\expandafter\endcsname
                    \csname XINT_expr_onliteral_#1\endcsname
}%
```

Modified at 2.0a1 (2026/11/01).

BREAKING CHANGE. The qfloat() interface is REMOVED from \xintiieval. Only kept in \xintfloateval and \xinteval (with xintexpr.sty).

qraw() should never have been mentioned in user manual. It is a use-at-own-risk. Depends on knowing internals.

THIS AI COMMENT IS PARTLY OBSOLETE. IT LOCATED \xintFor LOOPS AT WRONG PLACES REGARDING CODE LEGIBILITY. USER MANUAL INTERVENTION MODIFIED LAYOUT. BESIDES THE \xintFor LOOPS THEMSELVES USED AN INTERMEDIATE MACRO \XINT_tmpa STILL REDUCING LEGIBILIT. THE AI HAS MESSED UP THE INTERNAL LOGIC.

calx: the onliterals are looked up per parser (\XINT_<p>_onliteral_<name>): the \xinteval ones are copied for \xintfloateval and \xintiieval. qint() of \xintiieval gives an M value, qfloat() of \xintfloateval an F value (as float()), random() and qrand() of \xintfloateval F values (calxfp's \calxFpRandomSdigits, \calxFpRandomSixteen). The argument of qfloat() of \xintfloateval is read by \CALX_fround: a raw value A, A[N], A/B, A/B[N] (any xintfrac input with xintexpr.sty). qfrac(A/B) (the slash is mandatory; A and B f-expand to integers in strict format: 0 or an optional - then digits without leading zero): \xintfloateval: A/B rounded to P digits (\CALX_fqfrac: exact F values of A and B by calxfp's \CALX_ffromraw, then its correctly rounded division \CALX_fr_xk; B may be negative; B = 0 signals as x / 0); \xintiieval: A/B[0] (B > 0), for round(), trunc(), num() etc. The \xinteval qfrac() (\xintRaw, any xintfrac input) is xintexpr.sty's; loading it does not change the two calx ones.

```
\def\XINT_expr_onliteral_qraw #1)%
    {\expandafter\XINT_expr_put_op_first\expanded{{#1}\expandafter
      }\romannumeral`&&@\XINT_expr_getop}%
\let\XINT_flexpr_onliteral_qraw\XINT_expr_onliteral_qraw
\let\XINT_iiexpr_onliteral_qraw\XINT_expr_onliteral_qraw
\def\XINT_expr_onliteral_qfloat #1)%
   {\expandafter\XINT_expr_put_op_first
    \expanded{{{\calxFpFloatdigits{#1}}}\expandafter
     }\romannumeral`&&@\XINT_expr_getop}%
\let\XINT_flexpr_onliteral_qfloat\XINT_expr_onliteral_qfloat
\def\XINT_expr_onliteral_random #1)%
    {\expandafter\XINT_expr_put_op_first
      \expanded{{{\XINTinRandomFloatSdigits}}\expandafter
      }\romannumeral`&&@\XINT_expr_getop}%
\let\XINT_iiexpr_onliteral_random\XINT_expr_onliteral_random
\def\XINT_flexpr_onliteral_random #1)%
   {\expandafter\XINT_expr_put_op_first
```

```
    \expanded{{{\calxFpRandomSdigits}}\expandafter
     }\romannumeral`&&@\XINT_expr_getop}%
\def\XINT_expr_onliteral_qrand #1)%
   {\expandafter\XINT_expr_put_op_first
    \expanded{{{\XINTinRandomFloatSixteen}}\expandafter
     }\romannumeral`&&@\XINT_expr_getop}%
\let\XINT_iiexpr_onliteral_qrand\XINT_expr_onliteral_qrand
\def\XINT_flexpr_onliteral_qrand #1)%
   {\expandafter\XINT_expr_put_op_first
    \expanded{{{\calxFpRandomSixteen}}\expandafter
     }\romannumeral`&&@\XINT_expr_getop}%
\def\XINT_expr_onliteral_rbit #1)%
   {\expandafter\XINT_expr_put_op_first
    \expanded{{{\xintRandBit[0]}}\expandafter
     }\romannumeral`&&@\XINT_expr_getop}%
\let\XINT_flexpr_onliteral_rbit\XINT_expr_onliteral_rbit
\let\XINT_iiexpr_onliteral_rbit\XINT_expr_onliteral_rbit
\def\XINT_iiexpr_onliteral_qint #1)%
   {\expandafter\XINT_expr_put_op_first
    \expanded{{{\romannumeral0\CALX_mfromraw{\xintiNum{#1}}}}}\expandafter
     }\romannumeral`&&@\XINT_expr_getop}%
\def\XINT_expr_onliteral_qfrac #1/#2)%
   {\expandafter\XINT_expr_put_op_first
    \expanded{{{#1/#2[0]}}\expandafter
     }\romannumeral`&&@\XINT_expr_getop}%
\let\XINT_iiexpr_onliteral_qfrac\XINT_expr_onliteral_qfrac
\def\XINT_flexpr_onliteral_qfrac #1/#2)%
   {\expandafter\XINT_expr_put_op_first
    \expanded{{{\CALX_fqfrac{#1}{#2}}}\expandafter
     }\romannumeral`&&@\XINT_expr_getop}%
\def\CALX_fqfrac #1#2%
   {\romannumeral0\expandafter\CALX_fr_xk\expanded
    {\romannumeral0\CALX_ffromraw{#1[0]}\xint:%
     \romannumeral0\CALX_ffromraw{#2[0]}\xint:{\the\numexpr\XINTdigits}}}%
```

32.29. Regular built-in functions: num(), reduce(), preduce(), abs(), sgn(), frac(), floor(), ceil(), sqr(), ?(), !(), not(), odd(), even(), isint(), isone(), factorial(), sqrt(), sqrtr(), inv(), round(), trunc(), float(), sfloat(), ilog10(), divmod(), mod(), binomial(), pfactorial(), randrange(), iquo(), irem(), gcd(), lcm(), max(), min(), `+`(), `*`(), all(), any(), xor(), len(), first(), last(), reversed(), if(), ifint(), ifone(), ifsgn(), nuple(), unpack(), flat() and zip()

```
\def\XINT:expr:f:one:and:opt #1#2#3!#4#5%
{%
    \if\relax#3\relax\expandafter\xint_firstoftwo\else
                       \expandafter\xint_secondoftwo\fi
    {#4}{#5[\calxListItemToNum{#2}]}{#1}%
}%
\def\XINT:expr:f:tacitzeroifone #1#2#3!#4#5%
{%
    \if\relax#3\relax\expandafter\xint_firstoftwo\else
```

```
                        \expandafter\xint_secondoftwo\fi
    {#4{0}}{#5{\calxListItemToNum{#2}}}{#1}%
}%
\def\XINT:expr:f:iitacitzeroifone #1#2#3!#4%
{%
    \if\relax#3\relax\expandafter\xint_firstoftwo\else
                        \expandafter\xint_secondoftwo\fi
    {#4{0[0]}}{#4{#2}}{#1}%
}%
\def\XINT_iiexpr_func_num #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\calxListItemToNum#3}}%
}%
\def\XINT_flexpr_func_num #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\calxFpTrunc#3}}%
}%
\def\XINT_flexpr_func_abs #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\calxFpAbs#3}}%
}%
\def\XINT_iiexpr_func_abs #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\calxiiAbs#3}}%
}%
\def\XINT_iiexpr_func_sgn #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\calxSgn#3}}%
}%
\def\XINT_flexpr_func_sgn #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\calxFpSgn#3}}%
}%
\def\XINT_flexpr_func_frac #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\calxFpFrac#3}}%
}%
```

no \XINT_iiexpr_func_frac

```
\def\XINT_flexpr_func_floor #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\calxFpFloor#3}}%
}%
```

The floor and ceil functions in \xintiiexpr require protect(a/b) or, better, \qfrac(a/b); else the / will be executed first and do an integer rounded division.

```
\def\XINT_iiexpr_func_floor #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\calxiiFloor#3}}%
}%
\def\XINT_flexpr_func_ceil #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\calxFpCeil#3}}%
}%
\def\XINT_iiexpr_func_ceil #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\calxiiCeil#3}}%
}%
\def\XINT_flexpr_func_sqr #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\calxFpSqr#3}}%% calx
}%
\def\XINT_iiexpr_func_sqr #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\calxiiSqr#3}}%
}%
\def\XINT_iiexpr_func_? #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\calxIsNotZero#3}}%
}%
\def\XINT_flexpr_func_? #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\calxFpIsNotZero#3}}%
}%
\def\XINT_iiexpr_func_! #1#2#3%
{%
```

```
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\calxIsZero#3}}%
}%
\def\XINT_flexpr_func_! #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\calxFpIsZero#3}}%
}%
\def\XINT_iiexpr_func_not #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\calxIsZero#3}}%
}%
\def\XINT_flexpr_func_not #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\calxFpIsZero#3}}%
}%
\def\XINT_flexpr_func_odd #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\calxFpOdd#3}}%
}%
\def\XINT_iiexpr_func_odd #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\calxiiOdd#3}}%
}%
\def\XINT_flexpr_func_even #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\calxFpEven#3}}%
}%
\def\XINT_iiexpr_func_even #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\calxiiEven#3}}%
}%
\def\XINT_flexpr_func_isint #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\calxFpIsInt#3}}%
}%
```

```
\def\XINT_iiexpr_func_isint #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\calxiiIsInt#3}}%
}%
\def\XINT_flexpr_func_isone #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\calxFpIsOne#3}}%
}%
\def\XINT_iiexpr_func_isone #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\calxiiIsOne#3}}%
}%
\def\XINT_flexpr_func_factorial #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:and:opt:direct
    \XINT:expr:f:one:and:opt#3,!\calxFpFacdigits\calxFpFac
    }}%
}%
\def\XINT_iiexpr_func_factorial #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\calxiiFac#3}}%
}%
\def\XINT_flexpr_func_sqrt #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:and:opt:direct
    \XINT:expr:f:one:and:opt #3,!\calxFpSqrtdigits\calxFpSqrt% calx
    }}%
}%
\def\XINT_iiexpr_func_sqrt #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\calxiiSqrt#3}}%
}%
\def\XINT_iiexpr_func_sqrtr #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\calxiiSqrtR#3}}%
}%
\def\XINT_flexpr_func_inv #1#2#3%
{%
```

```
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\calxFpInv#3}}%
}%
\def\XINT_flexpr_func_round #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:tacitzeroifone:direct
    \XINT:expr:f:tacitzeroifone #3,!\calxFpIRound\calxFpRoundTo
    }}%
}%
\def\XINT_iiexpr_func_round #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:iitacitzeroifone:direct
    \XINT:expr:f:iitacitzeroifone #3,!\calxiiRound
    }}%
}%
\def\XINT_flexpr_func_trunc #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:tacitzeroifone:direct
    \XINT:expr:f:tacitzeroifone #3,!\calxFpITrunc\calxFpTruncTo
    }}%
}%
\def\XINT_iiexpr_func_trunc #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:iitacitzeroifone:direct
    \XINT:expr:f:iitacitzeroifone #3,!\calxiiTrunc
    }}%
}%
```

Hesitation at 1.3e about using \XINTinFloatSdigits and \XINTinFloatS. Finally I add a sfloat() function. It helps for xinttrig.sty.

At 2.0a1, there is no xinttrig.sty anymore. The sfloat() function and its support macros \XINTinFloatSdigits and \XINTinFloatS have no real rationale: the S variant goal was to produce smaller mantissas, but there is hardly any utility for this with the calx float model. So we let the sfloat() be like float() in \xintfloateval. Neither float() nor sfloat() were ever defined in \xintiieval. The \xinteval float() and sfloat() keep their legacy meanings and are defined in xintexpr.sty. Most of the comments that folllow below are either outdate or regard the \xinteval situation.

float_() was added at 1.4, as a shortcut alias to float() skipping the check for an optional second argument. This is useful to transfer function definitions between \xintexpr and \xintfloatexpr contexts.

No need for a similar shortcut for sfloat() as currently used in xinttrig.sty to go from float to expr: as it is used there as sfloat(x) with dummy x, it sees there is no optional argument, contrarily to for example float(\xintexpr...\relax) which has to allow for the inner expression to expand to an ople with two items, so does not know in which branch it is at time of definiion.

After some hesitation at 1.4e regarding guard digits mechanism the float_() got renamed to float_dgt(), but then renamed back to float_() to avoid a breaking change and having to document it.

Nevertheless the documentation of 1.4e mentioned float_dgt()... but it was still float_()...

now changed into float_dgt() for real at 1.4f.
  1.4f also adds private float_dgtormax and sfloat_dgtormax for matters of xinttrig.

```
\def\XINT_flexpr_func_float #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:and:opt:direct
    \XINT:expr:f:one:and:opt #3,!\calxFpFloatdigits\calxFpFloat
    }}%
}%
\def\XINT_flexpr_func_float_dgt #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\calxFpFloatdigits#3}}%
}%
\let\XINT_flexpr_func_sfloat\XINT_flexpr_func_float
\expandafter\def\csname XINT_flexpr_func_ilog10\endcsname #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:and:opt:direct
    \XINT:expr:f:one:and:opt #3,!\calxFpiLogTendigits\calxFpiLogTen
    }}%
}%
\expandafter\def\csname XINT_iiexpr_func_ilog10\endcsname #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\calxiiLogTen#3}}%
}%
\def\XINT_flexpr_func_divmod #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{\romannumeral`&&@%
    \XINT:NEhook:f:one:from:two
    {\romannumeral`&&@\calxFpDivMod #3}}%
}%
\def\XINT_iiexpr_func_divmod #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{\romannumeral`&&@%
    \XINT:NEhook:f:one:from:two
    {\romannumeral`&&@\calxiiDivMod #3}}%
}%
\def\XINT_flexpr_func_mod #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{\romannumeral`&&@%
    \XINT:NEhook:f:one:from:two
    {\romannumeral`&&@\calxFpMod#3}}%
}%
\def\XINT_iiexpr_func_mod #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{\romannumeral`&&@%
    \XINT:NEhook:f:one:from:two
    {\romannumeral`&&@\calxiiMod#3}}%
```

```
}%
\def\XINT_flexpr_func_binomial #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{\romannumeral`&&@%
    \XINT:NEhook:f:one:from:two
    {\romannumeral`&&@\calxFpBinomial #3}}%
}%
\def\XINT_iiexpr_func_binomial #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{\romannumeral`&&@%
    \XINT:NEhook:f:one:from:two
    {\romannumeral`&&@\calxiiBinomial #3}}%
}%
\def\XINT_flexpr_func_pfactorial #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{\romannumeral`&&@%
    \XINT:NEhook:f:one:from:two
    {\romannumeral`&&@\calxFpPFactorial #3}}%
}%
\def\XINT_iiexpr_func_pfactorial #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{\romannumeral`&&@%
    \XINT:NEhook:f:one:from:two
    {\romannumeral`&&@\calxiiPFactorial #3}}%
}%
\def\XINT_flexpr_func_randrange #1#2#3%
{%
    \expandafter #1\expandafter #2\expanded{{{%
    \XINT:expr:randrange #3,!%
    }}}%
}%
\def\XINT:expr:randrange #1#2#3!%
{%
    \if\relax#3\relax\expandafter\xint_firstoftwo\else
                      \expandafter\xint_secondoftwo\fi
    {\calxintRandRange{\XINT:NEhook:f:one:from:one:direct\calxFpToM{#1}}}%
    {\calxintRandRangeAtoB
                       {\XINT:NEhook:f:one:from:one:direct\calxFpToM{#1}}%
                       {\XINT:NEhook:f:one:from:one:direct\calxFpToM{#2}}%
     }%
}%
\def\XINT_iiexpr_func_randrange #1#2#3%
{%
    \expandafter #1\expandafter #2\expanded{{{%
    \XINT:iiexpr:randrange #3,!%
    }}}%
}%
\def\XINT:iiexpr:randrange #1#2#3!%
{%
    \if\relax#3\relax\expandafter\xint_firstoftwo\else
                      \expandafter\xint_secondoftwo\fi
    {\calxintRandRange{#1}}%
```

```
3507     {\calxintRandRangeAtoB{#1}{#2}}%
3508 }%
3509 \def\XINT_iiexpr_func_iquo #1#2#3%
3510 {%
3511     \expandafter #1\expandafter #2\expandafter{\romannumeral`&&@%
3512     \XINT:NEhook:f:one:from:two
3513     {\romannumeral`&&@\calxiiQuo #3}}%
3514 }%
3515 \def\XINT_iiexpr_func_irem #1#2#3%
3516 {%
3517     \expandafter #1\expandafter #2\expandafter{\romannumeral`&&@%
3518     \XINT:NEhook:f:one:from:two
3519     {\romannumeral`&&@\calxiiRem #3}}%
3520 }%
3521 \def\XINT_iiexpr_func_gcd #1#2#3%
3522 {%
3523     \expandafter #1\expandafter #2\expandafter{\expandafter
3524     {\romannumeral`&&@\XINT:NEhook:f:from:delim:u\CALX_iiGCDof#3^}}%
3525 }%
3526 \def\XINT_iiexpr_func_lcm #1#2#3%
3527 {%
3528     \expandafter #1\expandafter #2\expandafter{\expandafter
3529     {\romannumeral`&&@\XINT:NEhook:f:from:delim:u\CALX_iiLCMof#3^}}%
3530 }%
3531 \def\XINT_iiexpr_func_max #1#2#3%
3532 {%
3533     \expandafter #1\expandafter #2\expandafter{\expandafter
3534     {\romannumeral`&&@\XINT:NEhook:f:from:delim:u\CALX_iiMaxof#3^}}%
3535 }%
3536 \def\XINT_flexpr_func_max #1#2#3%
3537 {%
3538     \expandafter #1\expandafter #2\expandafter{\expandafter
3539     {\romannumeral`&&@\XINT:NEhook:f:from:delim:u\calxFpMaxof#3^}}% calx
3540 }%
3541 \def\XINT_iiexpr_func_min #1#2#3%
3542 {%
3543     \expandafter #1\expandafter #2\expandafter{\expandafter
3544     {\romannumeral`&&@\XINT:NEhook:f:from:delim:u\CALX_iiMinof#3^}}%
3545 }%
3546 \def\XINT_flexpr_func_min #1#2#3%
3547 {%
3548     \expandafter #1\expandafter #2\expandafter{\expandafter
3549     {\romannumeral`&&@\XINT:NEhook:f:from:delim:u\calxFpMinof#3^}}% calx
3550 }%
3551 \expandafter
3552 \def\csname XINT_flexpr_func_+\endcsname #1#2#3%
3553 {%
3554     \expandafter #1\expandafter #2\expandafter{\expandafter
3555     {\romannumeral`&&@\XINT:NEhook:f:from:delim:u\calxFpSum#3^}}%
3556 }%
3557 \expandafter
3558 \def\csname XINT_iiexpr_func_+\endcsname #1#2#3%
```

```
{%
    \expandafter #1\expandafter #2\expandafter{\expandafter
    {\romannumeral`&&@\XINT:NEhook:f:from:delim:u\CALX_iiSum#3^}}%
}%
\expandafter
\def\csname XINT_flexpr_func_*\endcsname #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{\expandafter
    {\romannumeral`&&@\XINT:NEhook:f:from:delim:u\calxFpPrd#3^}}%
}%
\expandafter
\def\csname XINT_iiexpr_func_*\endcsname #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{\expandafter
    {\romannumeral`&&@\XINT:NEhook:f:from:delim:u\CALX_iiPrd#3^}}%
}%
\def\XINT_iiexpr_func_all #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{\expandafter
    {\romannumeral`&&@\XINT:NEhook:f:from:delim:u\CALX_ANDof#3^}}%
}%
\def\XINT_flexpr_func_all #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{\expandafter
    {\romannumeral`&&@\XINT:NEhook:f:from:delim:u\calxFpANDof#3^}}%
}%
\def\XINT_iiexpr_func_any #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{\expandafter
    {\romannumeral`&&@\XINT:NEhook:f:from:delim:u\CALX_ORof#3^}}%
}%
\def\XINT_flexpr_func_any #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{\expandafter
    {\romannumeral`&&@\XINT:NEhook:f:from:delim:u\calxFpORof#3^}}%
}%
\def\XINT_iiexpr_func_xor #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{\expandafter
    {\romannumeral`&&@\XINT:NEhook:f:from:delim:u\CALX_XORof#3^}}%
}%
\def\XINT_flexpr_func_xor #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{\expandafter
    {\romannumeral`&&@\XINT:NEhook:f:from:delim:u\calxFpXORof#3^}}%
}%
\def\XINT_iiexpr_func_len #1#2#3%
{%
    \expandafter#1\expandafter#2\expandafter{\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:LFL\xintLength
    {\romannumeral\XINT:NEhook:r:check#3^}%
    }}%
```

```
}%
\def\XINT_flexpr_func_len #1#2#3%
{%
    \expandafter#1\expandafter#2\expandafter{\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:LFL\calxFpLength
    {\romannumeral\XINT:NEhook:r:check#3^}%
    }}%
}%
\def\XINT_flexpr_func_first #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:LFL\xintFirstOne
    {\romannumeral\XINT:NEhook:r:check#3^}%
    }%
}%
\let\XINT_iiexpr_func_first\XINT_flexpr_func_first
\def\XINT_flexpr_func_last #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:LFL\xintLastOne
    {\romannumeral\XINT:NEhook:r:check#3^}%
    }%
}%
\let\XINT_iiexpr_func_last\XINT_flexpr_func_last
\def\XINT_flexpr_func_reversed #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:reverse\XINT_expr_reverse
    #3^^#3\xint:\xint:\xint:\xint:
          \xint:\xint:\xint:\xint:\xint_bye
    }%
}%
\def\XINT_expr_reverse #1#2%
{%
    \if ^\noexpand#2%
        \expandafter\XINT_expr_reverse:_one_or_none\string#1.%
    \else
        \expandafter\XINT_expr_reverse:_at_least_two
    \fi
}%
\def\XINT_expr_reverse:_at_least_two #1^^{\XINT_revwbr_loop {}}%
\def\XINT_expr_reverse:_one_or_none #1%
{%
    \if #1\bgroup\xint_dothis\XINT_expr_reverse:_nutple\fi
    \if #1^\xint_dothis\XINT_expr_reverse:_nil\fi
    \xint_orthat\XINT_expr_reverse:_leaf
}%
\edef\XINT_expr_reverse:_nil #1\xint_bye{\noexpand\fi\space}%
\def\XINT_expr_reverse:_leaf#1\fi #2\xint:#3\xint_bye
   {\fi\xint_gob_andstop_i#2}%
\def\XINT_expr_reverse:_nutple%
{%
```

```
3663     \expandafter\XINT_expr_reverse:_nutple_a\expandafter{\string}%
3664 }%
3665 \def\XINT_expr_reverse:_nutple_a #1^#2\xint:#3\xint_bye
3666 {%
3667     \fi\expandafter
3668     {\romannumeral0\XINT_revwbr_loop{}#2\xint:#3\xint_bye}%
3669 }%
3670 \let\XINT_iiexpr_func_reversed\XINT_flexpr_func_reversed
3671 \def\XINT_flexpr_func_if #1#2#3%
3672 {%
3673     \expandafter #1\expandafter #2\expandafter{%
3674     \romannumeral`&&@\XINT:NEhook:branch{\romannumeral`&&@\calxifNotZero #3}}%
3675 }%
3676 \let\XINT_iiexpr_func_if\XINT_flexpr_func_if
3677 \def\XINT_flexpr_func_ifint #1#2#3%
3678 {%
3679     \expandafter #1\expandafter #2\expandafter{%
3680     \romannumeral`&&@\XINT:NEhook:branch{\romannumeral`&&@\calxFpifInt #3}}%
3681 }%
3682 \def\XINT_iiexpr_func_ifint #1#2#3%
3683 {%
3684     \expandafter #1\expandafter #2\expandafter{%
3685     \romannumeral`&&@\XINT:NEhook:branch{\romannumeral`&&@\calxiiifInt #3}}%
3686 }%
3687 \def\XINT_flexpr_func_ifone #1#2#3%
3688 {%
3689     \expandafter #1\expandafter #2\expandafter{%
3690     \romannumeral`&&@\XINT:NEhook:branch{\romannumeral`&&@\calxFpifOne #3}}%
3691 }%
3692 \def\XINT_iiexpr_func_ifone #1#2#3%
3693 {%
3694     \expandafter #1\expandafter #2\expandafter{%
3695     \romannumeral`&&@\XINT:NEhook:branch{\romannumeral`&&@\calxiiifOne #3}}%
3696 }%
3697 \def\XINT_flexpr_func_ifsgn #1#2#3%
3698 {%
3699     \expandafter #1\expandafter #2\expandafter{%
3700     \romannumeral`&&@\XINT:NEhook:branch{\romannumeral`&&@\calxifSgn #3}}%
3701 }%
3702 \let\XINT_iiexpr_func_ifsgn\XINT_flexpr_func_ifsgn
3703 \def\XINT_flexpr_func_nuple #1#2#3{#1#2{{#3}}}%
3704 \let\XINT_iiexpr_func_nuple\XINT_flexpr_func_nuple
3705 \def\XINT_flexpr_func_unpack #1#2%#3%
3706    {\expandafter#1\expandafter#2\romannumeral0\XINT:NEhook:unpack}%
3707 \let\XINT_iiexpr_func_unpack\XINT_flexpr_func_unpack
3708 \def\XINT_flexpr_func_flat #1#2%#3%
3709 {%
3710     \expandafter#1\expandafter#2\expanded
3711     \XINT:NEhook:x:flatten\XINT:expr:flatten
3712 }%
3713 \let\XINT_iiexpr_func_flat\XINT_flexpr_func_flat
3714 \let\XINT:NEhook:x:flatten\empty
```

```
\def\XINT_flexpr_func_zip #1#2%#3%
{%
    \expandafter#1\expandafter#2\romannumeral`&&@%
    \XINT:NEhook:x:zip\XINT:expr:zip
}%
\let\XINT_iiexpr_func_zip\XINT_flexpr_func_zip
\let\XINT:NEhook:x:zip\empty
\def\XINT:expr:zip#1{\expandafter{\expanded\XINT_zip_A#1\xint_bye\xint_bye}}%
```

32.30. User declared functions

It is possible that the author actually does understand at this time the \xintNewExpr/\xintdeffunc refactored code and mechanisms for the first time since 2014: past evolutions such as the 2018 1.3 refactoring were done a bit in the fog (although they did accomplish a crucial step).

The 1.4 version of function and macro definitions is much more powerful than 1.3 one. But the mechanisms such as «omit», «abort» and «break()» in iter() et al. can't be translated into much else than their actual code when they potentially have to apply to non-numeric only context. The 1.4 \xintdeffunc is thus apparently able to digest them but its pre-parsing benefits are limited compared to simply assigning such parts of an expression to a mock-function created by \xintNewFunction (which creates simply a TeX macro from its substitution expression in macro parameters and add syntactic sugar to let it appear to \xintexpr as a genuine «function» although nothing of the syntax has really been pre-parsed.)

At 1.4 fetching the expression up to final semi-colon is done using \XINT_expr_fetch_to_semicolon, hence semi-colons arising in the syntax do not need to be hidden inside braces.

32.30.1 \xintdeffunc, \xintdefiifunc, \xintdeffloatfunc . . . 806
32.30.2 \xintdefufunc, \xintdefiiufunc, \xintdeffloatufunc . . . 809
32.30.3 \xintunassignexprfunc, \xintunassigniiexprfunc, \xintunassignfloatexprfunc . 811
32.30.4 \xintNewFunction . . . 811
32.30.5 Mysterious stuff . . . 812
32.30.6 \XINT_expr_redefinemacros . . . 824
32.30.7 Still more mysterious . . . 826
32.30.8 \xintexprSafeCatcodes, \xintexprRestoreCatcodes . . . 828

32.30.1. \xintdeffunc, \xintdefiifunc, \xintdeffloatfunc

Modified at 1.2c (2015/11/16). Note: it is possible to have same name assigned both to a variable and a function: things such as add(f(f), f=1..10) are possible.

Modified at 1.2c (2015/11/16). Function names first expanded then detokenized and cleaned of spaces.

Modified at 1.2e (2015/11/22). No \detokenize anymore on the function names. And #1(#2)#3=#4 parameter pattern to avoid to have to worry if a : is there and it is active.

Modified at 1.2f (2016/03/12). La macro associée à la fonction ne débute plus par un \romannumeral, car de toute façon elle est pour emploi dans \csname..\endcsname.

Modified at 1.2f (2016/03/12). Comma separated expressions allowed (formerly this required using parenthesis \xintdeffunc foo(x,..):=(.., .., ..);

Modified at 1.3c (2018/06/17). Usage of \xintexprSafeCatcodes to be compatible with an active semi-colon at time of use; the colon was not a problem (see ##3) already.

Modified at 1.3e (2019/04/05). \xintdefefunc variant added for functions which will expand completely if used with numeric arguments in other function definitions. They can't be used for recursive definitions.

Their functionality was merged into \xintdeffunc et al. at 1.4. The original macros were removed at 1.4m.

Modified at 1.4 (2020/01/31). Multi-letter variables can be used (with no prior declaration)

Modified at 1.4 (2020/01/31). The new internal data model has caused many worries initially (such as whether to allow functions with «ople» outputs in contrast to «numbers» or «nutples») but in the end all is simpler again and the refactoring of ? and ?? in function definitions allows to fuse inert functions (allowing recursive definitions) and expanding functions (expanding completely if with numeric arguments) into a single entity.

A special situation is with functions of no variables. In that case it will be handled as an inert entity, else they would not be different from variables.

Modified at 1.4 (2020/01/31). Addition de la syntaxe déclarative \xintdeffunc foo(a,b,...,*z) = ...;

Modified at 1.4m (2022/06/10). Removal of the \xintdefefunc et al.macros deprecated at 1.4.

```
\def\XINT_def_deffunc #1#2#3#4#5%
{%
  \def #1##1(##2)##3={%
   \edef\XINT_deffunc_tmpa {##1}%
   \edef\XINT_deffunc_tmpa {\xint_zapspaces_o \XINT_deffunc_tmpa}%
   \def\XINT_deffunc_tmpb {0}%
   \edef\XINT_deffunc_tmpd {##2}%
   \edef\XINT_deffunc_tmpd {\xint_zapspaces_o\XINT_deffunc_tmpd}%
   \def\XINT_deffunc_tmpe {0}%
   \expandafter#5\romannumeral\XINT_expr_fetch_to_semicolon
  }%  end of \xintdeffunc_a definition
  \def#5##1{%
   \def\XINT_deffunc_tmpc{(##1)}%
   \ifnum\xintLength:f:csv{\XINT_deffunc_tmpd}>\xint_c_
     \xintFor ####1 in {\XINT_deffunc_tmpd}\do
      {%
       \xintifForFirst{\let\XINT_deffunc_tmpd\empty}{}%
       \def\XINT_deffunc_tmpf{####1}%
       \if*\xintFirstItem{####1}%
         \xintifForLast
         {%
          \def\XINT_deffunc_tmpe{1}%
          \edef\XINT_deffunc_tmpf{\xintTrim{1}{####1}}%
         }%
         {%
          \edef\XINT_deffunc_tmpf{\xintTrim{1}{####1}}%
          \xintMessage{xintexpr}{Error}
          {Only the last positional argument can be variadic. Trimmed ####1 to
          \XINT_deffunc_tmpf}%
         }%
       \fi
       \XINT_expr_makedummy{\XINT_deffunc_tmpf}%
       \edef\XINT_deffunc_tmpd{\XINT_deffunc_tmpd{\XINT_deffunc_tmpf}}%
       \edef\XINT_deffunc_tmpb {\the\numexpr\XINT_deffunc_tmpb+\xint_c_i}%
       \edef\XINT_deffunc_tmpc {subs(\xint_noxpd\expandafter{\XINT_deffunc_tmpc},%
                                \XINT_deffunc_tmpf=################\XINT_deffunc_tmpb)}%
      }%
  \fi
```

Logic at 1.4 is simplified here compared to earlier releases.

Modified at 1.4n (2025/09/05). Usage of \xintmeaning wrapper of engine's \meaning. See near \XINT_NewExpr for explanations.

```
    \ifcase\XINT_deffunc_tmpb\space
      \expandafter\XINT_expr_defuserfunc_none\csname
    \else
      \expandafter\XINT_expr_defuserfunc\csname
    \fi
            XINT_#2_func_\XINT_deffunc_tmpa\expandafter\endcsname
    \csname XINT_#2_userfunc_\XINT_deffunc_tmpa\expandafter\endcsname
    \expandafter{\XINT_deffunc_tmpa}{#2}%
    \expandafter#3\csname XINT_#2_userfunc_\XINT_deffunc_tmpa\endcsname
                               [\XINT_deffunc_tmpb]{\XINT_deffunc_tmpc}%
    \ifxintverbose\xintMessage {xintexpr}{Info}%
         {Function \XINT_deffunc_tmpa\space for \string\xint #4 parser
          associated to \string\XINT_#2_userfunc_\XINT_deffunc_tmpa\space
          with \ifxintglobaldefs global \fi meaning \expandafter\xintmeaning
          \csname XINT_#2_userfunc_\XINT_deffunc_tmpa\endcsname}%
    \fi
    \xintFor* ####1 in {\XINT_deffunc_tmpd}:{\xintrestorevariablesilently{####1}}%
    \xintexprRestoreCatcodes
  }% end of \xintdeffunc_b definition
}%
\def\xintdefiifunc    {\xintexprSafeCatcodes\xintdefiifunc_a}%
\def\xintdeffloatfunc {\xintexprSafeCatcodes\xintdeffloatfunc_a}%
\XINT_def_deffunc\xintdefiifunc_a   {iiexpr}\XINT_NewIIFunc   {iiexpr}\xintdefiifunc_b
\XINT_def_deffunc\xintdeffloatfunc_a{flexpr}\XINT_NewFloatFunc{floatexpr}\xintdeffloatfunc_b
\def\XINT_expr_defuserfunc_none #1#2#3#4%
{%
    \XINT_global
    \def #1##1##2##3%
    {%
         \expandafter##1\expandafter##2\expanded{%
           {\XINT:NEhook:usernoargfunc\csname XINT_#4_userfunc_#3\endcsname}%
         }%
    }%
}%
\let\XINT:NEhook:usernoargfunc \empty
\def\XINT_expr_defuserfunc #1#2#3#4%
{%
    \if0\XINT_deffunc_tmpe
    \XINT_global
    \def #1##1##2%##3%
    {%
         \expandafter ##1\expandafter##2\expanded\bgroup{\iffalse}\fi
         \XINT:NEhook:userfunc{XINT_#4_userfunc_#3}#2%##3%
    }%
    \else
```

Last argument in the call signature is variadic (was prefixed by *).

```
    \def #1##1{%
    \XINT_global\def #1####1####2%####3%
    {%
```

```
3809          \expandafter ####1\expandafter####2\expanded\bgroup{\iffalse}\fi
3810          \XINT:NEhook:userfunc:argv{##1}{XINT_#4_userfunc_#3}#2%####3%
3811      }}\expandafter#1\expandafter{\the\numexpr\XINT_deffunc_tmpb-1}%
3812      \fi
3813 }%
```

Deliberate brace stripping of #3 to reveal the elements of the ople, which may be atoms i.e. numeric data such as {1}, or again ople, which means that the corresponding item was a nutple, for example it came from input syntax such as foo(1, 2, [1, 2], 3), so (up to details of raw encoding) {1}{2}{{1}{2}}{3}, which gives 4 braced arguments to macro #2.

```
3814 \def\XINT:NEhook:userfunc #1#2#3{#2#3\iffalse{{\fi}}}%
```

Here #1 indicates the number k-1 of standard positional arguments of the call signature, the kth and last one having been declared of variadic type. The braces around \xintTrim{#1}{#4} have the effect to gather all these remaining elements to provide a single one to the TeX macro.

For example input was foo(1,2,3,4,5) and call signature was foo(a,b,*z). Then #4 will fetch {{1}{2}{3}{4}{5}}, with one level of brace removal. We will have \xintKeep{2}{{1}{2}{3}{4}{5}} which produces {1}{2}. Then {\xintTrim{2}{{1}{2}{3}{4}{5}}} which produces {{3}{4}{5}}. So the macro will be used as \macro{1}{2}{{3}{4}{5}} having been declared as a macro with 3 arguments.

The above comments were added in June 2021 but the code was done on January 19, 2020 for 1.4.

Note on June 10, 2021: at core level \XINT_NewFunc is used which is derived from \XINT_NewExpr which has always prepared TeX macros with non-delimited parameters. A refactoring could add a final delimiter, for example \relax. The macro with 3 arguments would be defined as \def\macro#1#2#3\relax{...} for example. Then we could transfer to TeX core processing what is achieved here via \xintKeep/\xintTrim, of course adding efficiency, via insertion of the delimiter. In the case of foo(1,2,3,4,5) we would have the #3 of delimited \macro fetch {3}{4}{5}, no brace removal, which is equivalent to current situation fetching {{3}{4}{5}} with brace removal. But let's see in case of foo(1,2,3) then. This would lead to delimited \macro{1}{2}{3}\relax and #3 will fetch {3}, removing one brace pair. Whereas current non-delimited \macro is used as \macro{1}{2}{{3}} from the Keep/Trim, then #3 fetches {{3}}, removing one brace pair. Not the same thing. So it seems there is a stumbling-block here to adopt such an alternative method, in relation with brace removal. Rather relieved in fact, as my head starts spinning in ople world. Seems better to stop thinking about doing something like that, and what it would imply as consequences for user declarative interface also. Oples are dangerous to mental health, let's stick with one-ples: « named arguments in function body declaration must stand for one-ples », even the last one, although a priori it could be envisioned if foo has been declared with call signature (x,y,z) and is used with more items that z is mapped to the ople of extra elements beyond the first two ones. For my sanity I stick with my January 2020 concept of (x,y,*z) which makes z stand for a nutple always and having to be used as such in the function body (possibly unpacked there using *z).

```
3815 \def\XINT:NEhook:userfunc:argv #1#2#3#4%
3816    {\expandafter#3\expanded{\xintKeep{#1}{#4}{\xintTrim{#1}{#4}}}\iffalse{{\fi}}}%
```

32.30.2. \xintdefufunc, \xintdefiiufunc, \xintdeffloatufunc

Added at 1.4

Modified at 1.4k (2022/05/18). \xintexprRestoreCatcodes was in only one branch of \xint_defufunc_b, and as a result sanitization of catcodes via \xintexprSafeCatcodes was never reverted.

That the bug remained unseen and in particular did not break compilation of user manual (where the | must be active), was a sort of unhappy miracle due to the | ending up recovering its active catcode from some ulterior \xintdefiifunc whose Safe/Restore behaved as described in the user manual, i.e. it did a restore to the state before the first unpaired Safe, and this miraculous recovery happened before breakage had happened, by sheer luck, or rather lack of luck, else I would have seen and solved the problem two years ago...

```
\def\XINT_def_defufunc #1#2#3#4#5#6%
{%
  \def #1##1(##2)##3={%
   \edef\XINT_defufunc_tmpa {##1}%
   \edef\XINT_defufunc_tmpa {\xint_zapspaces_o \XINT_defufunc_tmpa}%
   \edef\XINT_defufunc_tmpd {##2}%
   \edef\XINT_defufunc_tmpd {\xint_zapspaces_o\XINT_defufunc_tmpd}%
   \expandafter#5\romannumeral\XINT_expr_fetch_to_semicolon
  }% end of \xint_defufunc_a
  \def#5##1{%
   \def\XINT_defufunc_tmpc{(##1)}%
   \ifnum\xintLength:f:csv{\XINT_defufunc_tmpd}=\xint_c_i
     \expandafter#6%
   \else
     \xintMessage {xintexpr}{ERROR}
        {Universal functions must be functions of one argument only,
         but the declaration of \XINT_defufunc_tmpa\space
         has \xintLength:f:csv{\XINT_defufunc_tmpd} of them. Canceled.}%
     \xintexprRestoreCatcodes
   \fi
   }% end of \xint_defufunc_b
  \def #6{%
   \XINT_expr_makedummy{\XINT_defufunc_tmpd}%
   \edef\XINT_defufunc_tmpc {subs(\xint_noxpd\expandafter{\XINT_defufunc_tmpc},%
                             \XINT_defufunc_tmpd=########1)}%
   \expandafter\XINT_expr_defuserufunc
   \csname XINT_#2_func_\XINT_defufunc_tmpa\expandafter\endcsname
   \csname XINT_#2_userufunc_\XINT_defufunc_tmpa\expandafter\endcsname
   \expandafter{\XINT_defufunc_tmpa}{#2}%
   \expandafter#3\csname XINT_#2_userufunc_\XINT_defufunc_tmpa\endcsname
                               [1]{\XINT_defufunc_tmpc}%
   \ifxintverbose\xintMessage {xintexpr}{Info}%
        {Universal function \XINT_defufunc_tmpa\space for \string\xint #4 parser
         associated to \string\XINT_#2_userufunc_\XINT_defufunc_tmpa\space
         with \ifxintglobaldefs global \fi meaning \expandafter\xintmeaning
         \csname XINT_#2_userufunc_\XINT_defufunc_tmpa\endcsname}%
   \fi
   \xintexprRestoreCatcodes
  }% end of \xint_defufunc_c
}%
\def\xintdefiiufunc    {\xintexprSafeCatcodes\xintdefiiufunc_a}%
\def\xintdeffloatufunc {\xintexprSafeCatcodes\xintdeffloatufunc_a}%
\XINT_def_defufunc\xintdefiiufunc_a   {iiexpr}\XINT_NewIIFunc   {iiexpr}%
          \xintdefiiufunc_b\xintdefiiufunc_c
\XINT_def_defufunc\xintdeffloatufunc_a{flexpr}\XINT_NewFloatFunc{floatexpr}%
          \xintdeffloatufunc_b\xintdeffloatufunc_c
\def\XINT_expr_defuserufunc #1#2#3#4%
{%
    \XINT_global
    \def #1##1##2%##3%
    {%
         \expandafter ##1\expandafter##2\expanded
```

```
          \XINT:NEhook:userufunc{XINT_#4_userufunc_#3}#2%##3%
    }%
}%
\def\XINT:NEhook:userufunc #1{\XINT:expr:mapwithin}%
```

32.30.3. \xintunassignexprfunc, \xintunassigniiexprfunc, \xintunassignfloatexprfunc

See the \xintunassignvar for the embarrassing explanations why I had not done that earlier. A bit lazy here, no warning if undefining something not defined, and attention no precaution respective built-in functions.

```
\def\XINT_def_unassignfunc #1{\expandafter\def\csname xintunassign#1func\endcsname ##1{%
   \edef\XINT_unfunc_tmpa{##1}%
   \edef\XINT_unfunc_tmpa {\xint_zapspaces_o\XINT_unfunc_tmpa}%
   \XINT_global\expandafter
     \let\csname XINT_#1_func_\XINT_unfunc_tmpa\endcsname\XINT_undefined
   \XINT_global\expandafter
     \let\csname XINT_#1_userfunc_\XINT_unfunc_tmpa\endcsname\XINT_undefined
   \XINT_global\expandafter
     \let\csname XINT_#1_userufunc_\XINT_unfunc_tmpa\endcsname\XINT_undefined
   \ifxintverbose\xintMessage {xintexpr}{Info}%
    {Function \XINT_unfunc_tmpa\space for \string\xint #1 parser now
     \ifxintglobaldefs globally \fi undefined.}%
   \fi}}%
\XINT_def_unassignfunc{iiexpr}\XINT_def_unassignfunc{floatexpr}%
```

32.30.4. \xintNewFunction

1.2h (2016/11/20). Syntax is \xintNewFunction{<name>}[nb of arguments]{expression with #1, #2,... as in \xintNewExpr}. This defines a function for all three parsers but the expression parsing is delayed until function execution. Hence the expression admits all constructs, contrarily to \xintNewExpr or \xintdeffunc.

As the letters used for variables in \xintdeffunc, #1, #2, etc... can not stand for non numeric «oples», because at time of function call f(a, b, c, ...) how to decide if #1 stands for a or a, b etc... ? Or course «a» can be packed and thus the macro function can handle #1 as a «nutple» and for this be defined with the * unpacking operator being applied to it.

```
\def\xintNewFunction #1#2[#3]#4%
{%
  \edef\XINT_newfunc_tmpa {#1}%
  \edef\XINT_newfunc_tmpa {\xint_zapspaces_o \XINT_newfunc_tmpa}%
  \def\XINT_newfunc_tmpb ##1##2##3##4##5##6##7##8##9{#4}%
  \begingroup
    \ifcase #3\relax
        \toks0{}%
    \or \toks0{##1}%
    \or \toks0{##1##2}%
    \or \toks0{##1##2##3}%
    \or \toks0{##1##2##3##4}%
    \or \toks0{##1##2##3##4##5}%
    \or \toks0{##1##2##3##4##5##6}%
    \or \toks0{##1##2##3##4##5##6##7}%
    \or \toks0{##1##2##3##4##5##6##7##8}%
    \else \toks0{##1##2##3##4##5##6##7##8##9}%
    \fi
```

```
    \expandafter
  \endgroup\expandafter
  \XINT_global\expandafter
  \def\csname XINT_expr_macrofunc_\XINT_newfunc_tmpa\expandafter\endcsname
  \the\toks0\expandafter{\XINT_newfunc_tmpb
    {\XINTfstop.{{##1}}}{\XINTfstop.{{##2}}}{\XINTfstop.{{##3}}}%
    {\XINTfstop.{{##4}}}{\XINTfstop.{{##5}}}{\XINTfstop.{{##6}}}%
    {\XINTfstop.{{##7}}}{\XINTfstop.{{##8}}}{\XINTfstop.{{##9}}}}%
  \expandafter\XINT_expr_newfunction
    \csname XINT_expr_func_\XINT_newfunc_tmpa\expandafter\endcsname
    \expandafter{\XINT_newfunc_tmpa}\xintbareeval
  \expandafter\XINT_expr_newfunction
    \csname XINT_iiexpr_func_\XINT_newfunc_tmpa\expandafter\endcsname
    \expandafter{\XINT_newfunc_tmpa}\xintbareiieval
  \expandafter\XINT_expr_newfunction
    \csname XINT_flexpr_func_\XINT_newfunc_tmpa\expandafter\endcsname
    \expandafter{\XINT_newfunc_tmpa}\xintbarefloateval
  \ifxintverbose
    \xintMessage {xintexpr}{Info}%
        {Function \XINT_newfunc_tmpa\space for the expression parsers is
         associated to \string\XINT_expr_macrofunc_\XINT_newfunc_tmpa\space
         with \ifxintglobaldefs global \fi meaning \expandafter\xintmeaning
         \csname XINT_expr_macrofunc_\XINT_newfunc_tmpa\endcsname}%
  \fi
}%
\def\XINT_expr_newfunction #1#2#3%
{%
    \XINT_global
    \def#1##1##2##3%
      {\expandafter ##1\expandafter ##2%
       \romannumeral0\XINT:NEhook:macrofunc
       #3{\csname XINT_expr_macrofunc_#2\endcsname##3}\relax
      }%
}%
\let\XINT:NEhook:macrofunc\empty
```

32.30.5. Mysterious stuff

There was an \xintNewExpr already in 1.07 from May 2013, which was modified in September 2013 to work with the # macro parameter character, and then refactored into a more powerful version in June 2014 for 1.1 release of 2014/10/28.

It is always too soon to try to comment and explain. In brief, this attempts to hack into the *purely numeric* \xintexpr parsers to transform them into *symbolic* parsers, allowing to do once and for all the parsing job and inherit a gigantic nested macro. Originally only f-expandable nesting. The initial motivation was that the \csname encapsulation impacted the string pool memory. Later this work proved to be the basis to provide support for implementing user-defined functions and it is now its main purpose.

Deep refactorings happened at 1.3 and 1.4.

At 1.3 the crucial idea of the «hook» macros was introduced, reducing considerably the preparatory work done by \xintNewExpr.

At 1.4 further considerable simplifications happened, and it is possible that the author currently does at long last understand the code!

The 1.3 code had serious complications with trying to identify would-be «list» arguments, distinguishing them from «single» arguments (things like parsing `#2+[[#1..[#3]..#4][#5:#6]]*#7` and convert it to a single nested f-exandable macro...)

The conversion at 1.4 is both more powerful and simpler, due in part to the new storage model which from `\csname` encapsulated comma separated values up to 1.3f became simply a braced list of braced values, and also crucially due to the possibilities opened up by usage of `\expanded` primitive.

```
\catcode`~ 12
\def\XINT:NE:hastilde#1~#2#3\relax{\unless\if !#21\fi}%
\def\XINT:NE:hashash#1{%
\def\XINT:NE:hashash##1#1##2##3\relax{\unless\if !##21\fi}%
}\expandafter\XINT:NE:hashash\string#%
\def\XINT:NE:unpack #1{%
\def\XINT:NE:unpack ##1%
{%
    \if0\XINT:NE:hastilde ##1~!\relax
        \XINT:NE:hashash  ##1#1!\relax 0\else
        \expandafter\XINT:NE:unpack:p\fi
    \xint_stop_atfirstofone{##1}%
}}\expandafter\XINT:NE:unpack\string#%
\def\XINT:NE:unpack:p#1#2%
    {{~romannumeral0~expandafter~xint_stop_atfirstofone~expanded{#2}}}%
\def\XINT:NE:f:one:from:one #1{%
\def\XINT:NE:f:one:from:one ##1%
{%
    \if0\XINT:NE:hastilde ##1~!\relax
        \XINT:NE:hashash  ##1#1!\relax 0\else
        \xint_dothis\XINT:NE:f:one:from:one_a\fi
    \xint_orthat\XINT:NE:f:one:from:one_b
    ##1&&A%
}}\expandafter\XINT:NE:f:one:from:one\string#%
\def\XINT:NE:f:one:from:one_a\romannumeral`&&@#1#2&&A%
{%
    \expandafter{\detokenize{\expandafter#1}#2}%
}%
\def\XINT:NE:f:one:from:one_b#1{%
\def\XINT:NE:f:one:from:one_b\romannumeral`&&@##1##2&&A%
{%
    \expandafter{\romannumeral`&&@%
        \if0\XINT:NE:hastilde ##2~!\relax
            \XINT:NE:hashash  ##2#1!\relax 0\else
        \expandafter\string\fi
    ##1{##2}}%
}}\expandafter\XINT:NE:f:one:from:one_b\string#%
\def\XINT:NE:f:one:from:one:direct #1#2{\XINT:NE:f:one:from:one:direct_a #2&&A{#1}}%
\def\XINT:NE:f:one:from:one:direct_a #1#2&&A#3%
{%
    \if ###1\xint_dothis {\detokenize{#3}}\fi
    \if  ~#1\xint_dothis {\detokenize{#3}}\fi
    \xint_orthat {#3}{#1#2}%
}%
\def\XINT:NE:f:one:from:two #1{%
```

```
\def\XINT:NE:f:one:from:two ##1%
{%
    \if0\XINT:NE:hastilde ##1~!\relax
        \XINT:NE:hashash  ##1#1!\relax 0\else
        \xint_dothis\XINT:NE:f:one:from:two_a\fi
    \xint_orthat\XINT:NE:f:one:from:two_b ##1&&A%
}}\expandafter\XINT:NE:f:one:from:two\string#%
\def\XINT:NE:f:one:from:two_a\romannumeral`&&@#1#2&&A%
{%
    \expandafter{\detokenize{\expandafter#1\expanded}{#2}}%
}%
\def\XINT:NE:f:one:from:two_b#1{%
\def\XINT:NE:f:one:from:two_b\romannumeral`&&@##1##2##3&&A%
{%
    \expandafter{\romannumeral`&&@%
        \if0\XINT:NE:hastilde ##2##3~!\relax
            \XINT:NE:hashash  ##2##3#1!\relax 0\else
        \expandafter\string\fi
    ##1{##2}{##3}}%
}}\expandafter\XINT:NE:f:one:from:two_b\string#%
\def\XINT:NE:f:one:from:two:direct #1#2#3{\XINT:NE:two_fork #2&&A#3&&A#1{#2}{#3}}%
\def\XINT:NE:two_fork #1#2&&A#3#4&&A{\XINT:NE:two_fork_nn#1#3}%
\def\XINT:NE:two_fork_nn #1#2%
{%
    \if #1##\xint_dothis\string\fi
    \if  #1~\xint_dothis\string\fi
    \if #2##\xint_dothis\string\fi
    \if  #2~\xint_dothis\string\fi
    \xint_orthat{}%
}%
\def\XINT:NE:f:one:and:opt:direct#1{%
\def\XINT:NE:f:one:and:opt:direct##1!%
{%
    \if0\XINT:NE:hastilde ##1~!\relax
        \XINT:NE:hashash  ##1#1!\relax 0\else
        \xint_dothis\XINT:NE:f:one:and:opt_a\fi
    \xint_orthat\XINT:NE:f:one:and:opt_b ##1&&A%
}}\expandafter\XINT:NE:f:one:and:opt:direct\string#%
\def\XINT:NE:f:one:and:opt_a #1#2&&A#3#4%
{%
    \detokenize{\romannumeral-`0\expandafter#1\expanded{#2}$XINT_expr_exclam#3#4}%$
}%
\def\XINT:NE:f:one:and:opt_b\XINT:expr:f:one:and:opt #1#2#3&&A#4#5%
{%
    \if\relax#3\relax\expandafter\xint_firstoftwo\else
                       \expandafter\xint_secondoftwo\fi
    {\XINT:NE:f:one:from:one:direct#4}%
    {\expandafter\XINT:NE:f:onewithopttoone\expandafter#5%
        \expanded{{\XINT:NE:f:one:from:one:direct\calxListItemToNum{#2}}}}%
    {#1}%
}%
\def\XINT:NE:f:onewithopttoone#1#2#3{\XINT:NE:two_fork #2&&A#3&&A#1[#2]{#3}}%
```

```
\def\XINT:NE:f:tacitzeroifone:direct#1{%
\def\XINT:NE:f:tacitzeroifone:direct##1!%
{%
    \if0\XINT:NE:hastilde ##1~!\relax
        \XINT:NE:hashash  ##1#1!\relax 0\else
        \xint_dothis\XINT:NE:f:one:and:opt_a\fi
    \xint_orthat\XINT:NE:f:tacitzeroifone_b ##1&&A%
}}\expandafter\XINT:NE:f:tacitzeroifone:direct\string#%
\def\XINT:NE:f:tacitzeroifone_b\XINT:expr:f:tacitzeroifone #1#2#3&&A#4#5%
{%
    \if\relax#3\relax\expandafter\xint_firstoftwo\else
                       \expandafter\xint_secondoftwo\fi
    {\XINT:NE:f:one:from:two:direct#4{0}}%
    {\expandafter\XINT:NE:f:one:from:two:direct\expandafter#5%
        \expanded{{\XINT:NE:f:one:from:one:direct\calxListItemToNum{#2}}}}%
    {#1}%
}%
\def\XINT:NE:f:iitacitzeroifone:direct#1{%
\def\XINT:NE:f:iitacitzeroifone:direct##1!%
{%
    \if0\XINT:NE:hastilde ##1~!\relax
        \XINT:NE:hashash  ##1#1!\relax 0\else
        \xint_dothis\XINT:NE:f:iitacitzeroifone_a\fi
    \xint_orthat\XINT:NE:f:iitacitzeroifone_b ##1&&A%
}}\expandafter\XINT:NE:f:iitacitzeroifone:direct\string#%
\def\XINT:NE:f:iitacitzeroifone_a #1#2&&A#3%
{%
    \detokenize
    {\romannumeral`$XINT_expr_null\expandafter#1\expanded{#2}$XINT_expr_exclam#3}%
}%
\def\XINT:NE:f:iitacitzeroifone_b\XINT:expr:f:iitacitzeroifone #1#2#3&&A#4%
{%
    \if\relax#3\relax\expandafter\xint_firstoftwo\else
                       \expandafter\xint_secondoftwo\fi
    {\XINT:NE:f:one:from:two:direct#4{0[0]}}%
    {\XINT:NE:f:one:from:two:direct#4{#2}}%
    {#1}%
}%
\def\XINT:NE:x:one:from:two #1#2#3{\XINT:NE:x:one:from:two_fork #2&&A#3&&A#1{#2}{#3}}%
\def\XINT:NE:x:one:from:two_fork #1{%
\def\XINT:NE:x:one:from:two_fork ##1##2&&A##3##4&&A%
{%
    \if0\XINT:NE:hastilde ##1##3~!\relax\XINT:NE:hashash ##1##3#1!\relax 0%
    \else
        \expandafter\XINT:NE:x:one:from:two:p
    \fi
}}\expandafter\XINT:NE:x:one:from:two_fork\string#%
\def\XINT:NE:x:one:from:two:p #1#2#3%
   {~expanded{\detokenize{\expandafter#1}~expanded{{#2}{#3}}}}%
\def\XINT:NE:x:listsel #1{%
\def\XINT:NE:x:listsel ##1##2&%
{%
```

```
    \if0\expandafter\XINT:NE:hastilde\detokenize{##2}~!\relax
        \expandafter\XINT:NE:hashash\detokenize{##2}#1!\relax 0%
    \else
        \expandafter\XINT:NE:x:listsel:p
    \fi
    ##1##2&%
}}\expandafter\XINT:NE:x:listsel\string#%
\def\XINT:NE:x:listsel:p #1#2_#3&(#4%
{%
    \detokenize{\expanded\XINT:expr:ListSel{{#3}{#4}}}%
}%
\def\XINT:expr:ListSel{\expandafter\XINT:expr:ListSel_i\expanded}%
\def\XINT:expr:ListSel_i #1#2{{\XINT_ListSel_top #2_#1&({#2}}}%
\def\XINT:NE:f:reverse #1{%
\def\XINT:NE:f:reverse ##1^%
{%
    \if0\expandafter\XINT:NE:hastilde\detokenize\expandafter{\xint_gobble_i##1}~!\relax
        \expandafter\XINT:NE:hashash\detokenize{##1}#1!\relax 0%
    \else
        \expandafter\XINT:NE:f:reverse:p
    \fi
    ##1^%
}}\expandafter\XINT:NE:f:reverse\string#%
\def\XINT:NE:f:reverse:p #1^#2\xint_bye
{%
    \expandafter\XINT:NE:f:reverse:p_i\expandafter{\xint_gobble_i#1}%
}%
\def\XINT:NE:f:reverse:p_i #1%
{%
    \detokenize{\romannumeral0\XINT:expr:f:reverse{{#1}}}%
}%
\def\XINT:expr:f:reverse{\expandafter\XINT:expr:f:reverse_i\expanded}%
\def\XINT:expr:f:reverse_i #1%
{%
    \XINT_expr_reverse #1^^#1\xint:\xint:\xint:\xint:
                           \xint:\xint:\xint:\xint:\xint_bye
}%
\def\XINT:NE:f:from:delim:u #1{%
\def\XINT:NE:f:from:delim:u ##1##2^%
{%
    \if0\expandafter\XINT:NE:hastilde\detokenize{##2}~!\relax
        \expandafter\XINT:NE:hashash\detokenize{##2}#1!\relax 0%
        \xint_afterfi{\expandafter\XINT_fooof_checkifnumber\expandafter##1\string}%
    \else
        \xint_afterfi{\XINT:NE:f:from:delim:u:p##1\empty}%
    \fi
    ##2^%
}}\expandafter\XINT:NE:f:from:delim:u\string#%
\def\XINT:NE:f:from:delim:u:p #1#2^%
{%
    \detokenize
    {\expandafter\XINT:fooof:checkifnumber\expandafter#1}~expanded{#2}$XINT_expr_caret%$
```

```
}%
\def\XINT:fooof:checkifnumber#1{\expandafter\XINT_fooof_checkifnumber\expandafter#1\string}%
\def\XINT:NE:f:LFL#1#2{\expandafter\XINT:NE:f:LFL_a\expandafter#1#2\XINT:NE:f:LFL_a}%
\def\XINT:NE:f:LFL_a#1#2%
{%
    \if#2i\else\expandafter\XINT:NE:f:LFL_p
    \fi #1%
}%
\def\XINT:NE:r:check#1{%
\def\XINT:NE:r:check##1\XINT:NE:f:LFL_a
{%
    \if0\expandafter\XINT:NE:hastilde\detokenize{##1}~!\relax%
        \expandafter\XINT:NE:hashash\detokenize{##1}#1!\relax 0%
    \else
        \expandafter\XINT:NE:r:check:p
    \fi
    1\expandafter{\romannumeral\XINT:NEsaved:r:check##1}%
}}\expandafter\XINT:NE:r:check\string#%
\def\XINT:NE:r:check:p 1\expandafter#1{\XINT:NE:r:check:p_i#1}%
\def\XINT:NE:r:check:p_i\romannumeral\XINT:NEsaved:r:check{\XINT:NE:r:check:p_ii\empty}%
\def\XINT:NE:r:check:p_ii#1^%
{%
    5~expanded{{~romannumeral~XINT:NEsaved:r:check#1$XINT_expr_caret}}%$
}%
\def\XINT:NE:f:LFL_p#1%
{%
    \detokenize{\romannumeral`$XINT_expr_null\expandafter#1}%$
}%
\catcode`- 11
\def\XINT:NE:exec_? #1#2%
{%
    \XINT:NE:exec_?_b #2&&A#1{#2}%
}%
\def\XINT:NE:exec_?_b #1{%
\def\XINT:NE:exec_?_b ##1&&A%
{%
    \if0\XINT:NE:hastilde ##1~!\relax
        \XINT:NE:hashash  ##1#1!\relax 0%
    \xint_dothis\XINT:NE:exec_?:x\fi
    \xint_orthat\XINT:NE:exec_?:p
}}\expandafter\XINT:NE:exec_?_b\string#%
\def\XINT:NE:exec_?:x #1#2#3%
{%
    \expandafter\XINT_expr_check-_after?\expandafter#1%
    \romannumeral`&&@\expandafter\XINT_expr_getnext\romannumeral0\CALX_ifnz#3%
}%
\def\XINT:NE:exec_?:p #1#2#3#4#5%
{%
    \csname XINT_expr_func_*If\expandafter\endcsname
    \romannumeral`&&@#2\XINTfstop.{#3},[#4],[#5])%
}%
\expandafter\def\csname XINT_expr_func_*If\endcsname #1#2#3%
```

```
{%
    #1#2{~expanded{~calxifNotZero#3}}%
}%
\def\XINT:NE:exec_?? #1#2#3%
{%
    \XINT:NE:exec_??_b #2&&A#1{#2}%
}%
\def\XINT:NE:exec_??_b #1{%
\def\XINT:NE:exec_??_b ##1&&A%
{%
    \if0\XINT:NE:hastilde ##1~!\relax
        \XINT:NE:hashash  ##1#1!\relax 0%
    \xint_dothis\XINT:NE:exec_??:x\fi
    \xint_orthat\XINT:NE:exec_??:p
}}\expandafter\XINT:NE:exec_??_b\string#%
\def\XINT:NE:exec_??:x #1#2#3%
{%
    \expandafter\XINT_expr_check-_after?\expandafter#1%
    \romannumeral`&&@\expandafter\XINT_expr_getnext\romannumeral0\CALX_ifsgn#3%
}%
\def\XINT:NE:exec_??:p #1#2#3#4#5#6%
{%
    \csname XINT_expr_func_*IfSgn\expandafter\endcsname
    \romannumeral`&&@#2\XINTfstop.{#3},[#4],[#5],[#6])%
}%
\expandafter\def\csname XINT_expr_func_*IfSgn\endcsname #1#2#3%
{%
    #1#2{~expanded{~calxifSgn#3}}%
}%
\catcode`- 12
\def\XINT:NE:branch #1%
{%
    \if0\XINT:NE:hastilde #1~!\relax 0\else
        \xint_dothis\XINT:NE:branch_a\fi
    \xint_orthat\XINT:NE:branch_b #1&&A%
}%
\def\XINT:NE:branch_a\romannumeral`&&@#1#2&&A%
{%
    \expandafter{\detokenize{\expandafter#1\expanded}{#2}}%
}%
\def\XINT:NE:branch_b#1{%
\def\XINT:NE:branch_b\romannumeral`&&@##1##2##3&&A%
{%
    \expandafter{\romannumeral`&&@%
        \if0\XINT:NE:hastilde ##2~!\relax
            \XINT:NE:hashash  ##2#1!\relax 0\else
        \expandafter\string\fi
    ##1{##2}##3}%
}}\expandafter\XINT:NE:branch_b\string#%
\def\XINT:NE:seqx#1{%
\def\XINT:NE:seqx\XINT_allexpr_seqx##1##2%
{%
```

```
    \if 0\expandafter\XINT:NE:hastilde\detokenize{##2}~!\relax
         \expandafter\XINT:NE:hashash \detokenize{##2}#1!\relax 0%
    \else
      \expandafter\XINT:NE:seqx:p
    \fi \XINT_allexpr_seqx{##1}{##2}%
}}\expandafter\XINT:NE:seqx\string#%
\def\XINT:NE:seqx:p\XINT_allexpr_seqx #1#2#3#4%
{%
    \expandafter\XINT_expr_put_op_first
    \expanded {%
    {%
        \detokenize
        {%
           \expanded\bgroup
           \expanded
           {\xint_noxpd{\XINT_expr_seq:_b{#1#4\relax $XINT_expr_exclam #3}}%
             #2$XINT_expr_caret}%
        }%
    }%
    \expandafter}\romannumeral`&&@\XINT_expr_getop
}%
\def\XINT:NE:opx#1{%
\def\XINT:NE:opx\XINT_allexpr_opx ##1##2##3##4%##5##6##7##8%
{%
    \if 0\expandafter\XINT:NE:hastilde\detokenize{##4}~!\relax
         \expandafter\XINT:NE:hashash \detokenize{##4}#1!\relax 0%
    \else
      \expandafter\XINT:NE:opx:p
    \fi \XINT_allexpr_opx ##1{##2}{##3}{##4}% en fait ##2 = \xint_c_, ##3 = \relax
}}\expandafter\XINT:NE:opx\string#%
\def\XINT:NE:opx:p\XINT_allexpr_opx #1#2#3#4#5#6#7#8%
{%
    \expandafter\XINT_expr_put_op_first
    \expanded {%
    {%
        \detokenize
        {%
            \expanded\bgroup
            \expanded{\xint_noxpd{\XINT_expr_iter:_b
               {#1\expandafter\XINT_allexpr_opx_ifnotomitted
                  \romannumeral0#1#6\relax#7@\relax $XINT_expr_exclam #5}}%
                      #4$XINT_expr_caret$XINT_expr_tilde{{#8}}}%$
        }%
    }%
    \expandafter}\romannumeral`&&@\XINT_expr_getop
}%
\def\XINT:NE:iter{\expandafter\XINT:NE:itery\expandafter}%
\def\XINT:NE:itery#1{%
\def\XINT:NE:itery\XINT_expr_itery##1##2%
{%
    \if 0\expandafter\XINT:NE:hastilde\detokenize{##1##2}~!\relax
         \expandafter\XINT:NE:hashash \detokenize{##1##2}#1!\relax 0%
```

```
    \else
      \expandafter\XINT:NE:itery:p
    \fi \XINT_expr_itery{##1}{##2}%
}}\expandafter\XINT:NE:itery\string#%
\def\XINT:NE:itery:p\XINT_expr_itery #1#2#3#4#5%
{%
    \expandafter\XINT_expr_put_op_first
    \expanded {%
    {%
        \detokenize
        {%
          \expanded\bgroup
          \expanded{\xint_noxpd{\XINT_expr_iter:_b {#5#4\relax $XINT_expr_exclam #3}}%
                    #1$XINT_expr_caret$XINT_expr_tilde{#2}}%$
        }%
    }%
    \expandafter}\romannumeral`&&@\XINT_expr_getop
}%
\def\XINT:NE:rseq{\expandafter\XINT:NE:rseqy\expandafter}%
\def\XINT:NE:rseqy#1{%
\def\XINT:NE:rseqy\XINT_expr_rseqy##1##2%
{%
    \if 0\expandafter\XINT:NE:hastilde\detokenize{##1##2}~!\relax
         \expandafter\XINT:NE:hashash \detokenize{##1##2}#1!\relax 0%
    \else
      \expandafter\XINT:NE:rseqy:p
    \fi \XINT_expr_rseqy{##1}{##2}%
}}\expandafter\XINT:NE:rseqy\string#%
\def\XINT:NE:rseqy:p\XINT_expr_rseqy #1#2#3#4#5%
{%
    \expandafter\XINT_expr_put_op_first
    \expanded {%
    {%
        \detokenize
        {%
          \expanded\bgroup
          \expanded{#2\xint_noxpd{\XINT_expr_rseq:_b {#5#4\relax $XINT_expr_exclam #3}}%
                    #1$XINT_expr_caret$XINT_expr_tilde{#2}}%$
         }%
    }%
    \expandafter}\romannumeral`&&@\XINT_expr_getop
}%
\def\XINT:NE:iterr{\expandafter\XINT:NE:iterry\expandafter}%
\def\XINT:NE:iterry#1{%
\def\XINT:NE:iterry\XINT_expr_iterry##1##2%
{%
    \if 0\expandafter\XINT:NE:hastilde\detokenize{##1##2}~!\relax
         \expandafter\XINT:NE:hashash \detokenize{##1##2}#1!\relax 0%
    \else
      \expandafter\XINT:NE:iterry:p
    \fi \XINT_expr_iterry{##1}{##2}%
}}\expandafter\XINT:NE:iterry\string#%
```

```
\def\XINT:NE:iterry:p\XINT_expr_iterry #1#2#3#4#5%
{%
    \expandafter\XINT_expr_put_op_first
    \expanded {%
    {%
        \detokenize
        {%
          \expanded\bgroup
          \expanded{\xint_noxpd{\XINT_expr_iterr:_b {#5#4\relax $XINT_expr_exclam #3}}%
                    #1$XINT_expr_caret$XINT_expr_tilde #20$XINT_expr_qmark}%
        }%
    }%
    \expandafter}\romannumeral`&&@\XINT_expr_getop
}%
\def\XINT:NE:rrseq{\expandafter\XINT:NE:rrseqy\expandafter}%
\def\XINT:NE:rrseqy#1{%
\def\XINT:NE:rrseqy\XINT_expr_rrseqy##1##2%
{%
    \if 0\expandafter\XINT:NE:hastilde\detokenize{##1##2}~!\relax
         \expandafter\XINT:NE:hashash \detokenize{##1##2}#1!\relax 0%
    \else
      \expandafter\XINT:NE:rrseqy:p
    \fi \XINT_expr_rrseqy{##1}{##2}%
}}\expandafter\XINT:NE:rrseqy\string#%
\def\XINT:NE:rrseqy:p\XINT_expr_rrseqy #1#2#3#4#5#6%
{%
    \expandafter\XINT_expr_put_op_first
    \expanded {%
    {%
        \detokenize
        {%
           \expanded\bgroup
           \expanded{#2\xint_noxpd{\XINT_expr_rrseq:_b {#6#5\relax $XINT_expr_exclam #4}}%
                       #1$XINT_expr_caret$XINT_expr_tilde #30$XINT_expr_qmark}%
         }%
    }%
    \expandafter}\romannumeral`&&@\XINT_expr_getop
}%
\def\XINT:NE:x:toblist#1{%
\def\XINT:NE:x:toblist\XINT:expr:toblistwith##1##2%
{%
    \if 0\expandafter\XINT:NE:hastilde\detokenize{##2}~!\relax
         \expandafter\XINT:NE:hashash \detokenize{##2}#1!\relax 0%
    \else
      \expandafter\XINT:NE:x:toblist:p
    \fi \XINT:expr:toblistwith{##1}{##2}%
}}\expandafter\XINT:NE:x:toblist\string#%
\def\XINT:NE:x:toblist:p\XINT:expr:toblistwith #1#2{{\XINTfstop.{#2}}}%
\def\XINT:NE:x:flatten#1{%
\def\XINT:NE:x:flatten\XINT:expr:flatten##1%
{%
    \if 0\expandafter\XINT:NE:hastilde\detokenize{##1}~!\relax
```

```
4401          \expandafter\XINT:NE:hashash \detokenize{##1}#1!\relax 0%
4402     \else
4403       \expandafter\XINT:NE:x:flatten:p
4404     \fi \XINT:expr:flatten{##1}%
4405 }}\expandafter\XINT:NE:x:flatten\string#%
4406 \def\XINT:NE:x:flatten:p\XINT:expr:flatten #1%
4407 {%
4408     {{%
4409      \detokenize
4410      {%
4411          \expandafter\XINT:expr:flatten_checkempty
4412          \detokenize\expandafter{\expanded{#1}}$XINT_expr_caret%$
4413      }%
4414     }}%
4415 }%
4416 \def\XINT:NE:x:zip#1{%
4417 \def\XINT:NE:x:zip\XINT:expr:zip##1%
4418 {%
4419     \if 0\expandafter\XINT:NE:hastilde\detokenize{##1}~!\relax
4420          \expandafter\XINT:NE:hashash \detokenize{##1}#1!\relax 0%
4421     \else
4422       \expandafter\XINT:NE:x:zip:p
4423     \fi \XINT:expr:zip{##1}%
4424 }}\expandafter\XINT:NE:x:zip\string#%
4425 \def\XINT:NE:x:zip:p\XINT:expr:zip #1%
4426 {%
4427     \expandafter{%
4428      \detokenize
4429      {%
4430         \expanded\expandafter\XINT_zip_A\expanded{#1}\xint_bye\xint_bye
4431      }%
4432     }%
4433 }%
4434 \def\XINT:NE:x:mapwithin#1{%
4435 \def\XINT:NE:x:mapwithin\XINT:expr:mapwithin ##1##2%
4436 {%
4437     \if 0\expandafter\XINT:NE:hastilde\detokenize{##2}~!\relax
4438          \expandafter\XINT:NE:hashash \detokenize{##2}#1!\relax 0%
4439     \else
4440       \expandafter\XINT:NE:x:mapwithin:p
4441     \fi \XINT:expr:mapwithin {##1}{##2}%
4442 }}\expandafter\XINT:NE:x:mapwithin\string#%
4443 \def\XINT:NE:x:mapwithin:p \XINT:expr:mapwithin  #1#2%
4444 {%
4445     {{%
4446      \detokenize
4447      {%
```

Attention (2022/06/10) I do not remember why I left these two commented lines which docstrip will not remove, I hope this is not a forgotten left=over from some debugging session.

```
4448 %%       \expanded
4449 %%       {%
4450          \expandafter\XINT:expr:mapwithin_checkempty
```

```
          \expanded{\noexpand#1$XINT_expr_exclam\expandafter}%$
          \detokenize\expandafter{\expanded{#2}}$XINT_expr_caret%$
```

This is is the matching one.

```
%%       }%
      }%
    }}%
}%
\def\XINT:NE:x:ndmapx#1{%
\def\XINT:NE:x:ndmapx\XINT_allexpr_ndmapx_a ##1##2^%
{%
    \if 0\expandafter\XINT:NE:hastilde\detokenize{##2}~!\relax
         \expandafter\XINT:NE:hashash \detokenize{##2}#1!\relax 0%
    \else
      \expandafter\XINT:NE:x:ndmapx:p
    \fi \XINT_allexpr_ndmapx_a ##1##2^%
}}\expandafter\XINT:NE:x:ndmapx\string#%
\def\XINT:NE:x:ndmapx:p #1#2#3^\relax
{%
    \detokenize
    {%
      \expanded{%
      \expandafter#1\expandafter#2\expanded{#3}$XINT_expr_caret\relax %$
                }%
    }%
}%
```

Attention here that user function names may contain digits, so we don't use a \detokenize or ~ approach.

This syntax means that a function defined by \xintdeffunc never expands when used in another definition, so it can implement recursive definitions.

\XINT:NE:userefunc et al. added at 1.3e.

I added at \xintdefefunc, \xintdefiiefunc, \xintdeffloatefunc at 1.3e to on the contrary expand if possible (i.e. if used only with numeric arguments) in another definition.

The \XINTusefunc uses \expanded. Its ancestor \xintExpandArgs (xinttools 1.3) had some more primitive f-expansion technique.

```
\def\XINTusenoargfunc #1%
{%
    0\csname #1\endcsname
}%
\def\XINT:NE:usernoargfunc\csname #1\endcsname
{%
    ~romannumeral~XINTusenoargfunc{#1}%
}%
\def\XINTusefunc #1%
{%
    0\csname #1\expandafter\endcsname\expanded
}%
\def\XINT:NE:usefunc #1#2#3%
{%
    ~romannumeral~XINTusefunc{#1}{#3}\iffalse{{\fi}}%
}%
\def\XINTuseufunc #1%
{%
```

```
    \expanded\expandafter\XINT:expr:mapwithin\csname #1\expandafter\endcsname\expanded
}%
\def\XINT:NE:useufunc #1#2#3%
{%
    {{~expanded~XINTuseufunc{#1}{#3}}}%
}%
\def\XINT:NE:userfunc #1{%
\def\XINT:NE:userfunc ##1##2##3%
{%
    \if0\expandafter\XINT:NE:hastilde\detokenize{##3}~!\relax
        \expandafter\XINT:NE:hashash\detokenize{##3}#1!\relax 0%
      \expandafter\XINT:NE:userfunc_x
    \else
      \expandafter\XINT:NE:usefunc
    \fi {##1}{##2}{##3}%
}}\expandafter\XINT:NE:userfunc\string#%
\def\XINT:NE:userfunc_x #1#2#3{#2#3\iffalse{{\fi}}}%
\def\XINT:NE:userufunc #1{%
\def\XINT:NE:userufunc ##1##2##3%
{%
    \if0\expandafter\XINT:NE:hastilde\detokenize{##3}~!\relax
        \expandafter\XINT:NE:hashash\detokenize{##3}#1!\relax 0%
      \expandafter\XINT:NE:userufunc_x
    \else
      \expandafter\XINT:NE:useufunc
    \fi {##1}{##2}{##3}%
}}\expandafter\XINT:NE:userufunc\string#%
\def\XINT:NE:userufunc_x #1{\XINT:expr:mapwithin}%
\def\XINT:NE:macrofunc #1#2%
   {\expandafter\XINT:NE:macrofunc:a\string#1#2\empty&}%
\def\XINT:NE:macrofunc:a#1\csname #2\endcsname#3&%
   {{~XINTusemacrofunc{#1}{#2}{#3}}}%
\def\XINTusemacrofunc #1#2#3%
{%
    \romannumeral0\expandafter\xint_stop_atfirstofone
    \romannumeral0#1\csname #2\endcsname#3\relax
}%
```

32.30.6. \XINT_expr_redefinemacros

Completely refactored at 1.3.

Again refactored at 1.4. The availability of \expanded allows more powerful mechanisms and more importantly I better thought out the root problems caused by the handling of list operations in this context and this helped simplify considerably the code.

```
\catcode`- 11 \catcode0 11 \catcode2 11 \catcode3 11
\def\XINT_expr_redefinemacros {%
  \let\XINT:NEhook:unpack           \XINT:NE:unpack
  \let\XINT:NEhook:f:one:from:one        \XINT:NE:f:one:from:one
  \let\XINT:NEhook:f:one:from:one:direct \XINT:NE:f:one:from:one:direct
  \let\XINT:NEhook:f:one:from:two        \XINT:NE:f:one:from:two
  \let\XINT:NEhook:f:one:from:two:direct \XINT:NE:f:one:from:two:direct
  \let\XINT:NEhook:x:one:from:two        \XINT:NE:x:one:from:two
  \let\XINT:NEhook:f:one:and:opt:direct      \XINT:NE:f:one:and:opt:direct
```

```
  \let\XINT:NEhook:f:tacitzeroifone:direct   \XINT:NE:f:tacitzeroifone:direct
  \let\XINT:NEhook:f:iitacitzeroifone:direct \XINT:NE:f:iitacitzeroifone:direct
  \let\XINT:NEhook:x:listsel             \XINT:NE:x:listsel
  \let\XINT:NEhook:f:reverse             \XINT:NE:f:reverse
  \let\XINT:NEhook:f:from:delim:u        \XINT:NE:f:from:delim:u
  \let\XINT:NEhook:f:LFL            \XINT:NE:f:LFL
  \let\XINT:NEhook:r:check          \XINT:NE:r:check
  \let\XINT:NEhook:branch           \XINT:NE:branch
  \let\XINT:NEhook:seqx             \XINT:NE:seqx
  \let\XINT:NEhook:opx              \XINT:NE:opx
  \let\XINT:NEhook:rseq             \XINT:NE:rseq
  \let\XINT:NEhook:iter             \XINT:NE:iter
  \let\XINT:NEhook:rrseq            \XINT:NE:rrseq
  \let\XINT:NEhook:iterr            \XINT:NE:iterr
  \let\XINT:NEhook:x:toblist        \XINT:NE:x:toblist
  \let\XINT:NEhook:x:flatten        \XINT:NE:x:flatten
  \let\XINT:NEhook:x:zip            \XINT:NE:x:zip
  \let\XINT:NEhook:x:mapwithin      \XINT:NE:x:mapwithin
  \let\XINT:NEhook:x:ndmapx         \XINT:NE:x:ndmapx
  \let\XINT:NEhook:userfunc         \XINT:NE:userfunc
  \let\XINT:NEhook:userufunc        \XINT:NE:userufunc
  \let\XINT:NEhook:usernoargfunc    \XINT:NE:usernoargfunc
  \let\XINT:NEhook:macrofunc        \XINT:NE:macrofunc
  \def\XINTinRandomFloatSdigits{~XINTinRandomFloatSdigits }%
  \def\XINTinRandomFloatSixteen{~XINTinRandomFloatSixteen }%
  \def\calxFpRandomSdigits{~calxFpRandomSdigits }%
  \def\calxFpRandomSixteen{~calxFpRandomSixteen }%
  \def\calxintRandRange{~calxintRandRange }%
  \def\calxintRandRangeAtoB{~calxintRandRangeAtoB }%
  \def\xintRandBit{~xintRandBit }%
  \let\XINT_expr_exec_?  \XINT:NE:exec_?
  \let\XINT_expr_exec_?? \XINT:NE:exec_??
  \def\XINT_expr_op_?  {\XINT_expr_op__?{\XINT_expr_op_-xii\XINT_expr_oparen}}%
  \def\XINT_flexpr_op_?{\XINT_expr_op__?{\XINT_flexpr_op_-xii\XINT_flexpr_oparen}}%
  \def\XINT_iiexpr_op_?{\XINT_expr_op__?{\XINT_iiexpr_op_-xii\XINT_iiexpr_oparen}}%
```

calx: `\xintiieval` literals stay raw in compiled functions and macros: their meaning is detokenized and rescanned, where native `M` values would come back with a catcode 11 tag; the ii operations convert raw integers on entry. The fake operators literal as it is (

```
  \def\XINT_iiexpr_op_&&@ ##1##2]%
  {%
    \expandafter\XINT_expr_put_op_first
    \expanded{{{\CALX_rawE##1{\xint_zapspaces ##2 \xint_gobble_i}}}%
    \expandafter}\romannumeral`&&@\XINT_expr_getop
  }%
  \def\XINT_iiexpr_op_&&B ##1%
  {%
    \expandafter\XINT_expr_put_op_first
    \expanded{{{\xint_noxpd##1}}\expandafter}\romannumeral`&&@\XINT_expr_getop
  }%
  \let\XINT_iiexpr_op_&&C\XINT_flexpr_op_&&C
}%
\catcode`- 12 \catcode0 12 \catcode2 12 \catcode3 12
```

32.30.7. Still more mysterious

Modified at 2.0a1 (2026/11/01).

\XINT_NewExpr is the kernel underlying both \xintdeffunc and \xintNewExpr. Actually we declare here only \xintdeffloatfunc and \xintdefiifunc. All user level definitions, including all "NewExpr" variants are in xintexpr.sty. Nothing was changed at release 2.0a1. Only the split into calx.sty/xintexpr.sty kept here only \xintdeffloatfunc and \xintdefiifunc.

1.2c for \xintdeffunc, \xintdefiifunc, \xintdeffloatfunc.

At 1.3, NewFunc does not use anymore a comma delimited pattern for the arguments to the macro being defined.

At 1.4 we use \xintthebareeval, whose meaning now does not mean unlock from csname but firstofone to remove a level of braces This is involved in functioning of expr:userfunc and expr:userefunc

```
\def\XINT_NewFloatFunc{\XINT_NewExpr\xint_gobble_i\xintthebarefloateval\XINT_newfunc_clean}%
\def\XINT_NewIIFunc   {\XINT_NewExpr\xint_gobble_i\xintthebareiieval\XINT_newfunc_clean}%
\def\XINT_newfunc_clean #1>{}%
\ifdefined\notexpanded\let\XINT_newfunc_clean\xint_gobble_vi\fi%
```

1.2c adds optional logging. For this needed to pass to _NewExpr_a the macro name as parameter.

Up to and including 1.2c the definition was global. Starting with 1.2d it is done locally.

The \xintexprSafeCatcodes inserted here by \xintNewExpr is not paired with an \xintexprRestoreCatcodes, but this happens within a scope limiting group so does not matter. At 1.3c, \XINT_NewFunc et al. do not even execute the \xintexprSafeCatcodes, as it gets already done by \xintdeffunc prior to arriving here.

```
\def\XINT_NewExpr {\begingroup\XINT_NewExpr_a}%
\def\XINT_NewExpr_a #1#2#3#4#5[#6]%
{%
    \ifcase #6\relax
        \toks0 {\endgroup\XINT_global\def#4}%
    \or \toks0 {\endgroup\XINT_global\def#4##1}%
    \or \toks0 {\endgroup\XINT_global\def#4##1##2}%
    \or \toks0 {\endgroup\XINT_global\def#4##1##2##3}%
    \or \toks0 {\endgroup\XINT_global\def#4##1##2##3##4}%
    \or \toks0 {\endgroup\XINT_global\def#4##1##2##3##4##5}%
    \or \toks0 {\endgroup\XINT_global\def#4##1##2##3##4##5##6}%
    \or \toks0 {\endgroup\XINT_global\def#4##1##2##3##4##5##6##7}%
    \or \toks0 {\endgroup\XINT_global\def#4##1##2##3##4##5##6##7##8}%
    \or \toks0 {\endgroup\XINT_global\def#4##1##2##3##4##5##6##7##8##9}%
    \fi
    #1\xintexprSafeCatcodes
    \XINT_expr_redefinemacros
    \XINT_NewExpr_b #1#2#3#4%
}%
```

1.2d's \xintNewExpr makes a local definition. In earlier releases, the definition was global.

\the\toks0 inserts the \endgroup, but this will happen after \XINT_tmpa has already been expanded...

The %1 is \xint_firstofone for \xintNewExpr, \xint_gobble_i for \xintdeffunc.

Attention that at 1.4, there might be entire sub-xintexpressions embedded in detokenized form. They are re-tokenized and the main thing is that the parser should not mis-interpret catcode 11 characters as starting variable names. As some macros use : in their names, the retokenization must be done with : having catcode 11. To not break embedded non-evaluated sub-expressions, the \XINT_expr_getop was extended to intercept the : (alternative would have been to never inject any macro with : in its name... too late now). On the other hand the ! is not used in the macro

names potentially kept as is non expanded by the \xintNewExpr/\xintdeffunc process; it can thus be retokenized with catcode 12. But the «hooks» of seq(), iter(), etc... if deciding they can't evaluate immediately will inject a full sub-expression (possibly arbitrarily complicated) and append to it for its delayed expansion a catcode 11 ! character (as well as possibly catcode 3 ~ and ? and catcode 11 caret ^ and even catcode 7 &). The macros \XINT_expr_tilde etc... below serve for this injection (there are *two* successive \scantokens using different catcode regimes and these macros remain detokenized during the first pass!) and as consequence the final meaning may have characters such as ! or & present with both standard and special catcodes depending on where they are located. It may thus not be possible to (easily) retokenize the meaning as printed in the log file if \xintverbosetrue was issued.

If a defined function is used in another expression it would thus break things if its meaning was included pre-expanded ; a mechanism exists which keeps only the name of the macro associated to the function (this name may contain digits by the way), when the macro can not be immediately fully expanded. Thus its meaning (with its possibly funny catcodes) is not exposed. And this gives opportunity to pre-expand its arguments before actually expanding the macro.

There is a problem with xetex -8bit which will convert ^^@ possibly present in the meaning written to log if \xintverbosetrue into a null byte. It surprises at first, but perhaps because I am too much used to pdftex and luatex converting the zero byte perhaps produced meaning internally to ^^@ notation at the "output" stage.

I observed that ConTeXt will spit out such null byte "as is", as does xetex -8bit. This is a bit annoying because contrarily to xetex no option is needed, and such output lets some software consider the output file is a binary one (for example git diff).

```
\catcode`~ 3 \catcode`? 3
\def\XINT_expr_tilde{~}\def\XINT_expr_qmark{?}% catcode 3
\def\XINT_expr_caret{^}\def\XINT_expr_exclam{!}% catcode 11
\def\XINT_expr_tab{&}% catcode 7
\def\XINT_expr_null{&&@}%
```

Added at 1.4n (2025/09/05). Add \xintmeaning for matters of package test suite. At user level, a priori simply expands to \meaning. It gets redefined during execution of the test suite to keep logged expectations the same also with LMTX engine.

```
\ifdefined\xintmeaning\else\def\xintmeaning{\meaning}\fi
```

Added at 1.4n (2025/09/05). Add \XINT_expr_set_tilde to address LuaMetaTeX specifics. We won't need to restore because we will be in a group and \toks0 will bring the \endgroup.

When I observed that the whole \xintdeffunc thing was broken with ConTeXt due to what appeared to be some weirdness during expansion, I realized much to my surprise that the active tilde was not expanding inside \edef! I got lucky I could quickly find a then recent discussion precisely about this on tex.sx, from which I picked up the \amcode workaround. Such a trick is quite hidden in ConTeXt documentation, from which it is a long shot to deduce it behaves as it actually does.

Replacing about 130 occurrences of ~ in this file which are a core part of the mysterious dealings underpinning \xintdeffunc is not really an option. I don't have that many available ascii characters and I do not want to go into mass replacements and have to update unit tests sometimes checking internals. What a relief there is the \amcode way! (and seemingly no other way...).

```
\catcode`~ 13 \catcode`$ 11 %$
\def\XINT_NewExpr_set_tilde{\def~{$noexpand$}}%
\ifdefined\contextversion
  \ifdefined\amcode
    \def\XINT_NewExpr_set_tilde{\amcode`\~0\def~{$noexpand$}}%
  \else
    \xintMessage{xintexpr}{Error}{This ConTeXt is incompatible.}%
    \errhelp{xintexpr requires ConTeXt-LMTX to have its \string\amcode.}%
    \errmessage{The \noexpand\amcode primitive does not exist.}%
```

```
  \fi
\fi
\catcode`@ 14 \catcode`\% 6 \catcode`# 12
\def\XINT_NewExpr_b %1%2%3%4%5@
{@
    \def\XINT_tmpa %%1%%2%%3%%4%%5%%6%%7%%8%%9{%5}@
    \XINT_NewExpr_set_tilde
    \catcode`: 11 \catcode`_ 11 \catcode`\@ 11
    \catcode`# 12 \catcode`~ 13 \escapechar 126
    \endlinechar -1 \everyeof {\noexpand }@
    \edef\XINT_tmpb
    {\scantokens\expandafter{\romannumeral`&&@\expandafter
     %2\XINT_tmpa{#1}{#2}{#3}{#4}{#5}{#6}{#7}{#8}{#9}\relax}@
    }@
    \escapechar 92 \catcode`# 6 \catcode`$ 0 @ $
    \edef\XINT_tmpa %%1%%2%%3%%4%%5%%6%%7%%8%%9@
      {\scantokens\expandafter{\expandafter%3\meaning\XINT_tmpb}}@
    \the\toks0\expandafter
      {\XINT_tmpa{%%1}{%%2}{%%3}{%%4}{%%5}{%%6}{%%7}{%%8}{%%9}}@
    %1{\ifxintverbose
        \xintMessage{xintexpr}{Info}@
                    {\string%4\space now with @
                     \ifxintglobaldefs global \fi meaning \xintmeaning%4}@
       \fi}@
}@
\catcode`% 14
\XINTsetcatcodes % clean up to avoid surprises if something changes
```

32.30.8. \xintexprSafeCatcodes, \xintexprRestoreCatcodes

Modified at 1.3c (2018/06/17). Added \ifxintexprsafecatcodes to allow nesting

Modified at 1.4k (2022/05/18). The "allow nesting" from the 2018 comment was strange, because the behaviour, as correctly documented in user manual, was that in case of a series of \xintexprSafeCatcodes, the \xintexprRestoreCatcodes would set catcodes to what they were before the *first* sanitization. But as \xintdefvar and \xintdeffunc used such a pair this meant that they would incomprehensibly for user reset catcodes to what they were before a possible user \xintexprSafeCatcodes located before... very lame situation. Anyway. I finally fix at 1.4k that by removing the silly \ifxintexprsafecatcodes thing and replace it by some stack-like method, avoiding extra macros thanks to the help of \unexpanded.

Modified at 1.4m (2022/06/10). Use \protected rather than \unexpanded mechanism, for lisibility.

```
\protected\def\xintexprRestoreCatcodes{}%
\def\xintexprSafeCatcodes
{%
    \protected\edef\xintexprRestoreCatcodes{%
        \endlinechar=\the\endlinechar
        \catcode59=\the\catcode59   % ;
        \catcode34=\the\catcode34   % "
        \catcode63=\the\catcode63   % ?
        \catcode124=\the\catcode124 % |
        \catcode38=\the\catcode38   % &
        \catcode33=\the\catcode33   % !
        \catcode93=\the\catcode93   % ]
```

```
        \catcode91=\the\catcode91   % [
        \catcode94=\the\catcode94   % ^
        \catcode95=\the\catcode95   % _
        \catcode47=\the\catcode47   % /
        \catcode41=\the\catcode41   % )
        \catcode40=\the\catcode40   % (
        \catcode42=\the\catcode42   % *
        \catcode43=\the\catcode43   % +
        \catcode62=\the\catcode62   % >
        \catcode60=\the\catcode60   % <
        \catcode58=\the\catcode58   % :
        \catcode46=\the\catcode46   % .
        \catcode45=\the\catcode45   % -
        \catcode44=\the\catcode44   % ,
        \catcode61=\the\catcode61   % =
        \catcode96=\the\catcode96   % `
        \catcode32=\the\catcode32\relax  % space
        \protected\odef\xintexprRestoreCatcodes{\xintexprRestoreCatcodes}%
    }%
        \endlinechar=13 %
        \catcode59=12  % ;
        \catcode34=12  % "
        \catcode63=12  % ?
        \catcode124=12 % |
        \catcode38=4   % &
        \catcode33=12  % !
        \catcode93=12  % ]
        \catcode91=12  % [
        \catcode94=7   % ^
        \catcode95=8   % _
        \catcode47=12  % /
        \catcode41=12  % )
        \catcode40=12  % (
        \catcode42=12  % *
        \catcode43=12  % +
        \catcode62=12  % >
        \catcode60=12  % <
        \catcode58=12  % :
        \catcode46=12  % .
        \catcode45=12  % -
        \catcode44=12  % ,
        \catcode61=12  % =
        \catcode96=12  % `
        \catcode32=10  % space
}%
\let\XINT_tmpa\undefined \let\XINT_tmpb\undefined \let\XINT_tmpc\undefined
\let\XINT_tmpd\undefined \let\XINT_tmpe\undefined
```

32.31. The calx layer

calx: native float operators (wip). ATTENTION: the ! has catcode 12 here due to prior \XINTsetcatcodes.

ATTENTION: we do NOT set catcode of M (or N, or F) to 12 here. They only appear in \if tests,

and are needed in macro names.

Operator-level macros (letter-only names, frozen in user functions): two arguments (one for Sqr, Opp), precision \XINTdigits at run time.

Syntax (all f-expandable; arguments f-expanded, F, I or raw values; P = \XINTdigits unless given; results F or I values, without leading space):

```
\calxFpAdd{X}{Y}, \calxFpSub, \calxFpMul, \calxFpDiv -> round_P(X op Y)
  \calxFpMulExact{X}{Y}                -> X Y exact (X, Y finite F)
  \calxFpDivFloor{X}{Y}               -> floor(X'/Y') exact (X' = round_P(X))
  \calxFpMod{X}{Y}                    -> round_P(X' - Y' floor(X'/Y'))
  \calxFpDivMod{X}{Y}                 -> {floor(X'/Y')}{round_P(X' mod Y')}
  \calxFpInv{X}                       -> round_P(1/X)  (X exact)
  \calxFpE{X}{n}                      -> round_P(X) 10^n  (raw A[N])
  \calxFpSum{X1}...{Xn}^, \calxFpPrd  -> (((X1 op X2) op X3) ...), each op
                                      rounded (+(), *(); empty: 0, 1)
  \calxFpSqr{X}, \calxFpSqrtdigits{X}  -> round_P(X^2), round_P(sqrt X)
  \calxFpSqrt[P]{X}                    -> round_P(sqrt X)
  \calxFpOpp{X}, \calxFpAbs{X}         -> -X, |X|   (exact)
  \calxSgn{X}                          -> -1, 0 or 1   (any value)
  \calxFpLt{X}{Y}, \calxFpGt, \calxFpLtorEq, \calxFpGtorEq,
  \calxFpEq, \calxFpNotEq               -> 1 or 0 (exact comparison)
  \calxFpMaxof{X1}...{Xn}^, \calxFpMinof  -> the largest (smallest) Xi
  \calxFpTrunc{X}, \calxFpFloor{X}, \calxFpCeil{X}, \calxFpFrac{X}
                                       -> F (raw input: raw result)
  \calxFpITrunc{k}{X}, \calxFpIRound{k}{X}  -> trunc(X), round(X)
  \calxFpToNum{X}                      -> digits of trunc(X)
```

Internal helpers used by other files: \CALX_fcmp{X}{Y} (in \ifcase: 0 equal, 1 greater, 2 less, 3 unordered), \CALX_fonec (native 1), \CALX_fopp_a (sign flip), \CALX_ti_f (calxfp: trunc of an F value followed by \xint:), \CALX_fcnan (InvalidOperation signal of a comparison).

```
\def\calxFpAdd #1#2%
   {\romannumeral0\expandafter\CALX_fadd_in\expanded{#1\xint:#2\xint:}\the\XINTdigits.}%
\def\calxFpSub #1#2%
   {\romannumeral0\expandafter\CALX_fsub_in\expanded{#1\xint:#2\xint:}\the\XINTdigits.}%
\def\calxFpMul #1#2%
   {\romannumeral0\expandafter\CALX_fmul_in\expanded{#1\xint:#2\xint:}\the\XINTdigits.}%
\def\calxFpMulExact #1#2%
   {\romannumeral0\expandafter\CALX_fmulxs_in\expanded{#1\xint:#2\xint:}}%
\def\calxFpDiv #1#2%
   {\romannumeral0\expandafter\CALX_fdiv_in\expanded{#1\xint:#2\xint:}\the\XINTdigits.}%
\def\calxFpDivFloor #1#2%
   {\romannumeral0\expandafter\CALX_fdq_in\expanded{#1\xint:#2\xint:}\the\XINTdigits.\CALX_fdq_q}%
\def\calxFpMod #1#2%
   {\romannumeral0\expandafter\CALX_fdq_in\expanded{#1\xint:#2\xint:}\the\XINTdigits.\CALX_fdq_r}%
\def\calxFpDivMod #1#2%
   {\romannumeral0\expandafter\CALX_fdq_in\expanded{#1\xint:#2\xint:}\the\XINTdigits.\CALX_fdq_p}%
\def\calxFpInv #1%
   {\romannumeral0\expandafter\CALX_finv_in\romannumeral`&&@#1\xint:\the\XINTdigits.}%
\def\calxFpE #1%
   {\romannumeral0\expandafter\CALX_fe_in\romannumeral`&&@#1\xint:\the\XINTdigits.}%
```

calx: `+`(), `*`() of \xintfloateval: Burnol's 1.4p \XINT_sum/\XINT_prd loops (xintfrac), \xintadd/\xintmul replaced by \CALX_fsumop/\CALX_fprdop (rounded to P at each step, unlike 1.4p's exact \XINT_Sum); empty: 0[0], 1[0]; one item: returned as it is (f-expanded).

```
\def\calxFpSum {\romannumeral0\CALX_fpsum}%
```

```
\def\CALX_fpsum #1%
{%
    \xint_gob_til_^ #1\CALX_fpsum_empty ^%
    \expandafter\CALX_fpsum_loop\romannumeral`&&@#1\xint:
}%
\def\CALX_fpsum_empty ^#1\xint:{ 0[0]}%
\def\CALX_fpsum_loop #1\xint:#2%
{%
    \xint_gob_til_^ #2\CALX_fpsum_end ^%
    \expandafter\CALX_fpsum_loop
    \romannumeral0\CALX_fsumop{#1}{#2}\xint:
}%
\def\CALX_fpsum_end ^#1\CALX_fsumop #2#3\xint:{ #2}%
\def\calxFpPrd {\romannumeral0\CALX_fpprd}%
\def\CALX_fpprd #1%
{%
    \xint_gob_til_^ #1\CALX_fpprd_empty ^%
    \expandafter\CALX_fpprd_loop\romannumeral`&&@#1\xint:
}%
\def\CALX_fpprd_empty ^#1\xint:{ 1[0]}%
\def\CALX_fpprd_loop #1\xint:#2%
{%
    \xint_gob_til_^ #2\CALX_fpprd_end ^%
    \expandafter\CALX_fpprd_loop
    \romannumeral0\CALX_fprdop{#1}{#2}\xint:
}%
\def\CALX_fpprd_end ^#1\CALX_fprdop #2#3\xint:{ #2}%
\def\CALX_fsumop {\CALX_fadd\XINTdigits}%
\def\CALX_fprdop {\CALX_fmul\XINTdigits}%
\def\calxFpSqr #1%
   {\romannumeral0\expandafter\CALX_fsqr_in\romannumeral`&&@#1\xint:\the\XINTdigits.}%
\def\calxFpSqrtdigits #1%
   {\romannumeral0\expandafter\CALX_fsqrt_in\romannumeral`&&@#1\xint:\the\XINTdigits.}%
\def\calxFpSqrt [#1]#2%
   {\romannumeral0\expandafter\CALX_fsqrt_in\romannumeral`&&@#2\xint:\the\numexpr#1.}%
```

\calxFpToNum{X}: explicit digits of trunc(X), where X is a parser value (possibly still expandable). The ordinary entry takes one brace level. \calxListItemToNum{X} is for the legacy list selectors, whose items carry an additional brace level; it strips that level explicitly. Both paths enter the same conversion kernel, \CALX_fnum, with an already f-expanded input followed by \xint:.

```
\def\calxFpToNum #1%
   {\romannumeral0\expandafter\CALX_fnum\romannumeral`&&@#1\xint:}%
\def\calxListItemToNum #1%
   {\romannumeral0\expandafter\CALX_fnum\romannumeral`&&@\xint_firstofone#1%
    \xint:}%
```

\calxFpOpp: sign flip, the tag token kept; F+0.0., M+0.0. and I? unchanged. F, M and I: \CALX_fopp_a.

```
\def\calxFpOpp #1%
   {\romannumeral0\expandafter\CALX_fopp_e\romannumeral`&&@#1\xint:}%
```

raw input (strict A[N], A/B[N]): \CALX_fopp_x on the first token, a leading - removed, a leading 0 (zero) kept, else - prepended.

```
\def\CALX_fopp_e #1%
   {\if F#1\else\if I#1\else\if M#1\else\CALX_fopp_r\fi\fi\fi
    \CALX_fopp_a #1}%
\def\CALX_fopp_r #1\fi\fi\fi\CALX_fopp_a #2\xint:{\fi\fi\fi\CALX_fopp_x #2\xint:}%
\def\CALX_fopp_x #1#2\xint:%
   {\if-#1\xint_dothis{ #2}\fi\if0#1\xint_dothis{ #1#2}\fi\xint_orthat{ -#1#2}}%
\def\CALX_fopp_a #1#2#3\xint:%
{%
    \if#2-\xint_dothis{ #1+#3}\fi
    \if#2?\xint_dothis{ #1?}\fi
    \if0\CALX_fopp_b#3?\xint:\xint_dothis{ #1+#3}\fi
    \xint_orthat{ #1-#3}%
}%
\def\CALX_fopp_b #1#2\xint:{#1}%
```

Syntax (f-expandable; X a value of any parser, f-expanded; k a value with \CALX_mr digits, i.e. M or raw integer):

```
  \calxiiRound{k}{X}, \calxiiTrunc{k}{X} -> M (X 10^k rounded, truncated)
    \calxiiFloor{X}, \calxiiCeil{X}        -> M
    \calxCeilToNum{X}                      -> explicit digits of ceil(X)
```

round(x, k), trunc(x, k), floor(), ceil() of \xintiieval, the start value of n++. X 10^k as raw A/B[N + k] (M by \CALX_mtoraw_t, F by \CALX_ftoraw), then calxfp's \CALX_fintn with the kind \CALX_fintk_r, \CALX_fintk_t; floor, ceil: M (and I) values as they are, others \CALX_fintn with \CALX_fintk_f, \CALX_fintk_c; result \CALX_ntom. I values unchanged. \calxCeilToNum: \CALX_fint and \CALX_seq_k (I: \CALX_fnum, InvalidOperation, 0). \string on the tag of X after its f-expansion (values from compiled functions).

```
\def\calxiiRound #1#2%
   {\romannumeral0\expandafter\CALX_iirt_s\romannumeral`&&@#2\xint:{#1}%
    \CALX_fintk_r}%
\def\calxiiTrunc #1#2%
   {\romannumeral0\expandafter\CALX_iirt_s\romannumeral`&&@#2\xint:{#1}%
    \CALX_fintk_t}%
\def\calxiiFloor #1%
   {\romannumeral0\expandafter\CALX_iifl_s\romannumeral`&&@#1\xint:%
    \CALX_fintk_f}%
\def\calxiiCeil #1%
   {\romannumeral0\expandafter\CALX_iifl_s\romannumeral`&&@#1\xint:%
    \CALX_fintk_c}%
\def\CALX_iirt_s{\expandafter\CALX_iirt_a\string}%
\def\CALX_iirt_a #1%
{%
    \if I#1\xint_dothis{\CALX_iirt_i#1}\fi
    \if M#1\xint_dothis{\CALX_iirt_m#1}\fi
    \if F#1\xint_dothis{\CALX_iirt_f#1}\fi
    \xint_orthat{\CALX_iirt_r#1}%
}%
\def\CALX_iirt_i #1#2\xint:#3#4{ #1#2}%
\def\CALX_iirt_m #1#2\xint:#3%
   {\expandafter\CALX_iirt_b
    \romannumeral0\CALX_mtoraw_t#1#2\xint:[0]\xint:{#3}}%
\def\CALX_iirt_f #1#2\xint:#3%
   {\expandafter\CALX_iirt_b
    \romannumeral0\CALX_ftoraw{#1#2}[0]\xint:{#3}}%
```

```
\def\CALX_iirt_r #1#2\xint:#3{\CALX_iirt_b#1#2[0]\xint:{#3}}%
\def\CALX_iirt_b #1[#2]#3\xint:#4%
   {\expandafter\CALX_iirt_c\the\numexpr#2+\CALX_mr{#4}.{#1}}%
\def\CALX_iirt_c #1.#2#3%
   {\expandafter\CALX_ntom\romannumeral0\CALX_fintn#3#2[#1]\xint:}%
\def\CALX_iifl_s{\expandafter\CALX_iifl_a\string}%
\def\CALX_iifl_a #1%
{%
    \if I#1\xint_dothis{\CALX_iifl_copy#1}\fi
    \if M#1\xint_dothis{\CALX_iifl_copy#1}\fi
    \xint_orthat{\CALX_iifl_calc#1}%
}%
\def\CALX_iifl_copy #1#2\xint:#3{ #1#2}%
\def\CALX_iifl_calc #1#2\xint:#3%
   {\expandafter\CALX_ntom\romannumeral0\CALX_fintn#3#1#2\xint:}%
\def\calxCeilToNum #1%
   {\romannumeral0\expandafter\CALX_ctn_s\romannumeral`&&@#1\xint:}%
\def\CALX_ctn_s{\expandafter\CALX_ctn_a\string}%
\def\CALX_ctn_a #1#2\xint:%
{%
    \if I#1\xint_dothis{\CALX_fnum#1#2\xint:}\fi
    \xint_orthat{\expandafter\CALX_ctn_b\the\numexpr\expandafter\CALX_seq_k
      \romannumeral0\CALX_fint#1#2\xint:\CALX_fintk_c.}%
}%
\def\CALX_ctn_b #1.{ #1}%
```

calx: sign and zero tests on the values of the three parsers, by calxcore's \CALX_ifsgn and \CALX_ifnz (first tokens only, no xintfrac parsing). One macro token each, f-expandable, for the parser tables and the \xintNewExpr hooks:

```
  \calxSgn{X} -> -1, 0 or 1
    \calxIsZero{X}, \calxIsNotZero{X}   -> 1 or 0
    \calxifNotZero{X}{A}{B}             -> A (X <> 0) or B
    \calxifSgn{X}{A}{B}{C}              -> A, B or C (X < 0, = 0, > 0)
    \calxAND{X}{Y}, \calxOR{X}{Y}, \calxXOR{X}{Y}   -> 1 or 0
    \CALX_ANDof{X}{Y}...^, \CALX_ORof..., \CALX_XORof...   -> 1 or 0
```

I? is non-zero for the truth tests; \calxSgn and \calxifSgn signal InvalidOperation on it, then count it as 0. \calxAND, \calxOR, \calxXOR and the loops of the lists use \CALX_ifnz_a with \xint_firstoftwo, \xint_secondoftwo, which add no space (\CALX_xorof counts the non-zero items as signs in a \numexpr, as xint's \XINT_xorof).

```
\def\calxSgn #1%
   {\romannumeral0\expandafter\CALX_ifsgn_a\romannumeral`&&@#1\xint:{-1}{0}{1}}%
\def\calxIsZero #1%
   {\romannumeral0\expandafter\CALX_ifnz_a\romannumeral`&&@#1\xint:%
    \xint_stop_atfirstoftwo\xint_stop_atsecondoftwo{0}{1}}%
\def\calxIsNotZero #1%
   {\romannumeral0\expandafter\CALX_ifnz_a\romannumeral`&&@#1\xint:%
    \xint_stop_atfirstoftwo\xint_stop_atsecondoftwo{1}{0}}%
\def\calxifNotZero #1%
   {\romannumeral0\expandafter\CALX_ifnz_a\romannumeral`&&@#1\xint:%
    \xint_stop_atfirstoftwo\xint_stop_atsecondoftwo}%
\def\calxifSgn #1%
   {\romannumeral0\expandafter\CALX_ifsgn_a\romannumeral`&&@#1\xint:}%
\def\calxAND #1#2%
```

```
    {\romannumeral0\expandafter\CALX_ifnz_a\romannumeral`&&@#1\xint:%
     \xint_firstoftwo\xint_secondoftwo{\CALX_ifnz{#2}{1}{0}}{ 0}}%
\def\calxOR #1#2%
    {\romannumeral0\expandafter\CALX_ifnz_a\romannumeral`&&@#1\xint:%
     \xint_firstoftwo\xint_secondoftwo{ 1}{\CALX_ifnz{#2}{1}{0}}}%
\def\calxXOR #1#2%
    {\romannumeral0\expandafter\CALX_ifnz_a\romannumeral`&&@#1\xint:%
     \xint_firstoftwo\xint_secondoftwo{\CALX_ifnz{#2}{0}{1}}%
     {\CALX_ifnz{#2}{1}{0}}}%
\def\CALX_ANDof {\romannumeral0\CALX_andof}%
\def\CALX_andof #1%
    {\xint_gob_til_^ #1\CALX_andof_e ^%
     \expandafter\CALX_ifnz_a\romannumeral`&&@#1\xint:%
     \xint_firstoftwo\xint_secondoftwo\CALX_andof\CALX_andof_n}%
\def\CALX_andof_n #1^{ 0}%
\def\CALX_andof_e ^#1\CALX_andof_n{ 1}%
\def\CALX_ORof {\romannumeral0\CALX_orof}%
\def\CALX_orof #1%
    {\xint_gob_til_^ #1\CALX_orof_e ^%
     \expandafter\CALX_ifnz_a\romannumeral`&&@#1\xint:%
     \xint_firstoftwo\xint_secondoftwo\CALX_orof_y\CALX_orof}%
\def\CALX_orof_y #1^{ 1}%
\def\CALX_orof_e ^#1\CALX_orof{ 0}%
\def\CALX_XORof {\romannumeral0\CALX_xorof}%
\def\CALX_xorof {\if1\the\numexpr\CALX_xorof_a}%
\def\CALX_xorof_a #1%
    {\xint_gob_til_^ #1\CALX_xorof_e ^%
     \expandafter\CALX_ifnz_a\romannumeral`&&@#1\xint:%
     \xint_firstoftwo\xint_secondoftwo{-}{}\CALX_xorof_a}%
\def\CALX_xorof_e ^#1\CALX_xorof_a
    {1\relax\xint_afterfi{ 0}\else\xint_afterfi{ 1}\fi}%
```

calx: the same tests for \xintfloateval, with native F results (1, 0, -1): the values of \xintfloateval are F, M, I or raw with [N], never a bare integer. \CALX_fpone, \CALX_fpzero: <space>F+1.0.100000001!, <space>F+0.0. (written where F has catcode 12); \CALX_fpb <0 or 1>: the F value; the entries apply it to the digit of the general macros.

```
  \calxFpSgn{X} -> F -1, 0 or 1
    \calxFpIsZero{X}, \calxFpIsNotZero{X}  -> F 1 or 0
    \calxFpAND{X}{Y}, \calxFpOR, \calxFpXOR  -> F 1 or 0
    \calxFpANDof{X}{Y}...^, \calxFpORof, \calxFpXORof  -> F 1 or 0
    \calxFpEven{X}, \calxFpOdd, \calxFpIsOne, \calxFpIsInt -> F 1 or 0
    \calxFpLength{L}                        -> raw A[0]
    \calxFpiLogTendigits{X}, \calxFpiLogTen[P]{X}  -> F
```

ilog10(): X rounded to P digits (\CALX_fround; F, M, I, raw), then from the header F<s><D>.<e>.: 8e + D - 1 (\CALX_filt_f), as an F value (\CALX_ffromraw); zero: -"7FFF8000 as xint; I: Invalid-Operation, I?.

```
\catcode`F=12
\def\CALX_fpone{ F+1.0.100000001!}%
\def\CALX_fpzero{ F+0.0.}%
\def\CALX_fpsgn #1%
    {\romannumeral0\CALX_ifsgn{#1}{F-1.0.100000001!}{F+0.0.}{F+1.0.100000001!}}%
\catcode`F=11
\let\calxFpSgn\CALX_fpsgn
```

```
\def\CALX_fpb #1{\if1#1\expandafter\CALX_fpone\else\expandafter\CALX_fpzero\fi}%
\def\calxFpIsZero{\romannumeral0\expandafter\CALX_fpb\romannumeral`&&@\calxIsZero}%
\def\calxFpIsNotZero
   {\romannumeral0\expandafter\CALX_fpb\romannumeral`&&@\calxIsNotZero}%
\def\calxFpAND{\romannumeral0\expandafter\CALX_fpb\romannumeral`&&@\calxAND}%
\def\calxFpOR{\romannumeral0\expandafter\CALX_fpb\romannumeral`&&@\calxOR}%
\def\calxFpXOR{\romannumeral0\expandafter\CALX_fpb\romannumeral`&&@\calxXOR}%
\def\calxFpANDof{\romannumeral0\expandafter\CALX_fpb\romannumeral`&&@\CALX_ANDof}%
\def\calxFpORof{\romannumeral0\expandafter\CALX_fpb\romannumeral`&&@\CALX_ORof}%
\def\calxFpXORof{\romannumeral0\expandafter\CALX_fpb\romannumeral`&&@\CALX_XORof}%
```

even(), odd(): parity of the truncation of X (exact, as xintfrac's \xintNum). \CALX_fpar{X} -> <space>1 if odd, else <space>0. F: e > 0 or zero: even; e = 0: the lowest block; e < 0: the block of index -e (\CALX_nskip), if D > -8e, else the truncation is 0. M: e > 0 or zero: even, else the lowest block. Raw (also bare A, A/B) and I: calxfp's \CALX_fnum (I: InvalidOperation, 0), \CALX_nfromraw, the lowest block. isone(): \CALX_fcmp with 1 (exact; I: 0). isint(): X rounded to P digits (\CALX_fround, as xintfrac's \xintFloatIsInt), then zero or e >= 0: 1; e < 0: 1 iff D > -8e and the -e lowest blocks are zero (\CALX_fii_z); I: 0. ifint() (\calxFpifInt): the same test, first or second branch (\CALX_ifb). ifone() (\calxFpifOne): \CALX_fcmp with 1 (no rounding), I: second branch, no signal. ifint() of \xintiieval (calxint's \calxiiifInt): its values are integers, first branch; I: InvalidOperation, second branch (as \calxiiifOne).

```
\def\calxFpEven{\romannumeral0\expandafter\CALX_fpe\romannumeral0\CALX_fpar}%
\def\calxFpOdd{\romannumeral0\expandafter\CALX_fpb\romannumeral0\CALX_fpar}%
\def\CALX_fpe #1{\if1#1\expandafter\CALX_fpzero\else\expandafter\CALX_fpone\fi}%
\def\CALX_fpar #1{\expandafter\CALX_fpar_a\romannumeral`&&@#1\xint:}%
\def\CALX_fpar_a #1%
{%
    \if F#1\xint_dothis\CALX_fpar_f\fi
    \if M#1\xint_dothis\CALX_fpar_m\fi
    \xint_orthat\CALX_fpar_x #1%
}%
\def\CALX_fpar_f #1#2#3.#4.%
{%
    \ifnum#4>\xint_c_ \xint_dothis\CALX_fpar_zero\fi
    \ifnum#3=\xint_c_ \xint_dothis\CALX_fpar_zero\fi
    \ifnum#4=\xint_c_ \xint_dothis\CALX_fpar_g\fi
    \ifnum#3>\numexpr-\xint_c_viii*(#4)\relax
      \xint_dothis{\CALX_nskip\CALX_fpar_g -#4.}\fi
    \xint_orthat\CALX_fpar_zero
}%
\def\CALX_fpar_m #1#2#3.#4.%
{%
    \ifnum#3=\xint_c_ \xint_dothis\CALX_fpar_zero\fi
    \ifnum#4>\xint_c_ \xint_dothis\CALX_fpar_zero\fi
    \xint_orthat\CALX_fpar_g
}%
\def\CALX_fpar_zero #1\xint:{ 0}%
\def\CALX_fpar_g 1#1!#2\xint:{\ifodd#1 \xint_afterfi{ 1}\else\xint_afterfi{ 0}\fi}%
\def\CALX_fpar_x #1\xint:%
   {\expandafter\CALX_fpar_n\romannumeral0\CALX_nfromraw
    {\romannumeral0\CALX_fnum#1\xint:}\xint:}%
\catcode`N=12
\def\CALX_fpar_n N#1#2.#3\xint:%
```

```
4935    {\ifnum#2=\xint_c_ \xint_dothis{ 0}\fi\xint_orthat{\CALX_fpar_g#3\xint:}}%
4936 \catcode`N=11
4937 \def\calxFpIsOne #1%
4938    {\romannumeral0\ifcase\expandafter\CALX_fcmp_in\expanded{#1\xint:
4939           \CALX_fpone\xint:} \xint_afterfi\CALX_fpone
4940     \else\xint_afterfi\CALX_fpzero\fi}%
4941 \def\calxFpIsInt%
4942    {\romannumeral0\expandafter\CALX_fpb\romannumeral0\expandafter\CALX_fii
4943     \romannumeral0\CALX_fround{\XINTdigits}}%
4944 \def\CALX_fii #1{\if I#1\expandafter\CALX_fii_i\else\expandafter\CALX_fii_f\fi}%
4945 \def\CALX_fii_i #1\xint:{ 0}%
4946 \def\CALX_fii_f #1#2.#3.%
4947 {%
4948     \ifnum#2=\xint_c_ \xint_dothis\CALX_fii_yes\fi
4949     \ifnum#3<\xint_c_ \xint_dothis{\CALX_fii_n#2.#3.}\fi
4950     \xint_orthat\CALX_fii_yes
4951 }%
4952 \def\CALX_fii_n #1.#2.#3\xint:%
4953 {%
4954     \ifnum#1>\numexpr-\xint_c_viii*(#2)\relax
4955       \xint_dothis{\expandafter\CALX_fii_z\the\numexpr-#2.#3\xint:}\fi
4956     \xint_orthat{ 0}%
4957 }%
4958 \def\CALX_fii_z #1.1#2!%
4959 {%
4960     \ifnum#2>\xint_c_ \xint_dothis\CALX_fii_no\fi
4961     \ifnum#1=\xint_c_i \xint_dothis\CALX_fii_yes\fi
4962     \xint_orthat{\expandafter\CALX_fii_z\the\numexpr#1-\xint_c_i.}%
4963 }%
4964 \def\CALX_fii_no #1\xint:{ 0}%
4965 \def\CALX_fii_yes #1\xint:{ 1}%
4966 \def\calxFpifInt%
4967    {\romannumeral0\expandafter\CALX_ifb\romannumeral0\expandafter\CALX_fii
4968     \romannumeral0\CALX_fround{\XINTdigits}}%
4969 \def\CALX_ifb #1%
4970    {\if1#1\expandafter\xint_stop_atfirstoftwo\else
4971           \expandafter\xint_stop_atsecondoftwo\fi}%
4972 \def\calxFpifOne #1%
4973    {\romannumeral0\ifcase\expandafter\CALX_fcmp_in\expanded{#1\xint:
4974                   \CALX_fpone\xint:} %
4975     \expandafter\xint_stop_atfirstoftwo\else
4976     \expandafter\xint_stop_atsecondoftwo\fi}%
4977 \def\calxFpLength #1{\expanded{\xintLength{#1}[0]}}%
4978 \def\calxFpiLogTendigits%
4979    {\romannumeral0\expandafter\CALX_filt\romannumeral0\CALX_fround{\XINTdigits}}%
4980 \def\calxFpiLogTen [#1]%
4981    {\romannumeral0\expandafter\CALX_filt\romannumeral0\CALX_fround{#1}}%
4982 \def\CALX_filt #1%
4983    {\if I#1\expandafter\CALX_filt_i\else\expandafter\CALX_filt_f\fi #1}%
4984 \def\CALX_filt_i #1\xint:{\CALX_nsig{InvalidOperation}{ilog10 of I}{}}%
4985 \def\CALX_filt_f #1#2#3.#4.#5\xint:%
4986 {%
```

```
    \ifnum#3=\xint_c_ \expandafter\CALX_filt_z\fi
    \CALX_ffromraw{\the\numexpr\xint_c_viii*(#4)+#3-\xint_c_i[0]}%
}%
\def\CALX_filt_z #1#2{\CALX_ffromraw{-2147450880[0]}}%
```

calx: native comparisons and abs() of \xintfloateval.

\CALX_fcmp{X}{Y} (use in \ifcase): 0 if X = Y, 1 if X > Y, 2 if X < Y, 3 if unordered (an I?). Exact (no rounding). Two F values: the signs (0 for zero), then the orders 8e + D, then the sign of the exact difference \CALX_fsubx. I+, I- compare as infinities. Other inputs (\CALX_fc_x): each operand as an exact pair {p}{q} of F values (\CALX_fcq; q empty for 1): F itself; M: \CALX_fmx_c of calxfp (the digit count from the top block), zero F+0.0.; raw strict A[N] (or bare A, A/B as in compiled functions: [0] appended, ignored if [N] present): p by \CALX_ffromraw; A/B[N]: p of A[N], q of B[0] (B > 0). X = p/q, Y = r/s: p and r compared if q, s are empty, else p s and r q (\CALX_fmulx, exact), by \CALX_fc_n.

\calxFpLt, \calxFpGt, \calxFpLtorEq, \calxFpGtorEq: 1 or 0, an I? operand signals InvalidOperation (value 0), as Python's Decimal; \calxFpEq, \calxFpNotEq: I? is unequal to everything, no signal.

The operands come from the parsers: their tag may have catcode 11 (values stored in the meaning of a compiled function, read from a file). \CALX_fc_as and \CALX_fc_bs apply \string to the first token of each operand after its f-expansion: the copies handed to \CALX_fsubx, whose patterns contain F of catcode 12, then have a catcode 12 tag.

Expanded-operand entry: X\xint:Y\xint:, both input values are already f-expanded. The first token of each is normalized by \string before entering the common comparison kernel.

```
\def\CALX_fcmp #1%
   {\romannumeral0\expandafter\CALX_fc_as\romannumeral`&&@#1\xint:}%
\def\CALX_fcmp_in #1\xint:#2\xint:%
   {\expandafter\CALX_fc_ai\string#1\xint:#2\xint:}%
\def\CALX_fc_ai #1\xint:#2\xint:%
   {\expandafter\CALX_fc_bs\romannumeral`&&@#2\xint:#1\xint:}%
\def\CALX_fc_as{\expandafter\CALX_fc_a\string}%
\def\CALX_fc_a #1\xint:#2%
   {\expandafter\CALX_fc_bs\romannumeral`&&@#2\xint:#1\xint:}%
\def\CALX_fc_bs{\expandafter\CALX_fc_b\string}%
\def\CALX_fc_b #1#2\xint:#3%
{%
    \if I#3\xint_dothis{\CALX_fc_bi #1{#2}}\fi
    \if I#1\xint_dothis{\CALX_fc_bj #1{#2}#3}\fi
    \if F\if F#1#3\else Z\fi
      \xint_dothis{\CALX_fc_bn #1{#2}#3}\fi
    \xint_orthat{\CALX_fc_bx #1{#2}#3}%
}%
\def\CALX_fc_bi #1#2#3\xint:{\CALX_fc_i#3\xint:#1#2\xint:}%
\def\CALX_fc_bj #1#2#3#4\xint:{\CALX_fc_j#2\xint:#3#4\xint:}%
\def\CALX_fc_bn #1#2#3#4\xint:%
   {\CALX_fc_n{#3#4}{#1#2}#3#4\xint:#1#2\xint:}%
\def\CALX_fc_bx #1#2#3#4\xint:{\CALX_fc_x{#3#4}{#1#2}}%
\def\CALX_fc_x #1#2%
   {\expandafter\CALX_fc_xa\expanded{\CALX_fcq{#1}\CALX_fcq{#2}}}%
\def\CALX_fc_xa #1#2#3#4%
{%
    \if\relax\detokenize{#2#4}\relax
      \xint_dothis{\CALX_fc_n{#1}{#3}#1\xint:#3\xint:}\fi
    \xint_orthat{\expandafter\CALX_fc_xb\expanded
```

```
5021        {{\CALX_fcmul{#1}{#4}}{\CALX_fcmul{#3}{#2}}}}%
5022 }%
5023 \def\CALX_fc_xb #1#2{\CALX_fc_n{#1}{#2}#1\xint:#2\xint:}%
5024 \def\CALX_fcmul #1#2%
5025    {\if\relax\detokenize{#2}\relax\xint_dothis{#1}\fi
5026     \xint_orthat{\romannumeral0\CALX_fmulx{#1}{#2}}}%
5027 \def\CALX_fcq #1{\CALX_fcq_a #1\xint:}%
5028 \def\CALX_fcq_a #1#2\xint:%
5029 {%
5030     \if F#1\xint_dothis{{#1#2}{}}\fi
5031     \if M#1\xint_dothis{{\CALX_fcq_m #2\xint:}{}}\fi
5032     \xint_orthat{\CALX_fcq_r #1#2[0]\xint:}%
5033 }%
5034 \catcode`F=12 \catcode`N=12
5035 \def\CALX_fcq_m #1#2.#3.#4\xint:%
5036    {\ifnum#2=\xint_c_ \xint_dothis{F+0.0.}\fi
5037     \xint_orthat{\romannumeral0\CALX_fmx_c N#1#2.#4\xint:{#3}}}%
5038 \catcode`F=11 \catcode`N=11
5039 \def\CALX_fcq_r #1[#2]#3\xint:{\CALX_fcq_s #1/1/\xint:{#2}}%
5040 \def\CALX_fcq_s #1/#2/#3\xint:#4%
5041    {{\romannumeral0\CALX_ffromraw{#1[#4]}}%
5042     {\if\relax\detokenize{#3}\relax\else
5043        \romannumeral0\CALX_ffromraw{#2[0]}\fi}}%
```

X = I#1, Y any: I? unordered; I+ = I+, I+ > Y; I- = I-, I- < Y.

```
5044 \def\CALX_fc_i #1\xint:#2#3\xint:%
5045 {%
5046     \if ?#1\xint_dothis{ 3}\fi
5047     \if I#2\xint_dothis{\CALX_fc_ii #1#3\xint:}\fi
5048     \if -#1\xint_dothis{ 2}\fi
5049     \xint_orthat{ 1}%
5050 }%
5051 \def\CALX_fc_ii #1#2#3\xint:%
5052 {%
5053     \if ?#2\xint_dothis{ 3}\fi
5054     \if #1#2\xint_dothis{ 0}\fi
5055     \if -#1\xint_dothis{ 2}\fi
5056     \xint_orthat{ 1}%
5057 }%
```

Y = I#1, X not I.

```
5058 \def\CALX_fc_j #1\xint:#2\xint:%
5059 {%
5060     \if ?#1\xint_dothis{ 3}\fi
5061     \if -#1\xint_dothis{ 1}\fi
5062     \xint_orthat{ 2}%
5063 }%
```

{X}{Y}X\xint:Y\xint:, X = F<sX><DX>.<eX>.<blocks>, Y the same: the headers are read, Y's blocks are gobbled by \CALX_fc_o.

```
5064 \def\CALX_fc_n #1#2#3#4#5.#6.#7\xint:{\CALX_fc_m{#1}{#2}#4{#5}{#6}}%
5065 \def\CALX_fc_m #1#2#3#4#5#6#7#8.#9.%
5066 {%
5067     \expandafter\CALX_fc_o\the\numexpr
```

```
      \ifnum#4=\xint_c_ 0\else#31\fi-(\ifnum#8=\xint_c_ 0\else#71\fi)%
    \expandafter.\the\numexpr\xint_c_viii*(#5-#9)+#4-#8.#3{#1}{#2}%
}%
\def\CALX_fc_o #1.#2.#3#4#5#6\xint:%
{%
    \ifnum#1>\xint_c_ \xint_dothis{ 1}\fi
    \ifnum#1<\xint_c_ \xint_dothis{ 2}\fi
    \ifnum#2>\xint_c_ \xint_dothis{\CALX_fc_t#3{ 2}{ 1}}\fi
    \ifnum#2<\xint_c_ \xint_dothis{\CALX_fc_t#3{ 1}{ 2}}\fi
    \xint_orthat
      {\expandafter\CALX_fc_s\romannumeral0\CALX_fsubx{#4}{#5}\xint:}%
}%
\def\CALX_fc_t #1%
   {\if-#1\expandafter\xint_firstoftwo\else\expandafter\xint_secondoftwo\fi}%
\def\CALX_fc_s #1#2#3.#4\xint:%
   {\ifnum#3=\xint_c_ \xint_dothis{ 0}\fi\if-#2\xint_dothis{ 2}\fi
    \xint_orthat{ 1}}%
\def\calxFpLt #1#2%
   {\romannumeral0\ifcase\expandafter\CALX_fcmp_in\expanded{#1\xint:#2\xint:}
    \xint_afterfi\CALX_fpzero\or
    \xint_afterfi\CALX_fpzero\or\xint_afterfi\CALX_fpone\else\xint_afterfi\CALX_fpcnan\fi}%
\def\calxFpGt #1#2%
   {\romannumeral0\ifcase\expandafter\CALX_fcmp_in\expanded{#1\xint:#2\xint:}
    \xint_afterfi\CALX_fpzero\or
    \xint_afterfi\CALX_fpone\or\xint_afterfi\CALX_fpzero\else\xint_afterfi\CALX_fpcnan\fi}%
\def\calxFpLtorEq #1#2%
   {\romannumeral0\ifcase\expandafter\CALX_fcmp_in\expanded{#1\xint:#2\xint:}
    \xint_afterfi\CALX_fpone\or
    \xint_afterfi\CALX_fpzero\or\xint_afterfi\CALX_fpone\else\xint_afterfi\CALX_fpcnan\fi}%
\def\calxFpGtorEq #1#2%
   {\romannumeral0\ifcase\expandafter\CALX_fcmp_in\expanded{#1\xint:#2\xint:}
    \xint_afterfi\CALX_fpone\or
    \xint_afterfi\CALX_fpone\or\xint_afterfi\CALX_fpzero
    \else\xint_afterfi\CALX_fpcnan\fi}%
\def\calxFpEq #1#2%
   {\romannumeral0\ifcase\expandafter\CALX_fcmp_in\expanded{#1\xint:#2\xint:}
    \xint_afterfi\CALX_fpone\else
    \xint_afterfi\CALX_fpzero\fi}%
\def\calxFpNotEq #1#2%
   {\romannumeral0\ifcase\expandafter\CALX_fcmp_in\expanded{#1\xint:#2\xint:}
    \xint_afterfi\CALX_fpzero\else
    \xint_afterfi\CALX_fpone\fi}%
\def\CALX_fpcnan{\expandafter\CALX_fpb\romannumeral`&&@\CALX_fcnan}%
```

\calxFpAbs: F, M, I+, I-, I? or raw (strict A[N], A/B[N]: a leading - removed, \CALX_fabs_r); the tag token kept.

```
\def\calxFpAbs #1%
   {\romannumeral0\expandafter\CALX_fabs\romannumeral`&&@#1\xint:}%
\def\CALX_fabs #1%
   {\if F#1\expandafter\CALX_fabs_f\else\expandafter\CALX_fabs_x\fi #1}%
\def\CALX_fabs_f #1#2#3\xint:{ #1+#3}%
\def\CALX_fabs_x #1#2\xint:%
   {\if I#1\xint_dothis{\CALX_fabs_i#1#2}\fi
```

```
5118     \if M#1\xint_dothis{\CALX_fabs_f#1#2\xint:}\fi
5119     \xint_orthat{\CALX_fabs_r#1#2}}%
5120 \def\CALX_fabs_r #1%
5121    {\if-#1\expandafter\xint_firstoftwo\else\expandafter\xint_secondoftwo
5122     \fi{ }{ #1}}%
5123 \def\CALX_fabs_i #1#2%
5124    {\if ?#2\expandafter\xint_firstoftwo\else\expandafter\xint_secondoftwo
5125     \fi{ #1#2}{ #1+}}%
```

calx: max(), min() of \xintfloateval: \calxFpMaxof, \calxFpMinof {X1}{X2}...^: Burnol's 1.4p \XINT_maxof/\XINT_minof loops (xintfrac), \xintmax/\xintmin replaced by \CALX_fmax/\CALX_fmin (selection by \CALX_fcmp, no rounding); each item is f-expanded once and the value returned is expanded; empty list: 0/1[0]. An unordered comparison (I?) signals InvalidOperation and keeps the current value.

```
5126 \def\calxFpMaxof {\romannumeral0\CALX_maxof}%
5127 \def\CALX_maxof#1%
5128 {%
5129     \xint_gob_til_^ #1\CALX_maxof_empty ^%
5130     \expandafter\CALX_maxof_loop\romannumeral`&&@#1\xint:
5131 }%
5132 \def\CALX_maxof_empty ^#1\xint:{ 0/1[0]}%
5133 \def\CALX_maxof_loop #1\xint:#2%
5134 {%
5135     \xint_gob_til_^ #2\CALX_maxof_e ^%
5136     \expandafter\CALX_maxof_loop\romannumeral0\CALX_fmax{#1}{#2}\xint:
5137 }%
5138 \def\CALX_maxof_e ^#1\CALX_fmax #2#3\xint:{ #2}%
5139 \def\calxFpMinof {\romannumeral0\CALX_minof}%
5140 \def\CALX_minof#1%
5141 {%
5142     \xint_gob_til_^ #1\CALX_minof_empty ^%
5143     \expandafter\CALX_minof_loop\romannumeral`&&@#1\xint:
5144 }%
5145 \def\CALX_minof_empty ^#1\xint:{ 0/1[0]}%
5146 \def\CALX_minof_loop #1\xint:#2%
5147 {%
5148     \xint_gob_til_^ #2\CALX_minof_e ^%
5149     \expandafter\CALX_minof_loop\romannumeral0\CALX_fmin{#1}{#2}\xint:
5150 }%
5151 \def\CALX_minof_e ^#1\CALX_fmin #2#3\xint:{ #2}%
5152 \def\CALX_fmax #1#2{\expandafter\CALX_fmax_a\expandafter1\romannumeral`&&@#2\xint:{#1}}%
5153 \def\CALX_fmin #1#2{\expandafter\CALX_fmax_a\expandafter2\romannumeral`&&@#2\xint:{#1}}%
5154 \def\CALX_fmax_a #1#2\xint:#3%
5155    {\expandafter\CALX_mx_d\expandafter#1\romannumeral`&&@\CALX_fcmp{#2}{#3}.{#3}{#2}}%
5156 \def\CALX_mx_d #1#2.#3#4%
5157 {%
5158     \if 3#2\xint_dothis{\CALX_mx_sig{#3}}\fi
5159     \if #1#2\xint_dothis{ #4}\fi
5160     \xint_orthat{ #3}%
5161 }%
5162 \def\CALX_mx_sig #1%
5163    {\XINT_signalcondition{InvalidOperation}{max/min with I?}{}{ #1}}%
```

calx: floor(), ceil(), trunc(), round() (one argument) and frac() of \xintfloateval, exact (no

rounding to P). I+, I-, I? unchanged; frac(I+-) = 0.

F values: trunc by \CALX_ti_f (calxfp). floor, ceil: trunc, then -1 (+1) by \CALX_fsubx (\CALX_faddx) if the result differs from X and X < 0 (X > 0). round: trunc of X + 1/2 (X - 1/2 if X < 0). frac: X - trunc(X).

Raw values (A, A[N], A/B, A/B[N]) and M values: calxfp's \CALX_fintn with the kind macro (\CALX_fintk_f floor, \CALX_fintk_c ceil, \CALX_fintk_r round, \CALX_fintk_t trunc), result \CALX_fx_c 0.N. frac (\CALX_fl_kq) of a raw value: B = 1, the exact F value of A[N] goes to the F path; B > 1, the raw R/B[min(N,0)], with R the remainder of the truncated division (\CALX_nmodtrunc) of A 10^max(N,0) by B 10^max(-N,0); M values: 0. The kind macros, and \CALX_fl_kq for frac, also serve as markers (\ifx) in the F path.

The entries apply \string to the first token of X after its f-expansion (\CALX_ti_s, \CALX_fl_s): X is handed on to \CALX_fsubx, \CALX_faddx and \CALX_fcmp, and its tag may have catcode 11 when it comes from the meaning of a compiled function or from a file.

```
\edef\CALX_fonec{\romannumeral0\CALX_ffromraw{1[0]}}%
\edef\CALX_fhalfc{\romannumeral0\CALX_ffromraw{5[-1]}}%
\def\CALX_fl_kq {q}%
\def\calxFpTrunc #1%
   {\romannumeral0\expandafter\CALX_ti_s\romannumeral`&&@#1\xint:}%
\def\calxFpToM #1%
   {\romannumeral0\expandafter\CALX_ftom_a\romannumeral`&&@#1\xint:}%
\def\CALX_ti_s{\expandafter\CALX_ti_a\string}%
\def\calxFpITrunc #1{\calxFpTrunc}%
\def\CALX_ti_a #1%
   {\if F#1\expandafter\CALX_ti_f\else\expandafter\CALX_ti_x\fi #1}%
\def\CALX_ti_x #1%
   {\if I#1\expandafter\CALX_ti_xi\else\expandafter\CALX_ti_xn\fi #1}%
\def\CALX_ti_xi #1#2\xint:{ #1#2}%
\def\CALX_ti_xn #1#2\xint:%
   {\expandafter\CALX_fx_c\expandafter0\expandafter.%
    \romannumeral0\CALX_fintn\CALX_fintk_t#1#2\xint:\xint:}%
```

floor, ceil, round, frac: the trunc of F values, then a correction; the last token is the kind macro.

```
\def\calxFpFloor #1%
   {\romannumeral0\expandafter\CALX_fl_s\romannumeral`&&@#1\xint:%
    \CALX_fintk_f}%
\def\calxFpCeil #1%
   {\romannumeral0\expandafter\CALX_fl_s\romannumeral`&&@#1\xint:%
    \CALX_fintk_c}%
\def\calxFpIRound #1#2%
   {\romannumeral0\expandafter\CALX_fl_s\romannumeral`&&@#2\xint:%
    \CALX_fintk_r}%
\def\calxFpFrac #1%
   {\romannumeral0\expandafter\CALX_fl_s\romannumeral`&&@#1\xint:\CALX_fl_kq}%
\def\CALX_fl_s{\expandafter\CALX_fl_a\string}%
\def\CALX_fl_a #1%
   {\if F#1\expandafter\CALX_fl_f\else\expandafter\CALX_fl_x\fi #1}%
\def\CALX_fl_x #1#2\xint:#3%
{%
    \if I#1\xint_dothis{\CALX_fl_i#3#1#2}\fi
    \ifx#3\CALX_fl_kq\xint_dothis{\CALX_fl_fr #1#2\xint:}\fi
    \xint_orthat{\expandafter\CALX_fx_c\expandafter0\expandafter.%
      \romannumeral0\CALX_fintn#3#1#2\xint:\xint:}%
```

```
}%
\def\CALX_fl_fr #1#2\xint:%
   {\if M#1\xint_dothis{\CALX_ffr_a 0[0]\xint:}\fi
    \xint_orthat{\CALX_fl_fa #1#2[0]\xint:}}%
\def\CALX_fl_fa #1[#2]#3\xint:{\CALX_fl_fb #1/1/\xint:{#2}}%
\def\CALX_fl_fb #1/#2/#3\xint:{\CALX_fl_fd #2\xint:{#1}{#2}}%
\def\CALX_fl_fd #1#2\xint:#3#4#5%
   {\if1#1\if\relax\detokenize{#2}\relax\CALX_fl_fj\fi\fi
    \expandafter\CALX_fl_fc\romannumeral0\CALX_nmodtrunc
      {\CALX_nshl{#5}{\CALX_nfromraw{#3}}}%
      {\CALX_nshl{-#5}{\CALX_nfromraw{#4}}}\xint:{#4}{#5}{#3}}%
\def\CALX_fl_fj #1\xint:#2#3#4%
   {\fi\fi\expandafter\CALX_fl_f\romannumeral0\CALX_ffr_a#4[#3]\xint:\xint:%
    \CALX_fl_kq}%
\def\CALX_fl_fc #1\xint:#2#3#4%
   {\expanded{ \romannumeral0\CALX_ntoraw{#1}/#2%
    [\ifnum#3<\xint_c_ #3\else0\fi]}}%
\def\CALX_fl_i #1%
   {\ifx#1\CALX_fl_kq\expandafter\CALX_fl_ii\else\expandafter\space\fi}%
\def\CALX_fl_ii #1#2{\if ?#2\xint_dothis{ #1#2}\fi\xint_orthat\CALX_fpzero}%
\def\CALX_fl_f #1#2#3\xint:#4%
{%
    \ifx#4\CALX_fintk_r\xint_dothis{\CALX_fl_r#2{#1#2#3}}\fi
    \ifx#4\CALX_fl_kq\xint_dothis
      {\expandafter\CALX_fl_q\romannumeral0\CALX_ti_a#1#2#3\xint:\xint:%
       {#1#2#3}}\fi
    \xint_orthat
      {\expandafter\CALX_fl_b\romannumeral0\CALX_ti_a#1#2#3\xint:\xint:%
       #2#4{#1#2#3}}%
}%
\def\CALX_fl_q #1\xint:#2{\CALX_fsubx{#2}{#1}}%
\def\CALX_fl_r #1#2%
   {\expandafter\CALX_ti_a\romannumeral0%
    \if-#1\expandafter\CALX_fsubx\else\expandafter\CALX_faddx\fi
    {#2}\CALX_fhalfc\xint:}%
% floor: X < 0 and trunc(X) != X: trunc - 1; ceil: X > 0, != X: + 1.
\def\CALX_fl_b #1\xint:#2#3#4%
{%
    \ifx#3\CALX_fintk_f\if-#2\CALX_fl_c\fi\else\if-#2\else\CALX_fl_c\fi\fi
    \CALX_fl_e{#1}{#4}#3%
}%
\def\CALX_fl_e #1#2#3{ #1}%
\def\CALX_fl_c #1\CALX_fl_e #2#3#4%
{%
    \fi\fi
    \ifcase\CALX_fcmp{#2}{#3} \xint_afterfi{ #2}%
    \else\xint_afterfi{\ifx#4\CALX_fintk_f\expandafter\CALX_fsubx\else
                       \expandafter\CALX_faddx\fi{#2}\CALX_fonec}\fi
}%
```

32.32. Floating point powers with integer exponent

\calxFpPow{A}{B} -> F or I value

The exponent is inspected before any rounding. An integer in the \numexpr range uses arithmetic precision, not the installed log precision. Noninteger exponents, general fractions and large exponents use the scientific power. For raw fractions the test is format based; there is no divisibility test. When xintexpr is loaded it supplies the legacy normalization of raw fractions and of inputs not in raw format.

```
\def\calxFpPow #1#2%
   {\romannumeral0\expandafter\CALX_fip_dispatch\romannumeral`&&@#2\xint:{#1}}%
\let\XINTinFloatSciPow\calxFpPow
\def\CALX_fip_dispatch #1\xint:#2%
   {\expandafter\CALX_fip_pick\romannumeral0\expandafter\CALX_fip_tag
      \string#1\xint:.{#2}{#1}}%
\def\CALX_fip_tag #1%
{%
    \if I#1\xint_dothis\CALX_fip_unknown\fi
    \if F#1\xint_dothis\CALX_fip_native\fi
    \if M#1\xint_dothis\CALX_fip_integer\fi
    \xint_orthat\CALX_fip_raw #1%
}%
\def\CALX_fip_unknown #1\xint:{ \relax}%
```

The classifier returns a signed decimal integer (or 0), or \relax. The native header bounds the integer digit count before conversion. \CALX_frez strips zero limbs and decimal trailing zeros without rounding.

```
\def\CALX_fip_native #1#2#3.#4.#5\xint:%
{%
    \ifnum#3=0 \xint_dothis{ 0}\fi
    \ifnum#4>1 \xint_dothis{ \relax}\fi
    \ifnum\numexpr#3+8*#4\relax>10 \xint_dothis{ \relax}\fi
    \ifnum\numexpr#3+8*#4\relax<1 \xint_dothis{ \relax}\fi
    \xint_orthat{\expandafter\CALX_fip_norm\romannumeral0%
        \CALX_frez #1#2#3.#4.#5\xint:\xint:}%
}%
\def\CALX_fip_integer #1#2#3.#4.#5\xint:%
{%
    \ifnum#3=0 \xint_dothis{ 0}\fi
    \ifnum#4>1 \xint_dothis{ \relax}\fi
    \ifnum\numexpr#3+#4\relax>2 \xint_dothis{ \relax}\fi
    \xint_orthat{\expandafter\CALX_fip_num\romannumeral0%
        \CALX_mtoraw_t #1#2#3.#4.#5\xint:\xint:{0}}%
}%
\def\CALX_fip_raw #1\xint:{\CALX_fip_rawa #1[0]\xint:{#1}}%
\def\CALX_fip_rawa #1[#2]#3\xint:#4%
{%
    \if\relax\detokenize{#3}\relax \xint_dothis{\CALX_fip_bare{#4}}\fi
    \xint_orthat{\CALX_fip_rawb #1/1/\xint:{#2}{#4}}%
}%
\def\CALX_fip_bare #1{\CALX_fip_rawb #1/1/\xint:{0}{#1}}%
\def\CALX_fip_rawb #1/#2/#3\xint:#4#5%
   {\CALX_fip_rawc #2\xint:{#1}{#4}{#5}}%
\def\CALX_fip_rawc #1#2\xint:#3#4#5%
{%
    \ifnum\if1#1\if\relax\detokenize{#2}\relax1\else0\fi\else0\fi=1
        \xint_dothis{\CALX_fip_decimal{#3}{#4}}\fi
```

```
5294     \xint_orthat{\CALX_fip_frac{#5}}%
5295 }%
5296 \def\CALX_fip_frac #1{ \relax}%
5297 \def\CALX_fip_decimal #1#2%
5298    {\expandafter\CALX_fip_dec\the\numexpr#2\relax.{#1}}%
5299 \def\CALX_fip_dec #1.#2%
5300 {%
5301     \ifnum#1<0 \xint_dothis{\CALX_fip_decneg{#2}{#1}}\fi
5302     \xint_orthat{\CALX_fip_num #2\xint:{#1}}%
5303 }%
5304 \def\CALX_fip_decneg #1#2%
5305 {%
5306     \ifnum#2<-\xintLength{#1} \xint_dothis{ \relax}\fi
5307     \xint_orthat{\expandafter\CALX_fip_norm\romannumeral0\expandafter
5308       \CALX_frez\romannumeral0\CALX_ffr_a #1[#2]\xint:\xint:\xint:}%
5309 }%
5310 \def\CALX_fip_norm #1/#2[#3]\xint:{\CALX_fip_norma #2\xint:{#1}{#3}}%
5311 \def\CALX_fip_norma #1#2\xint:#3#4%
5312 {%
5313     \ifnum\if1#1\if\relax\detokenize{#2}\relax1\else0\fi\else0\fi=1
5314         \xint_dothis{\CALX_fip_num #3\xint:{#4}}\fi
5315     \xint_orthat{ \relax}%
5316 }%
5317 \def\CALX_fip_num #1#2\xint:#3%
5318 {%
5319     \if0#1\xint_dothis{ 0}\fi
5320     \if-#1\xint_dothis{\CALX_fip_mag-#2\xint:{#3}}\fi
5321     \xint_orthat{\CALX_fip_mag+#1#2\xint:{#3}}%
5322 }%
5323 \def\CALX_fip_mag #1#2\xint:#3%
5324 {%
5325     \ifnum#3<0 \xint_dothis{ \relax}\fi
5326     \ifnum#3>9 \xint_dothis{ \relax}\fi
5327     \xint_orthat{\expandafter\CALX_fip_len\the\numexpr\xintLength{#2}+#3.%
5328         #1#2\xint:{#3}}%
5329 }%
5330 \def\CALX_fip_len #1.#2#3\xint:#4%
5331 {%
5332     \ifnum#1>10 \xint_dothis{ \relax}\fi
5333     \xint_orthat{\expandafter\CALX_fip_bound\expanded{#2#3\xintReplicate{#4}{0}}%
5334         \xint:{#1}}%
5335 }%
5336 \def\CALX_fip_bound #1#2#3\xint:#4%
5337 {%
5338     \ifnum\ifnum#4=10 #2\else0\fi>2 \xint_dothis{ \relax}\fi
5339     \ifnum\ifnum#4=10 \ifnum#2=2 #3\else0\fi\else0\fi>147483647
5340         \xint_dothis{ \relax}\fi
5341     \xint_orthat{ #1#2#3}%
5342 }%
```

Round the base to arithmetic precision P. Treat 0, 1, -1 and 2 as special exponents. The remaining powers use P+L+3 digits, L the decimal length of the absolute exponent. A conservative bound on the decimal order of all intermediate products protects the integer arithmetic used in the float

kernels near TeX's exponent limit. If that bound is too large, try the near-unity bound below before falling back to the scientific power.

```
\def\CALX_fip_pick #1#2.#3#4%
{%
    \ifx\relax#1\xint_dothis{\calx_lSciPow{#3}{#4}}\fi
    \if0#1\xint_dothis{ \CALX_fonec}\fi
    \xint_orthat{\expandafter\CALX_fip_base\romannumeral0%
        \CALX_fround\XINTdigits{#3}#1#2.{#4}}%
}%
\def\CALX_fip_base #1%
   {\if I#1\expandafter\CALX_fip_special\else\expandafter\CALX_fip_finite\fi #1}%
\def\CALX_fip_special #1#2\xint:#3#4.#5%
{%
    \if ?#2\xint_dothis{ #1?}\fi
    \if-#3\xint_dothis{\CALX_fpzero}\fi
    \xint_orthat{\expanded{ #1\ifodd#4 #2\else+\fi}}%
}%
\def\CALX_fip_finite #1#2#3.#4.#5\xint:#6#7.#8%
{%
    \ifnum#3=0 \xint_dothis{\CALX_fip_zero#6}\fi
    \ifnum\if+#6#7\else0\fi=1 \xint_dothis{ #1#2#3.#4.#5}\fi
    \ifnum#4>125000000 \xint_dothis{\calx_lSciPow{#1#2#3.#4.#5}{#8}}\fi
    \ifnum#4<-125000000 \xint_dothis{\calx_lSciPow{#1#2#3.#4.#5}{#8}}\fi
    \xint_orthat{\expandafter\CALX_fip_range\the\numexpr#3+8*#4-1.%
       #6#7.{#1#2#3.#4.#5}{#8}}%
}%
\def\CALX_fip_zero #1%
{%
    \if-#1\xint_dothis{\CALX_nsig{DivisionByZero}{0 ** b, b < 0}{+}}\fi
    \xint_orthat{\CALX_fpzero}%
}%
\def\CALX_fip_range #1.#2#3.#4#5%
{%
    \ifnum\ifnum#1<0 -#1\else\numexpr#1+1\relax\fi>%
        \numexpr999900000/#3-1\relax
        \xint_dothis{\CALX_fip_near #4\xint:#2#3.{#5}}\fi
    \xint_orthat{\CALX_fip_compute #2#3.{#4}}%
}%
```

The decimal-order bound can be too conservative for large powers of numbers near unity. In the rare fallback case, retain binary powering if 0.4 <= |A| <= 2.5: even for the largest \numexpr exponent the absolute logarithm to base ten of every intermediate power is less than 855e6.

```
\def\CALX_fip_near #1#2#3\xint:#4#5.#6%
{%
    \ifnum\CALX_fcmp{#1+#3}{4[-1]}=2
        \xint_dothis{\calx_lSciPow{#1#2#3}{#6}}\fi
    \ifnum\CALX_fcmp{#1+#3}{25[-1]}=1
        \xint_dothis{\calx_lSciPow{#1#2#3}{#6}}\fi
    \xint_orthat{\CALX_fip_compute #4#5.{#1#2#3}}%
}%
\def\CALX_fip_compute #1#2.#3%
{%
    \ifnum#2=1 \xint_dothis{\CALX_finv\XINTdigits{#3}}\fi
```

```
    \ifnum\if+#1#2\else0\fi=2 \xint_dothis{\CALX_fsqru\XINTdigits{#3}}\fi
    \xint_orthat{\expandafter\CALX_fip_finish\romannumeral0%
        \expandafter\CALX_fip_start\expanded{#2.%
          \the\numexpr\XINTdigits+\xintLength{#2}+3.{#3}}\xint:#1}%
}%
```

Left-to-right binary exponentiation. Read the low bits of the integer exponent to build the sequence of step macros in reverse order. The leading bit initializes the accumulator to the base, without a product by 1. For an odd bit, multiply the squared accumulator by the original P-digit base, rather than by a Q-digit power. Each square or product is rounded at Q = P+L+3 digits; the operands are already within Q digits. Each intermediate relative rounding error is bounded by 0.5*10^(1-Q), and at most n-1 such factors occur in the error of the positive power (counting repeated squarings with multiplicity). For a negative exponent, take the reciprocal of the Q-digit result directly at P, without first rounding its denominator.

```
\def\CALX_fip_start #1.#2.#3%
   {\CALX_fip_bits #1.{\CALX_fip_end}#2.{#3}{#3}}%
\def\CALX_fip_bits #1.#2%
{%
    \ifnum#1=1 \xint_dothis{#2}\fi
    \ifodd#1 \xint_dothis{\expandafter\CALX_fip_bits\the\numexpr(#1-1)/2.%
      {\CALX_fip_bitodd#2}}\fi
    \xint_orthat{\expandafter\CALX_fip_bits\the\numexpr#1/2.%
      {\CALX_fip_biteven#2}}%
}%
\def\CALX_fip_biteven #1\CALX_fip_end#2.#3#4%
   {\expandafter\CALX_fip_step\romannumeral0\CALX_fsqru{#2}{#4}\xint:%
      #1\CALX_fip_end#2.{#3}}%
\def\CALX_fip_bitodd #1\CALX_fip_end#2.#3#4%
   {\expandafter\CALX_fip_step\romannumeral0\CALX_fmulu{#2}{#3}%
      {\CALX_fsqru{#2}{#4}}\xint:#1\CALX_fip_end#2.{#3}}%
\def\CALX_fip_step #1\xint:#2\CALX_fip_end#3.#4%
   {#2\CALX_fip_end#3.{#4}{#1}}%
\def\CALX_fip_end #1.#2#3{ #3}%
\def\CALX_fip_finish #1\xint:#2%
{%
    \if-#2\xint_dothis{\CALX_finv\XINTdigits{#1}}\fi
    \xint_orthat{\CALX_fnround\XINTdigits{#1}}%
}%
```

32.33. Loading libraries

calx: calxtrig is loaded instead of xinttrig, calxlog (log function) instead of xintlog. The \xinteval hooks and the constants Pi, ... are defined at the end of calxtrig.tex (s50), the log family's in calxlog.tex. Due to \XINTsetcatcodes done a bit earlier the ! is catcode 12.

Formerly xintexpr loaded xintlog which loaded poormanlog which defines \@namedef. Backporting it here, it is a short name and is provided by LaTeX which will be used by vast majority of xint users.

```
\if1\ifx\@namedef\JFBUundefined1\else\ifx\@namedef\relax1\else0\fi\fi
  \def\@namedef#1{\expandafter\def\csname #1\endcsname}%
  \def\@nameuse#1{\csname #1\endcsname}%
\fi
```

If arithmetic precision exceeds the scientific limit at first loading, initialize both libraries at 100 digits. Restore arithmetic precision afterwards; in particular the trig constants must be

defined while the temporary precision is in effect.

```
\ifnum\XINTdigits>100
    \xintMessage{calx}{Info}{Scientific libraries initialized at 100 digits;
        arithmetic precision remains \the\XINTdigits.}%
    \let\CALX_scisavedigits\XINTdigits
    \mathchardef\XINTdigits=100
\fi
\input calxtrig.tex\relax
\input calxlog.tex\relax
\ifdefined\CALX_scisavedigits
    \let\XINTdigits\CALX_scisavedigits
    \let\CALX_scisavedigits\undefined
\fi
\CALX_restorecc\let\CALX_restorecc\undefined
```

32.34. \xintreloadcalxlog, \xintreloadcalxtrig, \xintreloadscilibs

Reconstruct the scientific libraries at the current arithmetic precision.

```
\def\xintreloadscilibs{\xintreloadcalxtrig\xintreloadcalxlog}%
```

32.35. \calxeval, \calxinteval

```
\let\calxeval\xintfloateval
\let\calxinteval\xintiieval
\XINTrestorecatcodesendinput%
```

---

33. Library calxlog implementation

Added at calx 2.0a1.

33.1 Catcodes, ε-TEX and reload detection . . . 849
33.2 Package identification . . . 850
33.3 Loading data tables . . . 851
33.4 \calxFpLog, \calxRawLog . . . 851
33.5 Natural logarithm, P1..P8 . . . 852
33.6 Natural logarithm, P9..P16 . . . 855
33.7 Natural logarithm, P17..P32 . . . 862
33.8 Natural logarithm, P33 and above . . . 870
33.9 \calxFpLogOnePlus, \calxRawLogOnePlus . . . 872
33.10 \calxFpLogTen, \calxRawLogTen . . . 876
33.11 \calxFpExp, \calxRawExp . . . 877
33.12 Exponential, P1..P8 . . . 881
33.13 Exponential, P17..P100 . . . 884
33.14 \calxFpPowTen, \calxRawPowTen . . . 896
33.15 \calxFpExpMinusOne, \calxRawExpMinusOne . . . 898
33.16 \calxFpPow, \calxRawPow . . . 901
33.17 \calxFpSinh, \calxFpCosh, \calxFpTanh, \calxFpCoth, \calxFpSech, \calxFpCsch, \calxFpAsinh, \calxFpAcosh, \calxFpAtanh, \calxFpAcoth, \calxFpAsech, \calxFpAcsch . . . 904
33.18 Aliases . . . 921
33.19 Reconstruction at the scientific precision . . . 921
33.20 \xintreloadcalxlog, \xintreloadxintlog . . . 926

33.1. Catcodes, ε-TEX and reload detection

The code of this subsection is copied over from the xint packages, with the package names changed.

```
\begingroup\catcode61\catcode48\catcode32=10\relax%
  \catcode13=5    % ^^M
  \endlinechar=13 %
  \catcode123=1   % {
  \catcode125=2   % }
  \catcode64=11   % @
  \catcode44=12   % ,
  \catcode46=12   % .
  \catcode58=12   % :
  \catcode94=7    % ^
  \catcode35=6    % #
  \catcode33=12   % !
  \def\empty{}\def\space{ }\newlinechar10
  \expandafter\let\expandafter\l\csname ver@calxlog.tex\endcsname
  \expandafter\let\expandafter\c\csname ver@calx.sty\endcsname
  \expandafter
    \ifx\csname PackageWarningNoLine\endcsname\relax
      \def\y#1#2{\immediate\write128{^^JPackage #1 Warning:^^J%
                  \space\space\space\space#2.^^J}}%
    \else
      \def\y#1#2{\PackageWarningNoLine{#1}{#2}}%
    \fi
  % attention that in this file \z by default does \endinput.
  \def\z{\endgroup\endinput}%
  \expandafter\ifx\csname numexpr\endcsname\relax
```

```
    \y{calx}{\numexpr not available, aborting input}%
  \else
```

BEWARE: on 2026/10/09 I hit a problem I knew about from years ago but forgot. It is known that TeX executes all macros on the line where \endinput is found. But imagine \endinput\input foo.tex\relax, then what will happen is that only the first line of foo.tex (if its EOL is not hidden by a comment char) is executed, so basically foo.tex is not read beyond that. I will put \endinput last in both situations, as is logical anyhow.

```
      \ifx\c\relax % calx.sty not yet loaded.
        \ifx\PackageWarningNoLine\relax
          \y{calx}{%
             Loading of calxlog.tex should happen via
             \string\input\space calx.sty\relax^^J\space\space\space\space
             Will try that now%
          }%
          \def\z{\endgroup
                 \input calx.sty\relax\endinput}%
        \else
          \y{calx}{%
             Bad move! Inputting calxlog.tex directly in a LaTeX\MessageBreak
             document is forbidden! Simply use \string\usepackage{calx}\MessageBreak
             in the preamble, as every obedient LaTeX user does!\MessageBreak
             Trying to recover now, hoping to be in the preamble%
          }%
          \def\z{\endgroup
                 \RequirePackage{calx}\endinput}%
        \fi
      \else % either calx.sty is trying to load us, or a second loading is
            % attempted on us by \input. Abort if the latter.
        \ifx\l\relax
            \def\z{\endgroup}%
        \else
            \y{calx}{calxlog.tex already loaded! Aborting this input}%
        \fi
      \fi
  \fi
\z%
```

We arrive here only after a \XINTsetcatcodes which is done (near its tail) in calx.sty. In particular ! has its normal catcode 12. But attention: the \XINTsetcatcode does nothing with the letters F, I, N so we have to be careful whenever we set them to be of catcode "other" (=12).

There appears to be a problem with miniltx which provides \ProvidesFile but does not let it define \ver@calxlog.tex. So we define it always.

33.2. Package identification

```
\ifdefined\ProvidesFile
  \ProvidesFile{calxlog.tex}%
  [2026/11/26 v2.0a2 A calx component (JFB)]%
\fi
\expandafter\def\csname ver@calxlog.tex\endcsname
  {2026/11/26 v2.0a2 A calx component (JFB)}%
```

33.3. Loading data tables

```
\catcode`\_ 11\catcode`\: 11
```

Natural logarithm, P1..P8: tables (records in calxlogdata.tex): \calx_llgUT<ab> (ab = 10..99) #1 -> #1c.t3.t2.t1.t0., c = floor(10^4/(ab+1)), round(10^12 ln(10^3/c)) in limbs; \calx_llgUV<j> (j = 0..101) -> e2.e1.e0., round(10^12 ln(1+j/10^3)).

```
\def\calx_llgULoadT #1.#2.#3.#4.#5.#6,%
{%
    \ifnum#1=\xint_c_ \expandafter\xint_gobble_i\else
    \expandafter\def\csname calx_llgUT#1\endcsname##1{##1#2.#3.#4.#5.#6.}\fi
    \calx_llgULoadT
}%
\def\calx_llgULoadV #1.#2.#3.#4,%
{%
    \ifnum#1<\xint_c_ \expandafter\xint_gobble_i\else
    \expandafter\def\csname calx_llgUV#1\endcsname{#2.#3.#4.}\fi
    \calx_llgULoadV
}%
\mathchardef\calx_ldigits=0
\mathchardef\calx_loutdigits=0
\mathchardef\calx_lscope=1
\mathchardef\calx_lpaircount=0
\input calxlogdata.tex\relax
```

33.4. \calxFpLog, \calxRawLog

Syntax of the public macros of this file (f-expandable; X, A, B f-expanded F, I or raw values; P = \calx_ldigits (working precision); results F or I values rounded to P digits; the \calxRaw<Name> variants give the same value as xint raw A[N]):

```
  \calxFpLog{X}, \calxFpLogOnePlus{X}, \calxFpLogTen{X} ln, ln(1+X), log10
    \calx_lExp{X}, \calxFpPowTen{X}, \calxFpExpMinusOne{X}  e^X, 10^X, e^X-1
    \calxFpPow{A}{B}                                          A^B
    \calxFpSinh{X}, \calxFpCosh, \calxFpTanh, \calxFpCoth, \calxFpSech,
    \calxFpCsch, \calxFpAsinh, \calxFpAcosh, \calxFpAtanh, \calxFpAcoth,
    \calxFpAsech, \calxFpAcsch                                hyperbolic
```

Internal entries (use with \romannumeral0; Y an F value rounded by \CALX_fround, followed by its \xint:):

```
  \calx_llga Y\xint: -> <space>ln Y (dispatch by P)
    \calx_llpa Y\xint:   -> <space>ln(1 + Y)
    \calx_lexa Y\xint:   -> <space>e^Y         (Y rounded at P + 14 digits)
```

and, f-expanded by its callers (\CALX_fnround, \CALX_fmulx),

```
  \calx_llpLP P.+<D>.<e>.<blocks>\xint: -> ln, F value at precision P
```

(internal log of a positive F value, not rounded to P digits; P <= 112).

```
\expandafter\def\csname calx_llgHsF-\endcsname#1\xint:
           {\CALX_nsig{InvalidOperation}{log(x), x < 0}{}}%
\expandafter\def\csname calx_llgHsI+\endcsname#1\xint:{ I+}%
\expandafter\def\csname calx_llgHsI-\endcsname#1\xint:
           {\CALX_nsig{InvalidOperation}{log(-inf)}{}}%
\expandafter\def\csname calx_llgHsI?\endcsname#1\xint:{ I?}%
\def\calx_lLog{\romannumeral0\expandafter\calx_llga
    \romannumeral0\CALX_fround{\calx_ldigits}}%
\def\calxRawLog #1{\romannumeral0\CALX_ftoraw{\calxFpLog{#1}}}%
```

```
\def\calx_llgnfl #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\calxFpLog#3}}%
}%
\def\calx_llgnex #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\calxRawLog#3}}%
}%
\let\XINT_flexpr_func_log\calx_llgnfl
\let\XINT_expr_func_log\calx_llgnex
```

33.5. Natural logarithm, P1..P8

P1..P8 entry (\calx_llga): the input is already rounded to P <= 8 digits, F+<D>.<e>.<blocks>\xint:, D <= 16. E = D - 1 + 8e; |E| >= 4000: the P16 code; else M = the 8 leading digits (\calx_lLin<n> of P16, padded).

```
\expandafter\let\csname calx_llgUtagF-\expandafter\endcsname
   \csname calx_llgHsF-\endcsname
\expandafter\let\csname calx_llgUtagI+\expandafter\endcsname
   \csname calx_llgHsI+\endcsname
\expandafter\let\csname calx_llgUtagI-\expandafter\endcsname
   \csname calx_llgHsI-\endcsname
\expandafter\let\csname calx_llgUtagI?\expandafter\endcsname
   \csname calx_llgHsI?\endcsname
\expandafter\def\csname calx_llgUtagF+\endcsname #1.#2.#3\xint:%
{%
    \ifnum#1=\xint_c_ \xint_dothis{\CALX_nsig{DivisionByZero}{log(0)}{-}}\fi
    \xint_orthat{\expandafter\calx_llgUb
      \the\numexpr#1-\xint_c_i+\xint_c_viii*#2.#1.#2.#3\xint:}%
}%
\def\calx_llgUb #1.#2.#3.%
{%
    \ifnum\ifnum#1<\xint_c_ -\fi#1>3999
      \xint_dothis\calx_llgUb_large\fi
    \xint_orthat\calx_llgUb_regular #1.#2.#3.%
}%
\def\calx_llgUb_large #1.#2.#3.#4\xint:%
   {\csname calx_lLtagF+\endcsname#2.#3.#4\xint:}%
\def\calx_llgUb_regular #1.#2.#3.#4\xint:%
   {\csname calx_lLin\ifnum#2>\xint_c_viii ii\else i\fi\endcsname
      \calx_llgUa#4\xint:#1.#2.}%
\def\calx_llgUa #1#2#3#4#5#6#7#8#9\relax{\calx_llgUc #1#2.#1#2#3#4.#5#6#7#8.}%
```

\calx_llgUc ab.Mh.Ml.E.: classes by E: -1 (cm), 0 (cz), > 0 (cp), < -1 (cn)

```
\def\calx_llgUc #1.#2.#3.#4.%
{%
    \csname calx_llgUc\ifcase\numexpr#4+\xint_c_i\relax m\or z\else
      \ifnum#4>\xint_c_ p\else n\fi\fi\endcsname #1.#2.#3.#4.%
}%
```

```
\def\calx_llgUcp{\calx_llgUo\calx_llgUfp}%
\def\calx_llgUcn{\calx_llgUo\calx_llgUfn}%
```

near 1: x < 1.011 (E = 0), x >= 0.989 (E = -1)

```
\def\calx_llgUcz #1.#2.%
{%
    \ifnum#2>1010 \expandafter\calx_llgUo\expandafter\calx_llgUfp
    \else\expandafter\calx_llgUzz\fi #1.#2.%
}%
\def\calx_llgUcm #1.#2.#3.%
{%
    \ifnum#2#3<98900000 \expandafter\calx_llgUo\expandafter\calx_llgUfn
    \else\expandafter\calx_llgUzm\fi #1.#2.#3.%
}%
```

Ordinary path, fixed point 10^-12. \calx_llgUo <chain>ab.Mh.Ml.E. A = 10^6 - c Mh, B = c Ml (u = (10^4 A - B) 10^-10); j; W = 10^12 u'; S12 = 10^12 (u' + u'^2/2 + u'^3/3); the chain (\calx_llgUfp: E >= 0, \calx_llgUfn: E < 0) gets S12.e2.e1.e0.E.t3.t2.t1.t0.

```
\def\calx_llgUo #1#2.{\csname calx_llgUT#2\endcsname\calx_llgUoa#1}%
\def\calx_llgUoa #1.#2.#3.#4.#5.#6#7.#8.#9.%
{%
    \expandafter\calx_llgUob\the\numexpr1000000-#1*#7\expandafter.%
    \the\numexpr#1*#8.#6#9.#2.#3.#4.#5.%
}%
\def\calx_llgUob #1.#2.%
{%
    \expandafter\calx_llgUoc
    \the\numexpr(#1*10000-#2)/(10000000-10*#1+#2/1000).#1.#2.%
}%
\def\calx_llgUoc #1.#2.#3.%
{%
    \expandafter\calx_llgUod
    \the\numexpr(#2*(1000+#1)-1000000*#1)*1000-#3*(1000+#1)/10.#1.%
}%
\def\calx_llgUod #1.#2.#3%
{%
    \expandafter#3\the\numexpr#1+(#1/1000)*(#1/1000)/2000000
      +((#1/1000)*(#1/1000)/2000000)*(#1/1000)/1500000000\expandafter.%
    \romannumeral`&&@\csname calx_llgUV#2\endcsname
}%
```

|y| 10^12 = s(E ln10 + ln m) 10^12, ln10 10^12 = 2|3025|8509|2994 + 0.045684

```
\def\calx_llgUfp #1.#2.#3.#4.#5.#6.#7.#8.#9.%
{%
    \expandafter\calx_llgUw\the\numexpr#6+2*#5%
    \expandafter\calx_lLR\the\numexpr\calx_lLoff+#7-#2+3025*#5%
    \expandafter\calx_lLR\the\numexpr\calx_lLoff+#8-#3-#1/10000+8509*#5%
    \expandafter\calx_lLR\the\numexpr\calx_lLoff+#9-#4-#1+#1/10000*10000
      +2994*#5+#5*45684/1000000\relax!+-12.%
}%
\def\calx_llgUfn #1.#2.#3.#4.#5.#6.#7.#8.#9.%
{%
    \expandafter\calx_llgUw\the\numexpr-#6-2*#5%
    \expandafter\calx_lLR\the\numexpr\calx_lLoff-#7+#2-3025*#5%
```

```
    \expandafter\calx_lLR\the\numexpr\calx_lLoff-#8+#3+#1/10000-8509*#5%
    \expandafter\calx_lLR\the\numexpr\calx_lLoff-#9+#4+#1-#1/10000*10000
      -2994*#5-#5*45684/1000000\relax!--12.%
}%
```

Near-1 path: delta = d 10^-k, y = delta (1 + delta H'), Q = round(10^10 delta H'); |y| = a (10^10 + Q) 10^-(k+10), a = |d| = a1 10^4 + a0. \calx_llgUzh a1.d.a0.10^k.10^(k-1).<sign><z0>.

```
\begingroup\catcode`F=12
\gdef\calx_llgUzz #1.#2.#3.#4.%
{%
    \ifnum#2#3=10000000 \xint_dothis{ F+0.0.}\fi
    \xint_orthat{\expandafter\calx_llgUzh\the\numexpr#2-1000\expandafter.%
      \the\numexpr#2#3-10000000.#3.10000000.1000000.+-17.}%
}%
\endgroup
\def\calx_llgUzm #1.#2.#3.#4.%
{%
    \expandafter\calx_llgUzh\the\numexpr9999-#2\expandafter.%
    \the\numexpr#2#3-100000000\expandafter.%
    \the\numexpr10000-#3.100000000.10000000.--18.%
}%
\def\calx_llgUzh #1.#2.#3.#4.#5.%
{%
    \expandafter\calx_llgUzb\the\numexpr
      #2*(-500000000+#2*(333333333+#2*(-250000000
      +#2*200000000/#4)/#4)/#4)/#5.#1.#3.%
}%
```

exact a (10^10 + Q): columns 100 a1 | 100 a0 + a1 Q1 | a1 Q0 + a0 Q1 | a0 Q0

```
\def\calx_llgUzb #1.#2.#3.%
{%
    \expandafter\calx_llgUw\the\numexpr100*#2%
    \expandafter\calx_lLR\the\numexpr\calx_lLoff+#2*(#1/10000)+100*#3%
    \expandafter\calx_lLR\the\numexpr
                          \calx_lLoff+#2*(#1-#1/10000*10000)+#3*(#1/10000)%
    \expandafter\calx_lLR\the\numexpr\calx_lLoff+#3*(#1-#1/10000*10000)\relax!%
}%
```

Writer: T!l2!l1!l0!<sign><z0>. (|y| = T l2 l1 l0 10^z0): H = T l2, L = l1 l0, n = digits of H (<= 9), v = n - P; M = round(H 10^8 + L)/10^(8+v)); g = z0 + 8 + v; \calx_ldgoc M.b.g.<sign> writes the native.

```
\def\calx_llgUw #1!#2!#3!#4!#5#6.%
{%
    \expandafter\calx_llgUwa\the\numexpr\expandafter\CALX_nd_c
      \the\numexpr#1#2\relax{10}987654321\xint:-\calx_ldigits.#1#2.#3#4.#6.#5%
}%
\def\calx_llgUwa #1.#2.#3.#4.%
{%
    \expandafter\calx_ldgoc\the\numexpr\ifnum#1>\xint_c_ #2/\CALX_pten{#1}\else
      #2*\CALX_pten{-#1}+#3/\CALX_pten{8+#1}\fi
    \expandafter.\the\numexpr(2*(#4+8+#1)-7)/16\expandafter.%
                 \the\numexpr#4+8+#1.%
}%
```

33.6. Natural logarithm, P9..P16

Natural logarithm, P9..P16. x = 10^E m, m = M 10^-15, M the 16-digit mantissa as limbs m1!m2!m3!m4!. Grid W = 24: V = 10^24 (1+t) as limbs v0!..v6! (v0 units limb, 0 or 1). Bootstrap 1+t0 = c m/10^4, c = nint(200000/(2a+1)), a = first two digits (unity zone: c = 10^4 if E = 0, 10^3 if E = -1, no constant at all). Stages k = 2,4,6 (classes B, C) or 2,4,6,8 (class A = [1.1,1.2) and the unity zone): 1+t <- (1+t)(1+j 10^-k), j = nint(-10R/(10^4+T)). Constants: tables for c, k=2, k=4 (calxlogdata.tex, \calx_lLb<a>, \calx_lLs<100j>, \calx_lLt<j>); k=6,8 on the fly; ln(1+t) = t - t^2/2. Final sum in 8-digit pairs, one carry chain with E ln10 (|E| < 10^4); 10^4 <= |E|: exact native addition of E floor(10^24 ln10) 10^-24. Unity zone with |x-1| < 10^-4: direct series y = delta (1 - Q).

```
\csname newcount\endcsname\calx_lLoff \calx_lLoff=1200000000
\csname newcount\endcsname\calx_lLob \calx_lLob=1000000000
\let\calx_lLQ\relax
\def\calx_lLR #1#2#3#4#5#6{+#1#2#3#4#5#6-120000!}%
\def\calx_lLRb #1#2#3#4#5#6{+#1#2#3#4#5#6-100000!}%
\def\calx_lLRz #1#2#3#4#5#6#7#8#9{+#1#2#3#4#5#6-120000\xint_gobble_i}%
\let\calx_lLfr\CALX_fnround
```

entry: F+<D>.<e>.<blocks>\xint: ; n = ceil(D/8) blocks, E = D-1+8e

```
\expandafter\def\csname calx_lLtagF+\endcsname #1.#2.#3\xint:%
{%
    \ifnum#1=\xint_c_ \xint_dothis{\CALX_nsig{DivisionByZero}{log(0)}{-}}\fi
    \xint_orthat{\expandafter\calx_lLtb
                 \the\numexpr(#1+\xint_c_iii)/\xint_c_viii
      \expandafter.\the\numexpr#1-\xint_c_i+\xint_c_viii*#2.#1.#3\xint:}%
}%
\def\calx_lLtb #1.#2.#3.#4\xint:%
   {\calx_lLfr\calx_ldigits{\csname calx_lLin%
                          \romannumeral#1\endcsname\calx_lLM#4\xint:#2.#3.}}%
```

digits of M (MS first), padded with zeros; #1 continuation (shared with the generic path): n = 1..5 blocks, z = 8n - D leading zeros

```
\def\calx_lLini #11#2!\xint:#3.#4.%
   {\csname calx_lLgz\the\numexpr8-#4\endcsname
    #1#200000000000000000000000000000000\relax#3.}%
\def\calx_lLinii #11#2!1#3!\xint:#4.#5.%
   {\csname calx_lLgz\the\numexpr16-#5\endcsname
    #1#3#200000000000000000000000000000000\relax#4.}%
\def\calx_lLiniii #11#2!1#3!1#4!\xint:#5.#6.%
   {\csname calx_lLgz\the\numexpr24-#6\endcsname
    #1#4#3#200000000000000000000000000000000\relax#5.}%
\def\calx_lLiniv #11#2!1#3!1#4!1#5!\xint:#6.#7.%
   {\csname calx_lLgz\the\numexpr32-#7\endcsname
    #1#5#4#3#200000000000000000000000000000000\relax#6.}%
\def\calx_lLinv #11#2!1#3!1#4!1#5!1#6!\xint:#7.#8.%
   {\csname calx_lLgz\the\numexpr40-#8\endcsname
    #1#6#5#4#3#200000000000000000000000000000000\relax#7.}%
\expandafter\def\csname calx_lLgz0\endcsname#1{#1}%
\expandafter\def\csname calx_lLgz1\endcsname#1#2{#1}%
\expandafter\def\csname calx_lLgz2\endcsname#1#2#3{#1}%
\expandafter\def\csname calx_lLgz3\endcsname#1#2#3#4{#1}%
\expandafter\def\csname calx_lLgz4\endcsname#1#2#3#4#5{#1}%
\expandafter\def\csname calx_lLgz5\endcsname#1#2#3#4#5#6{#1}%
```

```
\expandafter\def\csname calx_lLgz6\endcsname#1#2#3#4#5#6#7{#1}%
\expandafter\def\csname calx_lLgz7\endcsname#1#2#3#4#5#6#7#8{#1}%
```

a = the first two digits, read as tokens

```
\def\calx_lLM #1#2#3#4#5#6#7#8{\calx_lLMb{#1#2.#1#2#3#4!#5#6#7#8!}}%
\def\calx_lLMb #1#2#3#4#5#6#7#8#9{\calx_lLMc{#1#2#3#4#5!#6#7#8#9!}}%
\def\calx_lLMc #1#2\relax{\calx_lLclb#1}%
```

classification: a.m1!m2!m3!m4!E.

```
\def\calx_lLclb #1.#2!#3!#4!#5!#6.%
{%
    \ifnum#6=\xint_c_ \xint_dothis{\calx_lLcZ#1.#2!#3!#4!#5!}\fi
    \ifnum#6=-\xint_c_i \xint_dothis{\calx_lLcM#1.#2!#3!#4!#5!}\fi
    \ifnum\ifnum#6<\xint_c_ -\fi#6>9999
      \xint_dothis{\calx_lLcE#1.#2!#3!#4!#5!#6.}\fi
    \xint_orthat{\expandafter\calx_lLB\the\numexpr10*(200000/(2*#1+1));%
      #2!#3!#4!#5!\calx_lLsBa#1;#6;}%
}%
```

E = 0: a = 10 is the unity zone [1,1.1): x = 1 exact zero; x < 1.0001 direct series (\calx_lLUz); else stages 2..8 with c = 10^4. a = 11: class A. E = -1: a >= 90 (x <> 0.9) is the unity zone [0.9,1): x > 0.9999 direct series (\calx_lLUm); else stages 2..8 with c = 10^3.

```
\begingroup\catcode`F=12
\gdef\calx_lLcZ #1.%
{%
    \ifnum#1>11 \xint_dothis\calx_lLcZ_high\fi
    \ifnum#1=11 \xint_dothis\calx_lLcZ_oneone\fi
    \xint_orthat\calx_lLcZ_unity #1.%
}%
\gdef\calx_lLcZ_high #1.#2!#3!#4!#5!%
{%
    \expandafter\calx_lLB
       \the\numexpr10*(200000/(2*#1+1));#2!#3!#4!#5!\calx_lLsBa#1;0;%
}%
\gdef\calx_lLcZ_oneone #1.#2!#3!#4!#5!%
{%
    \expandafter\calx_lLB
       \the\numexpr10*(200000/(2*#1+1));#2!#3!#4!#5!\calx_lLsAa#1;0;%
}%
\gdef\calx_lLcZ_unity #1.#2!#3!#4!#5!%
{%
    \ifnum\numexpr#2#3-10000000+#4#5\relax=\xint_c_ \xint_dothis{F+0.0.}\fi
    \ifnum#2#3<10001000 \xint_dothis{\calx_lLUz#2!#3!#4!#5!}\fi
    \xint_orthat{\calx_lLB100000;#2!#3!#4!#5!\calx_lLsAa0;0;}%
}%
\gdef\calx_lLcM #1.%
{%
    \ifnum#1<90 \xint_dothis\calx_lLcM_low\fi
    \xint_orthat\calx_lLcM_regular #1.%
}%
\gdef\calx_lLcM_low #1.#2!#3!#4!#5!%
{%
    \expandafter\calx_lLB
       \the\numexpr10*(200000/(2*#1+1));#2!#3!#4!#5!\calx_lLsBa#1;-1;%
```

```
}%
\gdef\calx_lLcM_regular #1.#2!#3!#4!#5!%
{%
    \ifnum\numexpr#2#3-90000000+#4#5\relax
       =\xint_c_ \xint_dothis{\expandafter\calx_lLB
       \the\numexpr10*(200000/(2*#1+1));#2!#3!#4!#5!\calx_lLsBa#1;-1;}\fi
    \ifnum#2#3<99990000
      \xint_dothis{\calx_lLB10000;#2!#3!#4!#5!\calx_lLsAa0;0;}\fi
    \ifnum\numexpr#2#3-99990000+#4#5\relax=\xint_c_
      \xint_dothis{\calx_lLB10000;#2!#3!#4!#5!\calx_lLsAa0;0;}\fi
    \xint_orthat{\calx_lLUm#2!#3!#4!#5!}%
}%
\endgroup
\def\calx_lLcE #1.#2!#3!#4!#5!#6.%
{%
    \CALX_faddx{\expandafter\calx_lLB\the\numexpr10*(200000/(2*#1+1));%
       #2!#3!#4!#5!\calx_lLsBa#1;0;}%
      {\CALX_fmulx{\romannumeral0\CALX_ffromraw{#6[0]}}{\calx_lLzeta}}%
}%
\edef\calx_lLzeta{\romannumeral0\CALX_ffromraw
                  {230258509299404568401799 1[-24]}}%
```

bootstrap: J;m1!m2!m3!m4!<continuation> -> <cont>v0!...!v6!

```
\def\calx_lLB #1;#2!#3!#4!#5!#6%
{%
    \expandafter#6\the\numexpr\xint_c_\expandafter\calx_lLRb
    \the\numexpr\calx_lLob\expandafter\calx_lLRb
    \the\numexpr\calx_lLob+#1*#2\expandafter\calx_lLRb
    \the\numexpr\calx_lLob+#1*#3\expandafter\calx_lLRb
    \the\numexpr\calx_lLob+#1*#4\expandafter\calx_lLRb
    \the\numexpr\calx_lLob+#1*#5\relax!0000!%
}%
```

stage kernels: <cont>J;v0!..v6! -> <cont>v0'!..v6'!J; V' = V + floor(J V'/10^(4a)), a = 1 (\calx_lLKa) or 2 (\calx_lLKb); the lowest column only gives its carry.

```
\def\calx_lLKa #1#2;#3!#4!#5!#6!#7!#8!#9!%
{%
    \expandafter#1\the\numexpr#3\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+#4+#2*#3\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+#5+#2*#4\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+#6+#2*#5\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+#7+#2*#6\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+#8+#2*#7\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+#9+#2*#8\expandafter\calx_lLRz
    \the\numexpr\calx_lLoff+#2*#9\relax!#2;%
}%
\def\calx_lLKb #1#2;#3!#4!#5!#6!#7!#8!#9!%
{%
    \expandafter#1\the\numexpr#3\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+#4\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+#5+#2*#3\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+#6+#2*#4\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+#7+#2*#5\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+#8+#2*#6\expandafter\calx_lLR
```

```
    \the\numexpr\calx_lLoff+#9+#2*#7\expandafter\calx_lLRz
    \the\numexpr\calx_lLoff+#2*#8\relax!#2;%
}%
\def\calx_lLKz #1#2;#3!#4!#5!#6!#7!#8!#9!{#1#3!#4!#5!#6!#7!#8!#9!0;}%
```

selectors: y = (v0-1)10^4 + v1 = floor(10^4 t) is T \calx_lLdef <name><next><kernel><skip>{<factor>}{<selector>

```
\def\calx_lLsel #1#2#3#4;%
   {\ifnum#4=\xint_c_ \expandafter#2\else\expandafter#1\fi#3#4;}%
\def\calx_lLdef #1#2#3#4#5#6%
   {\def#1##1!##2!##3!##4!{\expandafter\calx_lLsel
      \expandafter#3\expandafter#4%
      \expandafter#2\the\numexpr#5(#6);##1!##2!##3!##4!}}%
```

stages k = 2,4,6: names #1,#2,#3, next #4, kernels shift 1 (#5), shift 2 (#6), skip (#7); stage k = 8: name, next, kernel, skip. Shared by P9..P16 and P17..P32.

```
\def\calx_lLsdef #1#2#3#4#5#6#7%
{%
    \calx_lLdef#1#2#5#7{100*}%
      {-10*(((##1-1)*10000+##2)*10+##3/1000)/((##1-1)*10000+##2+10000)}%
    \calx_lLdef#2#3#5#7{}%
      {-10*(((##1-1)*10000+##2)*1000+##3/10)/((##1-1)*10000+##2+10000)}%
    \calx_lLdef#3#4#6#7{100*}%
      {-10*((((##1-1)*10000+##2)*10000+##3)*10+##4/1000)/%
       ((##1-1)*10000+##2+10000)}%
}%
\def\calx_lLsdefd #1#2#3#4%
{%
    \calx_lLdef#1#2#3#4{}%
      {-10*((((##1-1)*10000+##2)*10000+##3)*1000+##4/10)/%
       ((##1-1)*10000+##2+10000)}%
}%
\calx_lLsdef\calx_lLsBa\calx_lLsBb\calx_lLsBc\calx_lLfB\calx_lLKa\calx_lLKb\calx_lLKz
\calx_lLsdef\calx_lLsAa\calx_lLsAb\calx_lLsAc\calx_lLsAd\calx_lLKa\calx_lLKb\calx_lLKz
\calx_lLsdefd\calx_lLsAd\calx_lLfA\calx_lLKb\calx_lLKz
```

final, stages 2,4,6: v0!..v6!J6; (then J4;J2;a;E;) w = floor(10^14 t); pairs P1 = v1v2, P2 = v3v4, P3 = v5v6; j6 terms (series of -ln(1+j 10^-6)): -100j at limb 2, 5000j^2 at limb 4, -100j^3/3 at limb 5 (+ remainder at limb 6), j^4/4 - j^5/(5 10^6) at limb 6; -t^2/2 = -w^2/(2 10^8) at limb 5.

```
\def\calx_lLfB #1!#2!#3!#4!#5!#6!#7!#8;%
{%
    \expandafter\calx_lLfBb\the\numexpr#8/100\expandafter;\the\numexpr
      (((#1-1)*10000+#2)*10000+#3)*1000000+%
       #4*100+#5/100;#8;#1!#2#3!#4#5!#6#7!%
}%
\def\calx_lLfBb #1;#2;#3;#4!#5!#6!#7!%
{%
    \expandafter\calx_lLTa\the\numexpr#4#5-100000000-#3\expandafter!%
    \the\numexpr#6+#3*#3/2\expandafter!%
    \the\numexpr#7-(100*#1*#1*#1-3*(100*#1*#1*#1/3))*3333+#1*#1*#1*#1/4%
      -#1*#1*#1*#1*#1/5000000\expandafter!%
    \the\numexpr0-100*#1*#1*#1/3-#2*#2/200000000;%
}%
```

final, stages 2,4,6,8: v0!..v6!J8;J6; w = floor(10^16 t)

```
\def\calx_lLfA #1!#2!#3!#4!#5!#6!#7!#8;#9;%
```

```
{%
    \expandafter\calx_lLfAb\the\numexpr#9/100\expandafter;\the\numexpr
      ((((#1-1)*10000+#2)*10000+#3)*10000+#4)*10000+#5;#8;#9;%
      #1!#2#3!#4#5!#6#7!%
}%
\def\calx_lLfAb #1;#2;#3;#4;#5!#6!#7!#8!%
{%
    \expandafter\calx_lLTa\the\numexpr#5#6-100000000-#4-#3\expandafter!%
    \the\numexpr#7+#4*#4/2\expandafter!%
    \the\numexpr#8-(100*#1*#1*#1-3*(100*#1*#1*#1/3))*3333+#1*#1*#1*#1/4%
      -#1*#1*#1*#1*#1/5000000-#3*#3*#3/3-#2*#2/200000000\expandafter!%
    \the\numexpr5000*#3*#3-100*#1*#1*#1/3;%
}%
```

add the tables (8-digit pairs, no carry): A!P2!P3!S5;key; A = floor(10^8 y), P2, P3 the next pairs (entries: F1!c2!c3!...)

```
\def\calx_lLT #1#2#3;#4;%
   {\expandafter\expandafter\expandafter#1\csname#2#4\endcsname\calx_lLQ#3;}%
\def\calx_lLTa{\calx_lLT\calx_lLCAa{calx_lLt}}%
\def\calx_lLTb{\calx_lLT\calx_lLCAb{calx_lLs}}%
\def\calx_lLTc{\calx_lLT\calx_lLCAc{calx_lLb}}%
\def\calx_lLtmp #1#2%
   {\def#1##1!##2!##3!##4\calx_lLQ##5!##6!##7!%
     {\expandafter#2\the\numexpr##1+##5\expandafter!\the\numexpr##2+##6%
      \expandafter!\the\numexpr##3+##7!}}%
\calx_lLtmp\calx_lLCAa\calx_lLTb
\calx_lLtmp\calx_lLCAb\calx_lLTc
\calx_lLtmp\calx_lLCAc\calx_lLTd
\let\calx_lLtmp\relax
```

carry chain with E ln10 (limbs of floor(10^24 ln 10) written in). Template, shared with exp: ↪ calx_lLTdef <name><cont>{<params>} {<integer column prefix>}{<tail>}; params ##1 A (10^-8), ##2 P2 (10^-16), ##3 P3 (10^-24), ##4 S5 (10^-20), ##5 E; output <cont>I!l1!..!l6!<tail>.

```
\def\calx_lLTdef #1#2#3#4#5%
{%
  \def#1#3%
  {%
    \expandafter#2\the\numexpr#4 2*##5\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+3025*##5\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+##1+8509*##5\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+2994*##5\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+##2+456*##5\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+##4+8401*##5\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+##3+7991*##5\relax!#5%
  }%
}%
\calx_lLTdef\calx_lLTd\calx_lLNat{#1!#2!#3!#4;#5;}{}{}%
```

floor form I!l1!..!l6! -> native float, blocks written directly (value I + 0.l1...l6; negative: complement, |y| - 10^-24); D from calxcore's \CALX_ndig. Only |y| < 10^-8 (large-E path) uses \calx_trsa. Template (shared with P24): \calx_lLNdef <at><p><n>{<params>}{<e>} {<blocks p>}{<blocks n>}{<limbs>}{<rsa prefix>}; e = number of 8-digit pairs after I (D = 8e + ndig).

```
\begingroup\catcode`F=12
\gdef\calx_lLNdef #1#2#3#4#5#6#7#8#9%
```

```
{%
  \def#1##1!{\ifnum##1<\xint_c_ \expandafter#3\else\expandafter#2\fi##1!}%
  \def#2#4%
  {%
    \ifnum##1>\xint_c_ \xint_dothis{\expanded{F+\the\numexpr#5*8+%
      \CALX_ndig{##1}.-#5.#61\expandafter\xint_gobble_i
      \the\numexpr\xint_c_e_viii+##1!}}\fi
    \ifnum##2##3>\xint_c_ \xint_dothis{\expanded{F+\the\numexpr#5*8-8+%
      \CALX_ndig{##2##3}.-#5.#6}}\fi
    \xint_orthat{\expanded{\calx_trsa#9\calx_tbp+!\expandafter\xint_gobble_i
      \the\numexpr\xint_c_e_viii+##1!#8}}%
  }%
  \def#3#4%
  {%
    \ifnum##1<-\xint_c_i \xint_dothis{\expanded{F-\the\numexpr#5*8+%
      \CALX_ndig{-##1-1}.-#5.#7%
      1\expandafter\xint_gobble_i\the\numexpr\xint_c_e_viii-##1-1!}}\fi
    \ifnum##2##3<99999999 \xint_dothis{\expanded{F-\the\numexpr#5*8-8+%
      \CALX_ndig{99999999-##2##3}.-#5.#7}}\fi
    \xint_orthat{\expanded{\calx_trsa#9\calx_tbn-!\the\numexpr
      \xint_c_e_viii+##1!#8}}%
  }%
}%
\endgroup
\calx_lLNdef\calx_lLNat\calx_lLNp\calx_lLNn{#1!#2!#3!#4!#5!#6!#7!}{3}%
   {1#6#7!1#4#5!1#2#3!}%
   {\the\numexpr199999999-#6#7!\the\numexpr199999999-#4#5!%
    \the\numexpr199999999-#2#3!}%
   {#2!#3!#4!#5!#6!#7!}{4.3.}%
```

unity zone, |x-1| < 10^-4: delta = s D 10^-16, D < 10^12 as limbs a!b!c! (c <= 10^4). Horner G_n = 1/n - delta G_(n+1) on 6 fixed limbs (\calx_lLH<level><sign>), from G_5 (|x-1| >= 10^-5), G_4 (>= 10^-7), G_3 (>= 10^-10) or G_2; Q = delta G_2; y = s D (1 - Q) 10^-16 with D (1 - Q) computed exactly (\calx_lLY<sign>). Relative error of y below 3.4 10^-21 (remainder), budget 5 10^-20.

```
\csname newcount\endcsname\calx_lLoffh \calx_lLoffh=1400000000
\def\calx_lLRh #1#2#3#4#5#6{+#1#2#3#4#5#6-140000!}%
\def\calx_lLRhz #1#2#3#4#5#6#7#8#9{+#1#2#3#4#5#6-140000\xint_gobble_i}%
\def\calx_lLUz #1!#2!#3!#4!%
{%
    \expandafter\calx_lLUs\expandafter P\the\numexpr10*#2+(2*#3-999)/2000%
    \expandafter!\the\numexpr10*(#3-1000*((2*#3-999)/2000))+(2*#4-999)/2000%
    \expandafter!\the\numexpr10*(#4-1000*((2*#4-999)/2000))!%
}%
\def\calx_lLUm #1!#2!#3!#4!%
{%
    \expandafter\calx_lLUs\expandafter M\the\numexpr9999-#2\expandafter!%
    \the\numexpr9999-#3\expandafter!\the\numexpr10000-#4!%
}%
\def\calx_lLUs #1#2!#3!%
{%
    \ifnum#2>999 \xint_dothis{\csname calx_lLHd#1\endcsname
      2000!0000!0000!0000!0000!0000!}\fi
    \ifnum#2>9 \xint_dothis{\csname calx_lLHc#1\endcsname
```

```
      2500!0000!0000!0000!0000!0000!}\fi
    \ifnum\numexpr#2*10000+#3>99 \xint_dothis{\csname calx_lLHb#1\endcsname
      3333!3333!3333!3333!3333!3333!}\fi
    \xint_orthat{\csname calx_lLHq#1\endcsname
      5000!0000!0000!0000!0000!0000!}#2!#3!%
}%
```

G' = C <op> D 10^-16 G: g1..g6, a, b, c; C literal (floor of 1/n)

```
\def\calx_lLtmp #1#2#3#4#5#6#7#8#9%
{%
    \def#1##1!##2!##3!##4!##5!##6!##7!##8!##9!%
    {%
        \expandafter#2\the\numexpr#4\expandafter\calx_lLRh
        \the\numexpr\calx_lLoffh+#5\expandafter\calx_lLRh
        \the\numexpr\calx_lLoffh+#6#3##7*##1\expandafter\calx_lLRh
        \the\numexpr\calx_lLoffh+#7#3(##7*##2+##8*##1)\expandafter\calx_lLRh
        \the\numexpr\calx_lLoffh+#8#3(##7*##3+##8*##2+##9*##1)%
        \expandafter\calx_lLRh
        \the\numexpr\calx_lLoffh+#9#3(##7*##4+##8*##3+##9*##2)%
        \expandafter\calx_lLRhz
        \the\numexpr\calx_lLoffh#3(##7*##5+##8*##4+##9*##3)\relax!##7!##8!##9!%
    }%
}%
\calx_lLtmp\calx_lLHdP\calx_lLHcP-{2500}{0000}{0000}{0000}{0000}{0000}%
\calx_lLtmp\calx_lLHcP\calx_lLHbP-{3333}{3333}{3333}{3333}{3333}{3333}%
\calx_lLtmp\calx_lLHbP\calx_lLHqP-{5000}{0000}{0000}{0000}{0000}{0000}%
\calx_lLtmp\calx_lLHqP\calx_lLYP+{0}{0}{0}{0}{0}{0}%
\calx_lLtmp\calx_lLHdM\calx_lLHcM+{2500}{0000}{0000}{0000}{0000}{0000}%
\calx_lLtmp\calx_lLHcM\calx_lLHbM+{3333}{3333}{3333}{3333}{3333}{3333}%
\calx_lLtmp\calx_lLHbM\calx_lLHqM+{5000}{0000}{0000}{0000}{0000}{0000}%
\calx_lLtmp\calx_lLHqM\calx_lLYM+{0}{0}{0}{0}{0}{0}%
```

D (1 <op> |Q|), exact: q1..q6, a, b, c -> 1<8 digits>!y1!y0!f1!..!f6!

```
\def\calx_lLtmp #1#2#3%
{%
    \def#1##1!##2!##3!##4!##5!##6!##7!##8!##9!%
    {%
        \expandafter#3\the\numexpr\xint_c_e_viii+##7\expandafter\calx_lLRh
        \the\numexpr\calx_lLoffh+##8#2##7*##1\expandafter\calx_lLRh
        \the\numexpr\calx_lLoffh+##9#2(##7*##2+##8*##1)\expandafter\calx_lLRh
        \the\numexpr\calx_lLoffh#2(##7*##3+##8*##2+##9*##1)\expandafter\calx_lLRh
        \the\numexpr\calx_lLoffh#2(##7*##4+##8*##3+##9*##2)\expandafter\calx_lLRh
        \the\numexpr\calx_lLoffh#2(##7*##5+##8*##4+##9*##3)\expandafter\calx_lLRh
        \the\numexpr\calx_lLoffh#2(##7*##6+##8*##5+##9*##4)\expandafter\calx_lLRh
        \the\numexpr\calx_lLoffh#2(##8*##6+##9*##5)\expandafter\calx_lLRh
        \the\numexpr\calx_lLoffh#2##9*##6\relax!%
    }%
}%
\calx_lLtmp\calx_lLYP-\calx_lLUop
\calx_lLtmp\calx_lLYM+\calx_lLUom
\let\calx_lLtmp\relax
```

y = s Z 10^-40, Z the integer of the five blocks

```
\def\calx_lLUop 1#1!#2!#3!#4!#5!#6!#7!#8!#9!%
```

```
    {\expanded{\calx_trsa5.5.\calx_tbp+!#1!#2!#3!#4!#5!#6!#7!#8!#9!}}%
\def\calx_lLUom 1#1!#2!#3!#4!#5!#6!#7!#8!#9!%
    {\expanded{\calx_trsa5.5.\calx_tbp-!#1!#2!#3!#4!#5!#6!#7!#8!#9!}}%
```

special values (tags F-, I+, I-, I?)

```
\expandafter\let\csname calx_lLtagF-\expandafter\endcsname
    \csname calx_llgHsF-\endcsname
\expandafter\let\csname calx_lLtagI+\expandafter\endcsname
    \csname calx_llgHsI+\endcsname
\expandafter\let\csname calx_lLtagI-\expandafter\endcsname
    \csname calx_llgHsI-\endcsname
\expandafter\let\csname calx_lLtagI?\expandafter\endcsname
    \csname calx_llgHsI?\endcsname
```

(dispatch \calx_llga: at the end of the P17..P32 code)

33.7. Natural logarithm, P17..P32

Layout L = 8 ceil(P/8) (24 or 32), grid W = L + 8, V = 10^W (1+t) as v0!v1!..!vn! (n = W/4 = 8 or 10), followed by the sentinel pair \calx_lLQ!\calx_lLQ!. Stages k = 2,4,6,8 with tables for every class; series t - t^2/2 + t^3/3 (- t^4/4 for L = 32). |x - 1| < 10^-4: y = delta G by the series kernel \calx_tZ (calxtrig). entry: F+<D>.<e>.<blocks>\xint:

```
\begingroup\catcode`F=12
\expandafter\gdef\csname calx_lLGtagF+\endcsname #1.#2.#3\xint:%
{%
    \ifnum#1=\xint_c_ \xint_dothis{\CALX_nsig{DivisionByZero}{log(0)}{-}}\fi
    \xint_orthat{\expandafter\calx_lLGtb\the\numexpr(#1+\xint_c_iii)/\xint_c_viii
      \expandafter.\the\numexpr#1-\xint_c_i+\xint_c_viii*#2.#1.#3\xint:}%
}%
\endgroup
\def\calx_lLGtb #1.#2.#3.#4\xint:%
    {\calx_lLfr\calx_ldigits{\calx_lLGL\calx_ldigits.#1.#2.#3.#4\xint:}}%
```

Internal log at precision P (#1: the token \calx_ldigits, or digits delimited by the dot), n blocks, E, D, blocks: digits by \calx_lLin<n> (P <= 32) or \calx_lLGin, continuation \calx_lLGh<g> (g = (P-5)/8 groups more), then {P} in the slot read by \calx_lLGcl (#6).

```
\def\calx_lLGL #1.#2.#3.#4.#5\xint:%
    {\csname calx_l\ifnum#1>32 LGin\else Lin\romannumeral#2\fi\expandafter
     \endcsname\csname calx_lLGh\the\numexpr(#1-5)/8\endcsname#5\xint:#3.#4.{#1}}%
```

P > 32: blocks (LS first) reversed by a loop, leading zeros of the top block removed (\calx_lLgz), 112 zeros of padding (P <= 112)

```
\def\calx_lLGin #1{\calx_lLGir{#1}{}}%
\def\calx_lLGir #1#2#3{\xint_gob_til_xint: #3\calx_lLGie\xint:\calx_lLGis{#1}{#2}#3}%
\def\calx_lLGis #1#2#3#4!{\calx_lLGir{#1}{#4#2}}%
\def\calx_lLGie\xint:\calx_lLGis#1#2\xint:#3.#4.%
{%
    \csname calx_lLgz\the\numexpr8*((#4+3)/8)-#4\endcsname#1#2%
    00000000000000000000000000000000000000000000000000000000%
    00000000000000000000000000000000000000000000000000000000\relax#3.%
}%
```

digits -> a.m1!m2!..., 8 digits per call (g = (P-5)/8 more calls): first call \calx_lLGh<g> (g baked, g = 2..13)

```
\def\calx_lLtmp #1#2%
```

```
574    {\expandafter\edef\csname calx_lLGg#1\endcsname##1##2##3##4##5##6##7##8##9%
575       {\expandafter\noexpand\csname calx_lLGg#2\endcsname
576        {##1##2##3##4##5!##6##7##8##9!}}}%
577 \calx_lLtmp{1}{0}\calx_lLtmp{2}{1}\calx_lLtmp{3}{2}\calx_lLtmp{4}{3}%
578 \calx_lLtmp{5}{4}\calx_lLtmp{6}{5}\calx_lLtmp{7}{6}\calx_lLtmp{8}{7}%
579 \calx_lLtmp{9}{8}\calx_lLtmp{10}{9}\calx_lLtmp{11}{10}\calx_lLtmp{12}{11}%
580 \calx_lLtmp{13}{12}%
581 \def\calx_lLtmp #1%
582    {\expandafter\edef\csname calx_lLGh#1\endcsname##1##2##3##4##5##6##7##8%
583       {\expandafter\noexpand\csname calx_lLGg#1\endcsname
584        {##1##2.##1##2##3##4!##5##6##7##8!}}}%
585 \calx_lLtmp{2}\calx_lLtmp{3}\calx_lLtmp{4}\calx_lLtmp{5}\calx_lLtmp{6}\calx_lLtmp{7}%
586 \calx_lLtmp{8}\calx_lLtmp{9}\calx_lLtmp{10}\calx_lLtmp{11}\calx_lLtmp{12}%
587 \calx_lLtmp{13}%
588 \expandafter\def\csname calx_lLGg0\endcsname #1#2\relax#3.%
589    {\calx_lLGcl#3.#1\calx_lLQ!\calx_lLQ!}%
```

classification: E.a.m1!m2!...\calx_lLQ!\calx_lLQ!{P} (P: the token \calx_ldigits or digits; stages \calx_lLGsa<g>, g = (P-5)/8; P. appended at the end of the tail J8;J6;J4;J2;a;E; for \calx_lLGo / absorbed by L24) "#1#2" (E then a) is 10 iff E = 0, a = 10; in [-199,-191] iff E = -1, a >= 91 (a = 90 stays ordinary: x = 0.9 is not in the unity zone).

```
590 \def\calx_lLGcl #1.#2.%
591 {%
592     \ifnum\ifnum#1<\xint_c_ -\fi#1>9999
593       \xint_dothis\calx_lLGcl_large\fi
594     \xint_orthat\calx_lLGcl_regular #1.#2.%
595 }%
596 \def\calx_lLGcl_large #1.#2.#3!#4!#5\calx_lLQ!\calx_lLQ!#6%
597 {%
598     \CALX_faddx{\expandafter\calx_lLGB
599         \csname calx_lLGsa\the\numexpr(#6-5)/8\expandafter\endcsname
600         \the\numexpr10*(200000/(2*#2+1));#3!#4!#5\calx_lLQ!\calx_lLQ!#2;0;#6.}%
601         {\CALX_fmulx{\romannumeral0\CALX_ffromraw{#1[0]}}{\calx_lLzetaL}}%
602 }%
603 \def\calx_lLGcl_regular #1.#2.#3!#4!#5\calx_lLQ!\calx_lLQ!#6%
604 {%
605     \ifnum\ifnum#1#2=10 #3#4\else999999999\fi<10001000
606       \xint_dothis{\calx_lLWs#4!#5\calx_lLQ!{#6}}\fi
607     \ifnum#1#2=10
608       \xint_dothis{\expandafter\calx_lLGB
609         \csname calx_lLGsa\the\numexpr(#6-5)/8\endcsname
610         100000;#3!#4!#5\calx_lLQ!\calx_lLQ!0;0;#6.}\fi
611     \ifnum\ifnum#1#2<-199 \xint_c_\else\ifnum#1#2<-190 #3#4\else\xint_c_\fi\fi
612       >99989999 \xint_dothis{\expandafter\calx_lLWt
613         \the\numexpr#4-10000!#5\calx_lLQ!{#6}}\fi
614     \ifnum\ifnum#1#2<-199 \xint_c_\else#1#2\fi<-190
615       \xint_dothis{\expandafter\calx_lLGB
616         \csname calx_lLGsa\the\numexpr(#6-5)/8\endcsname
617         10000;#3!#4!#5\calx_lLQ!\calx_lLQ!0;0;#6.}\fi
618     \xint_orthat{\expandafter\calx_lLGB
619       \csname calx_lLGsa\the\numexpr(#6-5)/8\expandafter\endcsname
620       \the\numexpr10*(200000/(2*#2+1));#3!#4!#5\calx_lLQ!\calx_lLQ!#2;#1;#6.}%
621 }%
```

```
\edef\calx_lLzetaL{\romannumeral0\CALX_ffromraw{\calx_lLzraw}}%
```

end test on the sentinel pair (gob-til)

```
\def\calx_lLGgt #1\calx_lLQ{}%
```

bootstrap: <cont>J;m1!..!\calx_lLQ!\calx_lLQ! -> <cont>v0!..!vn!sentinel V = J M 10^4, J = 10c (unity zone: 10^5 if E = 0, 10^4 if E = -1)

```
\def\calx_lLGB #1#2;%
   {\expandafter#1\the\numexpr\xint_c_\expandafter\calx_lLRb
    \the\numexpr\calx_lLob\calx_lLGBp#2;}%
\def\calx_lLGBp #1;#2!#3!%
{%
    \calx_lLGgt#2\calx_lLGBe\calx_lLQ\expandafter\calx_lLRb
    \the\numexpr\calx_lLob+#1*#2\expandafter\calx_lLRb
    \the\numexpr\calx_lLob+#1*#3\calx_lLGBp#1;%
}%
\def\calx_lLGBe\calx_lLQ#1\calx_lLGBp#2;{\relax!0000!\calx_lLQ!\calx_lLQ!}%
```

stage kernels: <cont>J;v0!..!vn!sentinel -> <cont>v0'!..!vn'!sentinel J; V' = V + floor(J V/10^(4s)), s = 1 (\calx_lLGKa), 2 (\calx_lLGKb); the dropped column only gives its carry (\calx_lLRz).

```
\def\calx_lLGKa #1#2;#3!{\expandafter#1\the\numexpr#3\calx_lLGPa#2;#3!}%
\def\calx_lLGPa #1;#2!#3!#4!%
{%
    \calx_lLGgt#3\calx_lLGPe\calx_lLQ\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+#1*#2+#3\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+#1*#3+#4\calx_lLGPa#1;#4!%
}%
\def\calx_lLGPe\calx_lLQ\expandafter\calx_lLR\the\numexpr\calx_lLoff+#1*#2+#3%
   \calx_lLQ!%
   {\expandafter\calx_lLRz\the\numexpr\calx_lLoff+#1*#2\relax!\calx_lLQ!\calx_lLQ!#1;}%
\def\calx_lLGKb #1#2;#3!{\expandafter#1\the\numexpr#3\calx_lLGPb#2;0!#3!}%
\def\calx_lLGPb #1;#2!#3!#4!#5!%
{%
    \calx_lLGgt#4\calx_lLGPf\calx_lLQ\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+#1*#2+#4\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+#1*#3+#5\calx_lLGPb#1;#4!#5!%
}%
\def\calx_lLGPf\calx_lLQ\expandafter\calx_lLR\the\numexpr\calx_lLoff+#1*#2+#3%
   \calx_lLQ!\calx_lLQ!%
   {\expandafter\calx_lLRz\the\numexpr\calx_lLoff+#1*#2\relax!\calx_lLQ!\calx_lLQ!#1;}%
\def\calx_lLGKz #1#2;#3\calx_lLQ!\calx_lLQ!{#1#3\calx_lLQ!\calx_lLQ!0;}%
```

selectors (templates of P9..P16): one set per layout g = 2..13 (\calx_lLGsa<g>..\calx_lLGsd<g>, final \calx_lLGf<g>): the layout reaches the final through the selectors' continuations. final: layout g and sign by v0 (|t| < 5.061e-9: v1 = v2 = 0000 if v0 = 1, 9999 if v0 = 0)

```
\def\calx_lLtmp #1%
   {\expandafter\calx_lLtmpc\csname calx_lLGsa#1\expandafter\endcsname
    \csname calx_lLGsb#1\expandafter\endcsname\csname calx_lLGsc#1\expandafter
    \endcsname\csname calx_lLGsd#1\expandafter\endcsname
    \csname calx_lLGf#1\endcsname{#1}}%
\def\calx_lLtmpc #1#2#3#4#5#6%
{%
    \calx_lLsdef#1#2#3#4\calx_lLGKa\calx_lLGKb\calx_lLGKz
    \calx_lLsdefd#4#5\calx_lLGKb\calx_lLGKz
```

```
    \def#5##1!{\csname calx_lLGf\ifnum##1=\xint_c_ n\else p\fi#6\endcsname}%
}%
\calx_lLtmp{2}\calx_lLtmp{3}\calx_lLtmp{4}\calx_lLtmp{5}\calx_lLtmp{6}\calx_lLtmp{7}%
\calx_lLtmp{8}\calx_lLtmp{9}\calx_lLtmp{10}\calx_lLtmp{11}\calx_lLtmp{12}%
\calx_lLtmp{13}%
\let\calx_lLtmpc\relax
```

L = 24 (W = 32): v3..v8 -> P4!P3!P2!A; tau = |t| limbs (n: 9999-v, |t| - 10^-32), s = tau3tau4, rho = tau5tau6; pairs P2 (10^-16), P3 (10^-24), P4 (10^-32). y = t - t^2/2 + t^3/3. #1 suffix, #2 v1!v2!, #3 limb complement prefix, #4 pair complement prefix, #5 A, #6 sign of the cubic

```
\def\calx_lLtmp #1#2#3#4#5#6%
{%
    \expandafter\def\csname calx_lLGf#1\endcsname#2##1!##2!##3!##4!##5!##6!%
       \calx_lLQ!\calx_lLQ!%
    {%
        \expandafter\calx_lLGTva\the\numexpr##5##6%
        -((#3##1)*(#3##1)-2*((#3##1)*(#3##1)/2))*50000000%
        -((#3##1)*(#3##2)-10000*((#3##1)*(#3##2)/10000))*10000%
        -(#3##2)*(#3##2)/2-(#4##1##2)*(#4##3##4)/100000000%
        #6(#4##1##2)*(#4##1##2)/100000000*(#4##1##2)/300000000%
        \expandafter!\the\numexpr##3##4-(#3##1)*(#3##1)/2%
        -(#3##1)*(#3##2)/10000!##1##2!#5;%
    }%
}%
\calx_lLtmp{p2}{0000!0000!}{}{}{0}+%
\calx_lLtmp{n2}{9999!9999!}{9999-}{99999999-}{-1}-%
```

L = 32 (W = 40): step 1 -> Hs!Ls!u!w!tau5!tau6!lambda!mu! (s^2 = Hs 10^8 + Ls exactly, u = tau3, w = tau4); n: complements.

```
\expandafter\def\csname calx_lLGfp3\endcsname 0000!0000!#1!#2!#3!#4!#5!#6!#7!#8!%
   \calx_lLQ!\calx_lLQ!%
{%
    \expandafter\calx_lLGgp\the\numexpr#1*#1+2*#1*#2/10000\expandafter!%
    \the\numexpr(2*#1*#2-10000*(2*#1*#2/10000))*10000+#2*#2!%
    #1!#2!#3!#4!#5#6!#7#8!%
}%
\expandafter\def\csname calx_lLGfn3\endcsname 9999!9999!#1!#2!#3!#4!#5!#6!#7!#8!%
   \calx_lLQ!\calx_lLQ!%
{%
    \expandafter\calx_lLGgn\the\numexpr(9999-#1)*(9999-#1)%
      +2*(9999-#1)*(9999-#2)/10000\expandafter!%
    \the\numexpr(2*(9999-#1)*(9999-#2)-10000*(2*(9999-#1)*(9999-#2)/10000))%
      *10000+(9999-#2)*(9999-#2)\expandafter!%
    \the\numexpr9999-#1\expandafter!\the\numexpr9999-#2\expandafter!%
    \the\numexpr9999-#3\expandafter!\the\numexpr9999-#4\expandafter!%
    \the\numexpr99999999-#5#6\expandafter!\the\numexpr99999999-#7#8!%
}%
```

step 2: Hs!Ls!u!w!tau5!tau6!lambda!mu! -> P4!P3!P2!A!P5; #1 name, #2 pair complement prefix, #3 tokens writing A, #4 sign of t

```
\def\calx_lLtmp #1#2#3#4%
{%
  \def#1##1!##2!##3!##4!##5!##6!##7!##8!%
  {%
```

```
    \expandafter\calx_lLGTvb\the\numexpr#2##7%
      -(##1-2*(##1/2))*50000000-##2/2-##3*##5%
      -(##3*##6+##4*##5+##4*##6/10000)/10000%
      #4(##3*(##1/10000)/3%
         +(##3*(##1-10000*(##1/10000))+##4*(##1/10000))/30000)%
    \expandafter!\the\numexpr#2(##5*10000+##6)-##1/2%
    \expandafter!\the\numexpr#2(##3*10000+##4)%
    #3\the\numexpr#2##8%
      -(##2-2*(##2/2))*50000000%
      -((##3*##6+##4*##5-10000*((##3*##6+##4*##5+##4*##6/10000)/10000))%
        *10000+##4*##6)%
      -(##3*10000+##4)*##7/100000000%
      -(##5*10000+##6)*(##5*10000+##6)/200000000%
      -(##3*10000+##4)*##8/100000000/100000000%
      -(##5*10000+##6)*##7/100000000/100000000%
      #4((##3*(##1/10000)-3*(##3*(##1/10000)/3))*33333333%
        +((##3*(##1-10000*(##1/10000))+##4*(##1/10000))-30000*%
          ((##3*(##1-10000*(##1/10000))+##4*(##1/10000))/30000))*10000/3%
        +##4*(##1-10000*(##1/10000))/3%
        +(##3*10000+##4)*##2/300000000%
        +##1*(##5*10000+##6)/100000000%
        +##2*(##5*10000+##6)/100000000/100000000)%
      -##1*##1/400000000;%
  }%
}%
\calx_lLtmp\calx_lLGgp{}{\expandafter!\expandafter0\expandafter!}+%
\calx_lLtmp\calx_lLGgn{99999999-}%
   {\expandafter!\expandafter-\expandafter1\expandafter!}-%
```

tables: fetch + add, order Lv, Lu, Lt, Ls, Lb. Templates (shared with exp P17..P32): \calx_lLGT
↪ def <fetch><prefix><add>, \calx_lLGAdef <add><next> (L = 24), \calx_lLGBdef <add><next> (L = 32)
fetch: <accs>;<key>; -> <entry>\calx_lLQ<accs>;

```
\def\calx_lLGTdef #1#2#3%
   {\def#1##1;##2;{\expandafter\expandafter\expandafter#3%
      \csname#2##2\endcsname\calx_lLQ##1;}}%
```

add, L = 24: F1!c2!c3!c4!rest\calx_lLQ P4!P3!P2!A;

```
\def\calx_lLGAdef #1#2%
   {\def#1##1!##2!##3!##4!##5\calx_lLQ##6!##7!##8!##9;%
     {\expandafter#2\the\numexpr##6+##4\expandafter!\the\numexpr##7+##3%
      \expandafter!\the\numexpr##8+##2\expandafter!\the\numexpr##9+##1;}}%
```

add, L = 32: accs P4!P3!P2!A!P5; (c5 + P5 by \calx_lLGCe)

```
\def\calx_lLGBdef #1#2%
   {\def#1##1!##2!##3!##4!##5\calx_lLQ##6!##7!##8!##9!%
     {\expandafter#2\the\numexpr##6+##4\expandafter!\the\numexpr##7+##3%
      \expandafter!\the\numexpr##8+##2\expandafter!\the\numexpr##9+##1%
      \expandafter!\the\numexpr\calx_lLGCe##5\calx_lLQ}}%
\def\calx_lLGCe #1!#2\calx_lLQ#3;{#1+#3;}%
\calx_lLGTdef\calx_lLGTva{calx_lLv}\calx_lLGAva \calx_lLGAdef\calx_lLGAva\calx_lLGTua
\calx_lLGTdef\calx_lLGTua{calx_lLu}\calx_lLGAua \calx_lLGAdef\calx_lLGAua\calx_lLGTta
\calx_lLGTdef\calx_lLGTta{calx_lLt}\calx_lLGAta \calx_lLGAdef\calx_lLGAta\calx_lLGTsa
\calx_lLGTdef\calx_lLGTsa{calx_lLs}\calx_lLGAsa \calx_lLGAdef\calx_lLGAsa\calx_lLGTba
\calx_lLGTdef\calx_lLGTba{calx_lLb}\calx_lLGAba \calx_lLGAdef\calx_lLGAba\calx_lLGca
```

```
\calx_lLGTdef\calx_lLGTvb{calx_lLv}\calx_lLGAvb \calx_lLGBdef\calx_lLGAvb\calx_lLGTub
\calx_lLGTdef\calx_lLGTub{calx_lLu}\calx_lLGAub \calx_lLGBdef\calx_lLGAub\calx_lLGTtb
\calx_lLGTdef\calx_lLGTtb{calx_lLt}\calx_lLGAtb \calx_lLGBdef\calx_lLGAtb\calx_lLGTsb
\calx_lLGTdef\calx_lLGTsb{calx_lLs}\calx_lLGAsb \calx_lLGBdef\calx_lLGAsb\calx_lLGTbb
\calx_lLGTdef\calx_lLGTbb{calx_lLb}\calx_lLGAbb \calx_lLGBdef\calx_lLGAbb\calx_lLGcb
```

carry chains with E ln10 (limbs of floor(10^40 ln10)); the last pair is split (its column could leave the 10-digit range). Templates (shared with exp): \calx_lLGcadef / \calx_lLGcbdef <name><cont>{<params>} {<integer column prefix>}; params ##1..##4 P4!P3!P2!A (L24), ##1..##5 P4!P3!P2!A!P5 (L32), then E (##5 / ##6); output <cont>I!l1!..!ln! (L24: + tail #5; L32: + \calx_lLQ)

```
\def\calx_lLGcadef #1#2#3#4#5%
{%
  \def#1#3%
  {%
    \expandafter#2\the\numexpr#4 2*##5\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+3025*##5\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+##4+8509*##5\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+2994*##5\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+##3+456*##5\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+8401*##5\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+##2+7991*##5\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+4546*##5+##1/10000\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+8436*##5+##1-##1/10000*10000\relax!#5%
  }%
}%
\calx_lLGcadef\calx_lLGca\calx_lLNaat{#1!#2!#3!#4;#5;#6.}{}{}% P4!P3!P2!A;E;P.
\def\calx_lLGcbdef #1#2#3#4%
{%
  \def#1#3%
  {%
    \expandafter#2\the\numexpr#4 2*##6\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+3025*##6\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+##4+8509*##6\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+2994*##6\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+##3+456*##6\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+8401*##6\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+##2+7991*##6\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+4546*##6+##1/10000\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+8436*##6+##1-##1/10000*10000\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+4207*##6+##5/10000\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+6011*##6+##5-##5/10000*10000\relax!\calx_lLQ
  }%
}%
\calx_lLGcbdef\calx_lLGcb\calx_lLGo{#1!#2!#3!#4!#5;#6;}{}% P4!P3!P2!A!P5;E; (P. -> LGo)
```

output: L = 24 by the P16 template, L = 32 by calxtrig's \calx_trsa

```
\calx_lLNdef\calx_lLNaat\calx_lLNap\calx_lLNan{#1!#2!#3!#4!#5!#6!#7!#8!#9!}{4}%
   {1#8#9!1#6#7!1#4#5!1#2#3!}%
   {\the\numexpr199999999-#8#9!\the\numexpr199999999-#6#7!%
    \the\numexpr199999999-#4#5!\the\numexpr199999999-#2#3!}%
   {#2!#3!#4!#5!#6!#7!#8!#9!}{5.4.}%
```

generic (P > 24): I!l1!..!ln!\calx_lLQ P. (P from the tail), m = W/8 = ceil(P/8) + 1 pairs

```
\def\calx_lLGo #1!#2\calx_lLQ#3.%
   {\expandafter\calx_lLGoq\the\numexpr(#3+3)/8+2\expandafter.%
    \the\numexpr(#3+3)/8+1.#1!#2\calx_lLQ}%
\def\calx_lLGoq #1.#2.#3!#4\calx_lLQ%
{%
    \ifnum#3<\xint_c_
      \xint_dothis{\expanded{\calx_trsa#1.#2.\calx_tbn-!%
        \the\numexpr\xint_c_e_viii+#3!#4}}\fi
    \xint_orthat{\expanded{\calx_trsa#1.#2.\calx_tbp+!\expandafter\xint_gobble_i
      \the\numexpr\xint_c_e_viii+#3!#4}}%
}%
```

unity zone |x - 1| < 10^-4: y = delta G, G = sum_k (-delta)^k/(k+1), Horner by the series kernel \calx_tZ (calxtrig), decreasing widths: level k has scale 10^(4(A-k)), T_k = C_k - U' T_(k+1) 10^(-4b), U' = 10^4 delta (b_0 = M - 1 limbs, U_k its top b_k = min(b_0, A-k-1) limbs), C_k = floor(10^(4(A-k))/(k+1)) (\calx_lcdl<k>), T_(A-1) = C_(A-1) (one limb), levels k = A-2..1 (b_k, m = A-k-1, n = A-k), level 0 with C_0 = 10^(4A) (n = A+1 limbs, output negated), then the exact product level (\calx_tZlevp: b_0 by A+1 limbs, all columns, floor carries). A = ceil((P+4)/4) up to P = 64, ceil((P+5)/4) above; error of T_0 at most 3A - 4 + 1/(A+1) units 10^(-4A) (each level: |err| < 0.503). y = (x - 1) T_0 10^(-4A): exact product by one more kernel level, one rounding. Schedule \calx_lZul<P> (P = 17..112): {C_(A-1)}{<level descriptors>\calx_lLWk}.

```
\def\calx_lLWrv #1\relax{\calx_lLWrw#1\relax\relax}% reverse l!..l! (load time)
\def\calx_lLWrw #1!#2\relax#3\relax%
   {\if\relax\detokenize{#2}\relax#1!#3\else\calx_lLWrw#2\relax#1!#3\relax\fi}%
\def\calx_lLWzr #1.{\ifnum#1>\xint_c_ 0!\expandafter\calx_lLWzr
   \the\numexpr#1-1\expandafter.\fi}%
\def\calx_lLWmin #1#2{\ifnum#1<\numexpr#2\relax#1\else#2\fi}%
\def\calx_lLWlev #1.#2.#3.#4{\calx_tZlev{#1}{#2}{#3}{#4}}%
```

coefficient hook: log, log1p: 1/(k+1); expm1 sets it to exp's \calx_lexgK (1/(k+1)!) while building its schedules

```
\def\calx_lLWco{\calx_ttcoef{calx_lcdl}}%
\def\calx_lLWbl #1.%
{%
    \ifnum#1>\xint_c_
      \expandafter\calx_lLWlev\the\numexpr\calx_lLWmin\calx_lLWm{\calx_lLWa-#1-1}%
        \expandafter.\the\numexpr\calx_lLWa-#1-1\expandafter.%
        \the\numexpr\calx_lLWa-#1.%
        {\expandafter\calx_lLWrv\expanded{\calx_lLWco{#1}{\calx_lLWa-#1}%
          {\calx_lLWa-#1}\calx_ttq}\relax}\calx_tZu
      \expandafter\calx_lLWbl\the\numexpr#1-1.%
    \else
      \expandafter\calx_lLWlev\the\numexpr\calx_lLWmin\calx_lLWm{\calx_lLWa-1}%
        \expandafter.\the\numexpr\calx_lLWa-1\expandafter.%
        \the\numexpr\calx_lLWa+1.%
        {1!\expandafter\calx_lLWzr\calx_lLWa.}\calx_tZun
      \calx_lLWfin
    \fi
}%
\def\calx_lbuildZul #1%
{%
    \edef\calx_lLWa{\the\numexpr(#1+\ifnum#1>64 6\else5\fi)/4}%
    \edef\calx_lLWm{\the\numexpr2*((#1+3)/8)-1}%
    \XINT_global\expandafter\edef\csname calx_lZul#1\endcsname
```

```
    {%
      {\expandafter\calx_tZhd\expanded{\calx_ttcoef{calx_lcdl}{\calx_lLWa-1}{1}{1}%
       \calx_ttq}\relax}%
      {\expandafter\calx_lLWbl\the\numexpr\calx_lLWa-2.}%
    }%
}%
```

after level 0 (a hook; log1p builds its levels without it)

```
\def\calx_lLWfinish{\calx_tZlevp\calx_lLWm{\the\numexpr\calx_lLWa+1}\calx_tZuf
   \noexpand\calx_lLWk}%
```

U' = 10^4 (x - 1), limbs MS first. E = 0 (x = 1.0000ddd..): limb j = digits 2-4 of m_j and digit 1 of m_(j+1), j = 2..M (\calx_lLWsh); E = -1 (x = 0.9999ddd..): m_2 - 10^4, m_3, .., m_M. Then the kernel's U list (\calx_tZbt: negated, balanced, top limb kept whole).

```
\def\calx_lLWsh #1#2#3#4!#5%
   {\calx_tZgob#5\calx_lLWse\calx_tZq#2#3#4#5!\calx_lLWsh#5}%
\def\calx_lLWse\calx_tZq#1#2#3\calx_tZq!\calx_lLWsh\calx_tZq!{#1#2#30!\calx_tZq!}%
\def\calx_lLWs #1\calx_lLQ!#2%
   {\expandafter\calx_lLWu\expanded{\calx_lLWsh#1\calx_tZq!}{#2}}%
\def\calx_lLWt #1\calx_lLQ!{\calx_lLWu#1\calx_tZq!}%
\def\calx_lLWu #1\calx_tZq!#2%
{%
    \expandafter\expandafter\expandafter\calx_tZst
    \csname calx_lZul\the\numexpr#2\expandafter\endcsname
    \expandafter{\expanded{\calx_tZbt#1\calx_tZq!}}{#2}%
}%
```

final: {U}x_top.<limbs>\calx_tZq{P}: the exact product N = U'_int T_0,int on K = b_0 + A + 1 limbs (floor form, x_top = top limb, negative iff delta < 0), y = N 10^(-4K): \calx_trsa (calxtrig) with pairs MS first, a zero limb appended if K (i.e. A) is odd, complement conversion for N < 0 (|N| - 1 unit, harmless). N = 0 (x = 1): \calx_trsa returns F+1.<..>, impossible otherwise (|y| > 10^(-4K)): F+0.0. #4 = P (\calx_ldigits or digits); A = (P + 5 or 6)/4, c = (2 ((P+3)/8) + A)/2 computed here.

```
\def\calx_lLWk #1#2.#3\calx_tZq#4%
   {\expandafter\calx_lLWkk\the\numexpr(2*((#4+3)/8)+(#4+\ifnum#4>64 6\else
    5\fi)/4)/2\expandafter.\expanded{\ifnum#2<\xint_c_ \the\numexpr10000+#2\else
    \expandafter\xint_gobble_i\the\numexpr10000+#2\fi!%
    #3\ifodd\numexpr(#4+\ifnum#4>64 6\else5\fi)/4\relax 0000!\fi}\relax#2.}%
\def\calx_lLWkk #1.#2\relax#3.%
{%
    \expandafter\calx_lLWz\expanded{%
      \ifnum#3<\xint_c_ \expandafter\calx_lLWkn\else\expandafter\calx_lLWkp\fi
      #3.#1.#2}%
}%
\def\calx_lLWkp #1.#2.{\calx_trsa#2.#2.\calx_tbp+}%
\def\calx_lLWkn #1.#2.{\calx_trsa#2.#2.\calx_tbn-}%
\begingroup\catcode`F=12
\gdef\calx_lLWz F#1#2.%
   {\ifnum#2=\xint_c_i \expandafter\calx_lLWzz\fi F#1#2.}%
\gdef\calx_lLWzz F#1#2.#3.#4!{F+0.0.}%
\endgroup
```

33.8. Natural logarithm, P33 and above

L = 8 ceil(P/8) <= 112, W = L + 8 <= 120, n = W/4 limbs, m = n/2 pairs. After stage 8, |t| < 5.061e-9: y_s = ln(1+t) = t G(t), G = sum (-t)^k/(k+1), by the series kernel \calx_tZ (calxtrig) with U'' = 10^8 t (n-2 limbs v3..vn; t < 0: v3 - 10^4, v4, ..): level k has scale 10^(4(n-2-2k)), T_k = C_k - U'' T_(k+1) 10^(-4b), b = m = n-4-2k (two limbs less per level), levels k = (n-6)/2..1 from T = C_((n-4)/2) (two limbs); level 0: T_0 = G (C_0 = 10^(4(n-2)), n-1 limbs, output negated); final level: y_s 10^W = 0 + U'' T_0 10^(-4(n-2)) (b = n-2, m = n-1, n limbs, floor carries), written A!P2!..!Pm! with A = floor(10^8 y_s) (0 or -1). Error: per level |err| < 0.503 unit (total within ~4.1 units of 10^-W). Schedules \calx_lZlg<g> (keyed by the layout g = (P-5)/8 = 4..13): {c1.\calx_tZn c2}{<descriptors>\calx_lLGFk}.

```
\def\calx_lLGt #1!#2!{#2.\noexpand\calx_tZn#1}% LS first pair -> T items
\def\calx_lLGsl #1.%
{%
    \ifnum#1>\xint_c_
      \expandafter\calx_lLWlev\the\numexpr\calx_lLWa-4-2*#1\expandafter.%
        \the\numexpr\calx_lLWa-4-2*#1\expandafter.%
        \the\numexpr\calx_lLWa-2-2*#1.%
        {\expandafter\calx_lLWrv\expanded{\calx_ttcoef{calx_lcdl}{#1}{\calx_lLWa-2-2*#1}%
          {\calx_lLWa-2-2*#1}\calx_ttq}\relax}\calx_tZu
      \expandafter\calx_lLGsl\the\numexpr#1-1.%
    \else
      \expandafter\calx_lLWlev\the\numexpr\calx_lLWa-4\expandafter.%
        \the\numexpr\calx_lLWa-4\expandafter.\the\numexpr\calx_lLWa-1.%
        {1!\expandafter\calx_lLWzr\the\numexpr\calx_lLWa-2.}\calx_tZun
      \expandafter\calx_lLGsv\the\numexpr\calx_lLWa-2\expandafter.%
        \the\numexpr\calx_lLWa-1\expandafter.\calx_lLWa.%
        {\expandafter\calx_lLWzr\calx_lLWa.}\calx_tZuf
      \noexpand\calx_lLGFk
    \fi
}%
\def\calx_lLGsv #1.#2.#3.#4{\calx_tZlevx\calx_tZvn{#1}{#2}{#3}{#4}}%
\def\calx_lbuildZlg #1%
{%
    \edef\calx_lLWa{\the\numexpr2*#1+4}% n
    \XINT_global\expandafter\edef\csname calx_lZlg#1\endcsname
    {%
      {\expandafter\calx_lLGt\expanded{\calx_ttcoef{calx_lcdl}{(\calx_lLWa-4)/2}{2}{2}%
        \calx_ttq}}%
      {\expandafter\calx_lLGsl\the\numexpr(\calx_lLWa-6)/2.}%
    }%
}%
```

final, after \calx_lLGf<g> (v0 read): v1!..!vn!\calx_lLQ!\calx_lLQ!J8;..;key;E;P. U'' limbs MS first (t < 0: top v3 - 10^4), then the kernel with the schedule \calx_lZlg<g> (g baked in \calx_lLGfp<g>, \calx_lLGfn<g>)

```
\def\calx_lLGFs #1\calx_tZq!#2.%
{%
    \expandafter\expandafter\expandafter\calx_tZst
    \csname calx_lZlg#2\expandafter\endcsname
    \expandafter{\expanded{\calx_tZbt#1\calx_tZq!}}%
}%
```

{U}x_top.<n-1 limbs dddd!>\calx_tZq -> \calx_lLGTv A!P2!..!Pm!;

```
\def\calx_lLGFk #1#2.#3\calx_tZq%
   {\expandafter\calx_lLGFa\expanded{{#2}#3}\calx_tZq!\calx_tZq!}%
\def\calx_lLGFa #1#2!#3\calx_tZq!\calx_tZq!%
{%
    \expandafter\calx_lLGTv\the\numexpr#1*10000+#2\expandafter!%
    \expanded{\calx_lLGFb#3\calx_tZq!\calx_tZq!};%
}%
\def\calx_lLGFb #1!#2!{\calx_tZgob#1\calx_lLGFc\calx_tZq#1#2!\calx_lLGFb}%
\def\calx_lLGFc\calx_tZq#1\calx_lLGFb{}%
```

tables: A!P2!..!Pm!;key; -> sums (zip with the entry, m <= 14 items; a dummy item 0! lets the sentinel step read past the 14th)

```
\def\calx_lLGTx #1#2#3;#4;%
   {\expandafter\expandafter\expandafter\calx_lLGTw\csname#2#4\endcsname
    \relax#1{#3}}%
\def\calx_lLGTw #1\relax#2#3%
   {\expandafter#2\expanded{\calx_lLGad#3\calx_lLQ!;#10!\relax};}%
\def\calx_lLGad #1!#2;#3!%
   {\calx_lLGgt#1\calx_lLGae\calx_lLQ\the\numexpr#1+#3!\calx_lLGad#2;}%
\def\calx_lLGae\calx_lLQ#1\relax{}%
\def\calx_lLGTv{\calx_lLGTx\calx_lLGTu{calx_lLv}}%
\def\calx_lLGTu{\calx_lLGTx\calx_lLGTt{calx_lLu}}%
\def\calx_lLGTt{\calx_lLGTx\calx_lLGTs{calx_lLt}}%
\def\calx_lLGTs{\calx_lLGTx\calx_lLGTb{calx_lLs}}%
\def\calx_lLGTb{\calx_lLGTx\calx_lLGch{calx_lLb}}%
```

carry chain with E ln10: A!P2!..!Pm!;E; ; ln10 = 2.3025 8509 then the limbs of \calx_lLzl; pair P_i whole in its low limb column

```
\def\calx_lLGch{\expandafter\calx_lLGci\calx_lLzl\relax}%
```

template (shared with exp): \calx_lLGcidef <name><cont>{<params>} {<integer column prefix>}; params ##1 ln10 limbs, ##2 A, ##3 pairs, ##4 E

```
\def\calx_lLGcidef #1#2#3#4%
{%
  \def#1#3%
  {%
    \expandafter#2\the\numexpr#4 2*###4\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+3025*###4\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+##2+8509*###4\calx_lLGcy##3\calx_lLQ!;##4;##10!0!\relax
  }%
}%
\calx_lLGcidef\calx_lLGci\calx_lLGo{#1\relax#2!#3;#4;}{}%
\def\calx_lLGcy #1!#2;#3;#4!#5!%
{%
    \calx_lLGgt#1\calx_lLGce\calx_lLQ\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+#4*#3\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+#1+#5*#3\calx_lLGcy#2;#3;%
}%
\def\calx_lLGce\calx_lLQ#1\relax{\relax!\calx_lLQ}%
\def\calx_lLtmp #1%
{%
    \expandafter\def\csname calx_lLGfp#1\endcsname
       0000!0000!##1\calx_lLQ!\calx_lLQ!%
       {\calx_lLGFs##1\calx_tZq!#1.}%
```

```
    \expandafter\def\csname calx_lLGfn#1\endcsname
       9999!9999!##1!##2\calx_lLQ!\calx_lLQ!%
       {\expandafter\calx_lLGFs\the\numexpr##1-10000!##2\calx_tZq!#1.}%
}%
\calx_lLtmp4\calx_lLtmp5\calx_lLtmp6\calx_lLtmp7\calx_lLtmp8\calx_lLtmp9%
\calx_lLtmp{10}\calx_lLtmp{11}\calx_lLtmp{12}\calx_lLtmp{13}%
\let\calx_lLtmp\relax
```

dispatch: P17..P100 (LG), P1..P8 as before, P9..P16 by the P16 code (16-digit grid, input padded with zeros, one rounding to P digits)

```
\expandafter\let\csname calx_lLGtagF-\expandafter\endcsname
   \csname calx_llgHsF-\endcsname
\expandafter\let\csname calx_lLGtagI+\expandafter\endcsname
   \csname calx_llgHsI+\endcsname
\expandafter\let\csname calx_lLGtagI-\expandafter\endcsname
   \csname calx_llgHsI-\endcsname
\expandafter\let\csname calx_lLGtagI?\expandafter\endcsname
   \csname calx_llgHsI?\endcsname
\def\calx_llga_high #1#2{\csname calx_lLGtag#1#2\endcsname}%
\def\calx_llga_low #1#2{\csname calx_llgUtag#1#2\endcsname}%
\def\calx_llga_mid #1#2{\csname calx_lLtag#1#2\endcsname}%
```

33.9. \calxFpLogOnePlus, \calxRawLogOnePlus

log1p(x) = log(1 + x). Input rounded to P digits; x <= -1: errors as log; x = 0: 0.
A. |x| >= 0.1: r = round_P(1+x), d = (1+x) - r (exact, one digit), y = L(r) + d/r, L(r) the UNROUNDED internal log (P16 code for P <= 16), d/r by \CALX_fdiv at 9 digits, exact sum, one rounding.
B. 10^-4 <= |x| < 0.1: V = 10^W (1+x) exact (W = 24, or 8 ceil(P/8) + 8 for P > 16), the staged unity path of log (\calx_lLsAa, \calx_lLGsa).
C. |x| < 10^-4: x G(x) by the kernel (\calx_llpC below).

```
\def\calx_lLogOnePlus{\romannumeral0\expandafter\calx_llpa
    \romannumeral0\CALX_fround{\calx_ldigits}}%
\def\calxRawLogOnePlus #1{\romannumeral0\CALX_ftoraw{\calxFpLogOnePlus{#1}}}%
\def\calx_llpnfl #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\calxFpLogOnePlus#3}}%
}%
\def\calx_llpnex #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\calxRawLogOnePlus#3}}%
}%
```

expression function log1p: the only names with log1p (a digit: \csname)

```
\expandafter\let\csname XINT_flexpr_func_log1p\endcsname\calx_llpnfl
\expandafter\let\csname XINT_expr_func_log1p\endcsname\calx_llpnex
\def\calx_llpa #1#2{\csname calx_llptag#1#2\endcsname}%
\expandafter\let\csname calx_llptagI+\expandafter\endcsname
   \csname calx_llgHsI+\endcsname
\expandafter\let\csname calx_llptagI?\expandafter\endcsname
```

```
   \csname calx_llgHsI?\endcsname
\expandafter\def\csname calx_llptagI-\endcsname#1\xint:%
   {\CALX_nsig{InvalidOperation}{log1p(-inf)}{}}%
\begingroup\catcode`F=12
\expandafter\gdef\csname calx_llptagF+\endcsname #1.#2.#3\xint:%
{%
    \ifnum#1=\xint_c_ \xint_dothis{ F+0.0.}\fi
    \xint_orthat{\expandafter\calx_llpb
      \the\numexpr#1-\xint_c_i+\xint_c_viii*#2.+#1.#2.#3\xint:}%
}%
```

x < 0: E >= 0 means x <= -1 (\calx_llpm: 1 + x = 0 or < 0)

```
\expandafter\gdef\csname calx_llptagF-\endcsname #1.#2.%
{%
    \ifnum\numexpr#1-\xint_c_i+\xint_c_viii*#2>-\xint_c_i
      \xint_dothis\calx_llptag_negative_domain\fi
    \xint_orthat\calx_llptag_negative_regular #1.#2.%
}%
\gdef\calx_llptag_negative_domain #1.#2.#3\xint:%
   {\expandafter\calx_llpm\romannumeral0\CALX_faddx
        {F+1.0.100000001!}{F-#1.#2.#3}\xint:}%
\gdef\calx_llptag_negative_regular #1.#2.#3\xint:%
   {\expandafter\calx_llpb
      \the\numexpr#1-\xint_c_i+\xint_c_viii*#2.-#1.#2.#3\xint:}%
\gdef\calx_llpm F#1#2.#3\xint:%
{%
    \ifnum#2=\xint_c_ \xint_dothis{\CALX_nsig{DivisionByZero}{log1p(-1)}{-}}\fi
    \xint_orthat{\CALX_nsig{InvalidOperation}{log1p(x), x < -1}{}}%
}%
```

classes by the decimal exponent E of x: E >= -1, E <= -5, else B

```
\gdef\calx_llpb #1.#2#3.%
{%
    \ifnum#1>-\xint_c_ii \xint_dothis\calx_llpb_large\fi
    \ifnum#1<-\xint_c_iv \xint_dothis\calx_llpb_small\fi
    \xint_orthat\calx_llpb_regular #1.#2#3.%
}%
\gdef\calx_llpb_large #1.#2#3.#4.#5\xint:{\calx_llpA#2#3.#4.#5\xint:}%
\gdef\calx_llpb_small #1.#2#3.#4.#5\xint:{\calx_llpC lpl;#2#3.#4.#5\xint:}%
\gdef\calx_llpb_regular #1.#2#3.#4.#5\xint:{\calx_llpB#2#3.#4.#5\xint:}%
```

A: |x| >= 0.1

```
\gdef\calx_llpA #1\xint:%
   {\expandafter\calx_llpAa
    \romannumeral0\CALX_faddx{F+1.0.100000001!}{F#1}\xint:}%
\gdef\calx_llpAa #1\xint:%
   {\expandafter\calx_llpAb\romannumeral0\CALX_fround{\calx_ldigits}{#1}#1\xint:}%
\gdef\calx_llpAb #1\xint:#2\xint:%
   {\expandafter\calx_llpAc\romannumeral0\CALX_fsubx{#2}{#1}\xint:#1\xint:}%
\gdef\calx_llpAc F#1#2.%
{%
    \ifnum#2=\xint_c_ \xint_dothis\calx_llpAc_zero\fi
    \xint_orthat\calx_llpAc_regular F#1#2.%
}%
```

```
\gdef\calx_llpAc_zero F#1#2.#3\xint:F#4\xint:%
   {\CALX_fnround{\calx_ldigits}{\calx_llpL#4\xint:}}%
\gdef\calx_llpAc_regular F#1#2.#3\xint:F#4\xint:%
   {\CALX_fnround{\calx_ldigits}{\CALX_faddx{\calx_llpL#4\xint:}%
      {\CALX_fdiv{9}{F#1#2.#3}{F#4}}}}%
\endgroup
```

unrounded internal log of r = +<D>.<e>.<blocks> (bodies of \calx_lLtb, \calx_lLGtb without the rounding) \calx_llpLP P.+<D>.<e>.<blocks>\xint: at precision P (\calx_ldigits or digits; P <= 112; P <= 16: the P16 code), shared with pow

```
\def\calx_llpL{\calx_llpLP\calx_ldigits.}%
\def\calx_llpLP #1.+#2.#3.#4\xint:%
{%
    \expandafter\calx_llpLb\the\numexpr#1\expandafter.%
    \the\numexpr(#2+\xint_c_iii)/\xint_c_viii
    \expandafter.\the\numexpr#2-\xint_c_i+\xint_c_viii*#3.#2.#4\xint:%
}%
\def\calx_llpLb #1.%
   {\ifnum#1>16 \expandafter\calx_lLGL\else\expandafter\calx_llpLd\fi#1.}%
\def\calx_llpLd #1.{\calx_llpLs}%
\def\calx_llpLs #1.#2.#3.#4\xint:%
   {\csname calx_lLin\romannumeral#1\endcsname\calx_lLM#4\xint:#2.#3.}%
```

B: 10^-4 <= |x| < 0.1; Bw = W/8 blocks, nb blocks of N

```
\def\calx_llpB #1#2.#3.#4\xint:%
{%
    \expandafter\calx_llpBa\the\calx_llpBw
    \expandafter.\the\numexpr(#2+\xint_c_iii)/\xint_c_viii.#1#3.#4\xint:%
}%
\def\calx_llpBa #1.#2.#3#4.#5\xint:%
{%
    \calx_lLfr\calx_ldigits{\csname calx_llpB\calx_llpBkind#3%
      \endcsname{\expandafter\calx_llpzb\the\numexpr#1+#4.#5%
      \expandafter\calx_llpzb\the\numexpr-#4-#2.}}%
}%
\def\calx_llpzb #1.{\ifnum#1>\xint_c_ 100000000!\expandafter\calx_llpzb
   \the\numexpr#1-1\expandafter.\fi}%
```

the selectors receive V = v0!v1!..: v0 = 1, x 10^W; v0 = 0, 10^W - |x| 10^W

```
\expandafter\def\csname calx_llpBs+\endcsname #1%
   {\expandafter\calx_lLsAa\expandafter1\expandafter!\expanded
    {\expandafter\calx_tshr\expanded{#1}\calx_tZq!\relax}0;0;}%
\expandafter\def\csname calx_llpBs-\endcsname #1%
   {\expandafter\calx_lLsAa\expandafter0\expandafter!\expanded
    {\expandafter\calx_tshr\expanded{\expandafter\calx_llpcz\expanded{#1}%
     \calx_tZq!}\calx_tZq!\relax}0;0;}%
\expandafter\def\csname calx_llpBg+\endcsname #1%
   {\expandafter\calx_lLGsa\expandafter1\expandafter!\expanded
    {\expandafter\calx_tshr\expanded{#1}%
     \calx_tZq!\relax}\calx_lLQ!\calx_lLQ!0;0;\calx_ldigits.}%
\expandafter\def\csname calx_llpBg-\endcsname #1%
   {\expandafter\calx_lLGsa\expandafter0\expandafter!\expanded
    {\expandafter\calx_tshr\expanded{\expandafter\calx_llpcz\expanded{#1}%
     \calx_tZq!}\calx_tZq!\relax}\calx_lLQ!\calx_lLQ!0;0;\calx_ldigits.}%
```

```
    10^(8k) - X on blocks 1<8 digits>! LS first (inside \expanded): zero blocks stay (\calx_llpcz),
    the first nonzero one gives 10^8 - b, the others 10^8 - 1 - b (\calx_llpco)
1092 \def\calx_llpcz #1#2!{\calx_tZgob#1\calx_llpce\calx_tZq\calx_llpcy#2!}%
1093 \def\calx_llpcy #1!{\ifnum#1=\xint_c_ 100000000!\expandafter\calx_llpcz\else
1094    \the\numexpr200000000-#1!\expandafter\calx_llpco\fi}%
1095 \def\calx_llpco #1#2!{\calx_tZgob#1\calx_llpce\calx_tZq\calx_llpcb#2!}%
1096 \def\calx_llpcb #1!{\the\numexpr199999999-#1!\calx_llpco}%
1097 \def\calx_llpce\calx_tZq#1!{}%
    C: |x| < 10^-4: y = x G(x), G = sum (-x)^k/(k+1), by the kernel with the levels of log's unity zone
    for P' = max(P,17) but b_k = A-k-1 (\calx_llpS: U' = 10^4 x may have more limbs than log's delta),
    then an exact product level on X' (x = X' 10^(-8z), as calxtrig's sin) and calxtrig's finisher \
↪   calx_tshx (kind l: one rounding to P digits). U' list: x > 0: limbs of x (2z zero limbs, then
    those of N) without the first one (0000); x < 0: limbs of 1 + x, 9999 dropped, the next one minus
    10^4 (as log's E = -1 unity entry); a zero limb appended (\calx_tZbt needs two items); z > 14: U'
    = 0; then A zero limbs (the levels read up to A+1 items). kind;s D.e.blocks, kind lp (log1p) or em
    (expm1: schedules \calx_lemS, U lists \calx_lemU<s> = \calx_llpU<-s>)
1098 \def\calx_llpC #1;#2#3.#4.#5\xint:%
1099 {%
1100     \expandafter\calx_llpCa\the\numexpr-#4-(#3+\xint_c_iii)/\xint_c_viii
1101     \expandafter.\the\calx_llpP.#1;#2{#5}%
1102 }%
1103 \def\calx_llpCa #1.#2.#3#4#5;#6#7%
1104 {%
1105     \csname calx_l#3#4S\expandafter\endcsname\expanded{{\ifnum#1>14 0!0!0!0!%
1106       \else\csname calx_l#3#4U#6\endcsname#1.{#7}\fi
1107       \expandafter\calx_tZzr\the\numexpr(#2+\ifnum#2>64 6\else5\fi)/4.}%
1108       \noexpand\calx_llpF
1109       {\expandafter\calx_tZbt\expanded{\calx_tshr#7\calx_tZq!\relax
1110        \expandafter\calx_tZzr\the\numexpr2*((#2+10)/8).}\calx_tZq!}}#1.#6#5%
1111 }%
1112 \expandafter\def\csname calx_llpU+\endcsname #1.#2%
1113 {%
1114     \expandafter\calx_tZbt\expanded{\expandafter\calx_llpdr\expanded
1115       {\expandafter\calx_tZzr
1116        \the\numexpr2*#1.\calx_tshr#2\calx_tZq!\relax}0!}\calx_tZq!%
1117 }%
1118 \expandafter\def\csname calx_llpU-\endcsname #1.#2%
1119 {%
1120     \expandafter\calx_tZbt\expanded{\expandafter\calx_llpdn\expanded
1121       {\expandafter\calx_tshr\expanded{\expandafter\calx_llpcz\expanded
1122       {#2\calx_llpzb#1.}\calx_tZq!}\calx_tZq!\relax}0!}\calx_tZq!%
1123 }%
1124 \def\calx_llpdr #1!{}%
1125 \def\calx_llpdn #1!#2!{\the\numexpr#2-10000!}%
    load time, P' = 17..100: \calx_llpS{U list} (C_(A-1), levels A-2..0, level 0 negated), \calx_llp
↪   F{U}<T_0>\calx_tZq{X' list}: exact product level X'_int T_0 (b_p = 2 ceil((P'+7)/8) limbs of X',
    A+1 of T_0), then \calx_tshx with <pairs>.<m0>.{0000!}{post} (b_p + A fractional limbs)
1126 \def\calx_lbuildlp #1%
1127 {%
1128     \edef\calx_lLWa{\the\numexpr(#1+\ifnum#1>64 6\else5\fi)/4}%
1129     \edef\calx_lLWm{\the\numexpr\calx_lLWa-1}%
1130     \edef\calx_lLWp{\the\numexpr2*((#1+10)/8)}%
```

```
    \XINT_global\edef\calx_llpS##1%
    {%
      \noexpand\calx_tZd\expandafter\calx_tZhd\expanded{\calx_ttcoef{calx_lcdl}%
        {\calx_lLWa-1}{1}{1}\calx_ttq}\relax.\noexpand\calx_tZq{##1}%
      \expandafter\calx_lLWbl\the\numexpr\calx_lLWa-2.%
    }%
    \XINT_global\edef\calx_llpF##1##2\calx_tZq##3%
    {%
      \noexpand\calx_tZd##2\noexpand\calx_tZq{##3}%
      \calx_tZlevp{\calx_lLWp}{\the\numexpr\calx_lLWa+1\relax}\calx_tZuf
      \noexpand\calx_tshx{\ifodd\calx_lLWa\space
        \the\numexpr(\calx_lLWp+\calx_lLWa+3)/2.%
        \the\numexpr(\calx_lLWp+\calx_lLWa+1)/2.%
        {0000!}{0000!}\else
        \the\numexpr(\calx_lLWp+\calx_lLWa+2)/2.%
        \the\numexpr(\calx_lLWp+\calx_lLWa)/2.%
        {0000!}{}\fi}%
    }%
}%
```

33.10. \calxFpLogTen, \calxRawLogTen

log10(x). y = L(x) k_P, L(x) the unrounded internal ln (\calx_llpL of log1p: the P16 code for P <= 16), k_P = round_(P+12)(1/ln10) (\calx_lltk<P>, load time), exact product (\CALX_fmulx), one rounding. x = 1: 0; x = 10^k: k.

```
\def\calx_lLogTen{\romannumeral0\expandafter\calx_llta
    \romannumeral0\CALX_fround{\calx_ldigits}}%
\def\calxRawLogTen #1{\romannumeral0\CALX_ftoraw{\calxFpLogTen{#1}}}%
\def\calx_lltnfl #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\calxFpLogTen#3}}%
}%
\def\calx_lltnex #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\calxRawLogTen#3}}%
}%
```

expression function log10: the only names with log10 (a digit: \csname)

```
\expandafter\let\csname XINT_flexpr_func_log10\endcsname\calx_lltnfl
\expandafter\let\csname XINT_expr_func_log10\endcsname\calx_lltnex
\def\calx_llta #1#2{\csname calx_lltt#1#2\endcsname}%
\expandafter\let\csname calx_llttI+\expandafter\endcsname
   \csname calx_llgHsI+\endcsname
\expandafter\let\csname calx_llttI?\expandafter\endcsname
   \csname calx_llgHsI?\endcsname
\expandafter\def\csname calx_llttI-\endcsname#1\xint:%
   {\CALX_nsig{InvalidOperation}{log10(-inf)}{}}%
\expandafter\def\csname calx_llttF-\endcsname#1\xint:%
   {\CALX_nsig{InvalidOperation}{log10(x), x < 0}{}}%
```

```
\expandafter\def\csname calx_llttF+\endcsname #1.#2.#3\xint:%
{%
    \ifnum#1=\xint_c_ \xint_dothis{\CALX_nsig{DivisionByZero}{log10(0)}{-}}\fi
    \xint_orthat{\CALX_fnround{\calx_ldigits}%
      {\CALX_fmulx{\calx_llpL+#1.#2.#3\xint:}%
      {\calx_lltk}}}%
}%
```

k_P, P = 1..100, from 1/ln10 truncated to 130 digits (calxlogdata.tex)

33.11. \calxFpExp, \calxRawExp

Exponential, P9..P16. x rounded to P+14 digits on entry, sliced at 10^-24. k = ceil(Z/10^5), Z = nint(10^5 |x|/ln10) (approximate), bucket i of -r0 = k ln10 - x; buckets 1..98: a = A[i], c = nint(200000/(2a+1)), constant ln(10^4/c); bucket 0: c = 10^4, no constant; bucket 99: c = 1000, constant ln10 (equivalent to k = nint(x/ln10)): r1 = x - k ln10 + constant, |r1| < 0.0488, R1 = 10^10 r1 in one \numexpr; stages k = 2, 4, 6, 8 on the short residual R (tables Ls, Lt, Lu, Lv: -ln(1 + j 10^-k)); t on the grid 10^-24 (|t| < 5.5e-9), U = 10^24 (e^t - 1), G = c (100+j2)(10^4+j4) (10^6+j6)(10^8+j8) exact, Y' = G + G U 10^-24: exp x = 10^(k-24) Y', one rounding to P digits.

```
\let\calx_lexQ\relax
\def\calx_lExp{\romannumeral0\expandafter\calx_lexa
    \romannumeral0\CALX_fround{\calx_ldigits+14}}%
\def\calxRawExp #1{\romannumeral0\CALX_ftoraw{\calxFpExp{#1}}}%
\def\calx_lexnfl #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\calxFpExp#3}}%
}%
\def\calx_lexnex #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\calxRawExp#3}}%
}%
\let\XINT_flexpr_func_exp\calx_lexnfl
\let\XINT_expr_func_exp\calx_lexnex
\def\calx_lexa #1#2{\csname calx_lext#1#2\endcsname}%
\begingroup\catcode`F=12
\expandafter\gdef\csname calx_lextI+\endcsname#1\xint:{ I+}%
\expandafter\gdef\csname calx_lextI-\endcsname#1\xint:{ F+0.0.}%
\expandafter\gdef\csname calx_lextI?\endcsname#1\xint:{ I?}%
\expandafter\gdef\csname calx_lextF+\endcsname{\calx_lexb ex;+}%
\expandafter\gdef\csname calx_lextF-\endcsname{\calx_lexb ex;-}%
```

kind;s D, e, blocks (kind: ex = exp, pw = pow10, em = expm1: the readers \CALX_<kind>G, <kind>hG, <kind>gG, <kind>Ja); x = 0 -> 1; p = e + ceil(D/8) - 1 (top block position)

```
\gdef\calx_lexb_small #1;#2#3.%
{%
    \ifnum#3=\xint_c_ \xint_dothis\calx_lexb_zero\fi
    \xint_orthat\calx_lexb_regular #1;#2#3.%
}%
\gdef\calx_lexb_zero #1;#2#3.#4.#5\xint:{ F+1.0.100000001!}%
\gdef\calx_lexb_regular #1;#2#3.#4.#5\xint:%
```

```
{%
    \expandafter\calx_lexc
      \the\numexpr#4+(#3+\xint_c_iii)/\xint_c_viii-\xint_c_i\expandafter.%
      \the\numexpr#4\expandafter.%
      \csname calx_l#1\calx_lexhkind G\endcsname#2#5\xint:%
}%
```

p < -3: |x| < 10^-24, result 1; p >= 2: |x| >= 10^16, over/underflow; #3 the layout reader (\calx_lexG, or \calx_lexhG for P <= 8)

```
\gdef\calx_lexc #1.#2.%
{%
    \ifnum#1>\xint_c_i \xint_dothis\calx_lexc_over\fi
    \ifnum#1<-\xint_c_iii \xint_dothis\calx_lexc_small\fi
    \xint_orthat\calx_lexc_regular #1.#2.%
}%
\gdef\calx_lexc_over #1.#2.#3#4#5\xint:{\calx_lexO#4}%
\gdef\calx_lexc_small #1.#2.#3#4#5\xint:{ F+1.0.100000001!}%
\gdef\calx_lexc_regular #1.#2.#3#4#5\xint:%
{%
    \csname calx_lexL\the\numexpr#2+\xint_c_iii\endcsname
      #3#5100000000!100000000!100000000!100000000!\xint:#4%
}%
\endgroup
```

grid layout (e = -7..1, D <= P+21): drop -(e+3) low blocks or prepend e+3 zero blocks, then \calx_lexG reads f3 (10^-24), f2, f1, I (integer block) and the sign

```
\expandafter\def\csname calx_lexL-4\endcsname#1#2!#3!#4!#5!{#1}%
\expandafter\def\csname calx_lexL-3\endcsname#1#2!#3!#4!{#1}%
\expandafter\def\csname calx_lexL-2\endcsname#1#2!#3!{#1}%
\expandafter\def\csname calx_lexL-1\endcsname#1#2!{#1}%
\expandafter\let\csname calx_lexL0\endcsname\xint_firstofone
\expandafter\def\csname calx_lexL4\endcsname
   #1{#1100000000!100000000!100000000!100000000!}%
\expandafter\def\csname calx_lexL1\endcsname#1{#1100000000!}%
\expandafter\def\csname calx_lexL2\endcsname#1{#1100000000!100000000!}%
\expandafter\def\csname calx_lexL3\endcsname#1{#1100000000!100000000!100000000!}%
```

Z = nint(X5*434294482/10^9) ~ 10^5 |x|/ln10, X5 = nint(10^5 |x|); data for the table macro: s x12, s x34, s x56 (8-digit pairs), s xI. Rare path if the integer block xI >= 10^4 or the block at 10^8 (first of #5) is not zero.

```
\def\calx_lexGt #1!#2\xint:{#1}%
```

normal branch (shared): f3.f2.f1.xI.s<table letter><marker>; the marker (\calx_lexMk exp's writer, \calx_lemMk expm1's finisher, empty for the P <= 8 path) ends the stream and is read by the writers after k.

```
\def\calx_lexGn #1.#2.#3.#4.#5#6#7%
{%
    \expandafter\calx_lexK\the\numexpr(#4*100000+#3/1000)*434294482/1000000000.%
    #5#6#5#3.#5#2.#5#1.#5#4.#7%
}%
\def\calx_lexG 1#1!1#2!1#3!1#4!#5\xint:#6%
{%
    \ifnum#4>9999 \xint_dothis{\calx_lexHa#5\xint:#6#4.#3.#2.#1.}\fi
    \ifnum\calx_lexGt#5\xint:>100000000
      \xint_dothis{\calx_lexHa#5\xint:#6#4.#3.#2.#1.}\fi
```

```
    \xint_orthat{\calx_lexGn#1.#2.#3.#4.#6A\calx_lexMk}%
}%
```

P <= 8: as \calx_lexG with the table macros \calx_lexC<i>

```
\def\calx_lexhG 1#1!1#2!1#3!1#4!#5\xint:#6%
{%
    \ifnum#4>9999 \xint_dothis{\calx_lexHa#5\xint:#6#4.#3.#2.#1.}\fi
    \ifnum\calx_lexGt#5\xint:>100000000
      \xint_dothis{\calx_lexHa#5\xint:#6#4.#3.#2.#1.}\fi
    \xint_orthat{\calx_lexGn#1.#2.#3.#4.#6C\calx_lexMk}%
}%
```

k = ceil(s Z/10^5) (floor(a/b) = (2a-b+1)/(2b), a >= 0); #3 table letter (A: P <= 16, B: P > 16)

```
\def\calx_lexK #1.#2#3%
{%
    \expandafter\calx_lexKa\the\numexpr\if-#2 0-(2*#1-99999)/200000\else
      (2*#1+99999)/200000\fi.#1.#2#3%
}%
```

i = floor(m/1000), m = k 10^5 - s Z in [0, 10^5): table macro \calx_lex<A|B><i>

```
\def\calx_lexKa #1.#2.#3#4%
{%
    \csname calx_lex#4\the\numexpr(2*(#1*100000\if-#3+\else-\fi#2)-999)/2000%
    \endcsname#1.%
}%
```

the ln10 chain (template of log P9..P16): A!P2!P3!0;-k.sxI.<tail>

```
\calx_lLTdef\calx_lexC\calx_lexW{#1!#2!#3!#4;#5.#6.#7\calx_lexQ}{#6+}{#7}%
```

state R.S2.S4.S6.<J list>| (R1 and the Lb items from the table macro); stages: \calx_lexTmp <S><R><next>{<short table>}{<j of R = #1>}{<R update>} S: fetch F1!c2!c3!j. (load-time short tables); R: R, sums, J list

```
\def\calx_lexTmp #1#2#3#4#5#6%
{%
    \def#1##1.{\expandafter\expandafter\expandafter#2%
       \csname#4\the\numexpr#5\endcsname##1.}%
    \def#2##1!##2!##3!##4.##5.##6.##7.##8.##9|%
    {%
        \expandafter#3#6\the\numexpr##6+##1\expandafter.\the\numexpr##7+##2%
        \expandafter.\the\numexpr##8+##3.##4;##9|%
    }%
}%
\calx_lexTmp\calx_lexSa\calx_lexRa\calx_lexSb{calx_lexu}%
   {(#1+#1*#1/1000000000/20)/100000000}%
   {\the\numexpr#5+100*#1+#2/1000000\expandafter.}%
\calx_lexTmp\calx_lexSb\calx_lexRb\calx_lexSc{calx_lexv}%
   {(#1+#1*#1/1000000000/20)/1000000}%
   {\the\numexpr#5+100*#1+#2/1000000\expandafter.}%
\calx_lexTmp\calx_lexSc\calx_lexRc\calx_lexSd{calx_lexw}%
   {#1/10000}%
   {\the\numexpr#5+100*#1+#2/1000000\expandafter.}%
\calx_lexTmp\calx_lexSd\calx_lexRd\calx_lexV{calx_lexz}%
   {#1/100}{}%
\let\calx_lexTmp\relax
```

chain 2: t = s x - k ln10 + Lb + Ls + Lt + Lu + Lv

```
\def\calx_lexV #1.#2.#3.#4|#5.#6.#7.#8.#9.%
   {\calx_lexC#5+#1!#6+#2!#7+#3!0;-#8.#9.#4|\calx_lexQ}%
```

t = T0!t1!..!t6! (|t| < 5.5e-9: t1 = t2 = 0000 or 9999), s = floor(10^16 t) = T0 10^8 + t3t4, rho = t5t6: U = 10^24 (e^t - 1) = s 10^8 + rho + s^2/2e8 (dropped s rho/1e16 + t^3/6 < 0.6 units 10^-24), ONE chain, floor limbs u3 (signed)!u2!u1!u0!; then G (\calx_lexEc) and \calx_lexPS

```
\def\calx_lexW #1!#2!#3!#4!#5!#6!#7!%
{%
    \expandafter\calx_lexWj\expandafter{\the\numexpr
    0\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+#1*100000000+#4#5\expandafter\calx_lLR
    \the\numexpr\calx_lLoff\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+#6#7+(#1*100000000+#4#5)*(#1*100000000+#4#5)%
      /200000000\relax!}%
}%
\def\calx_lexWj #1#2;#3;#4;#5;#6;|{\calx_lexEc\calx_lexPS{#1}#6.#5.#4.#3.#2.}%
```

Y' = G + G U 10^-24 (G = g5..g0 ~ 10^24, U = u3..u0): columns c = i + m >= 6, guard c = 5 (its u0 term by \calx_lexPT), dropped columns and guard floor < 6 units of Y'; output T!y1!..!y5! (stream k.<marker>)

```
\def\calx_lexPS #1!#2!#3!#4!#5!#6!#7!#8!#9!%
{%
    \expandafter\calx_lexY\the\numexpr#1\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+#2\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+#3\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+#4+#1*#7\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+#5+#1*#8+#2*#7\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+#6+#1*#9+#2*#8+#3*#7\expandafter\calx_lLRz
    \the\numexpr\calx_lLoff+#2*#9+#3*#8+#4*#7\calx_lexPT#1!%
}%
\def\calx_lexPT #1!#2!{+#1*#2\relax!}%
```

writer: exp x = 10^(k-24) Y', Y' = T!y1!..!y5!, then k.<marker>. expm1 marker: \calx_lemF; frequent: P9..P16 and 1000 <= T <= 9999 ((2T - 10999)^2 <= 8999^2); else generic (\CALX_ffromraw, \CALX_fround).

```
\def\calx_lexY_low #1!#2!#3!#4!#5!#6!#7.#8%
{%
    \ifx#8\calx_lexMk\else\xint_dothis{\calx_lemF#1#2#3#4#5#6.#7.24.#8}\fi
    \xint_orthat{\calx_lexYg#1#2#3#4#5#6.#7.}%
}%
\def\calx_lexY_mid #1!#2!#3!#4!#5!#6!#7.#8%
{%
    \ifx#8\calx_lexMk\else\xint_dothis{\calx_lemF#1#2#3#4#5#6.#7.24.#8}\fi
    \ifnum\numexpr(2*#1-10999)*(2*#1-10999)>80982001
      \xint_dothis{\calx_lexYg#1#2#3#4#5#6.#7.}\fi
    \xint_orthat{\expandafter\calx_lexYa\calx_lexq#1#2.#3#4.#5.#7.}%
}%
\def\calx_lexYg #1.#2.{\expandafter\calx_lexYgg\the\numexpr#2-24.#1.}%
\def\calx_lexYgg #1.#2.%
   {\CALX_fnround{\calx_ldigits}{\CALX_ffromraw{#2[#1]}}}%
```

Q = 10^(16-P), C = 10^(P-8), Z = 1 if P = 16 (\calx_lexq); H = T y1, L = y2 y3: M = round(Y'/10^(24-P)) = Mh 10^8 + Ml, Mh = floor(H/Q), Ml = (H - Q Mh) C + nint(L/Q) (+ nint(y4/10^4) if P = 16; for P < 16 the rounding boundaries are integers: the lower limbs are irrelevant)

```
\def\calx_lexYa #1.#2.#3.#4.#5.#6.%
```

```
{%
    \expandafter\calx_lexYb\the\numexpr(2*#4-#1+1)/(2*#1)\expandafter.%
    \the\numexpr(#4-(2*#4-#1+1)/(2*#1)*#1)*#2+#5/#1+(2*#6+1)/20000*#3.%
}%
```

Mh.Ml.k. : carries (Ml = 10^8; M = 10^P -> 10^(P-1), k+1)

```
\def\calx_lexYb #1.#2.#3.%
{%
    \ifnum#2>99999999
    \xint_dothis{\expandafter\calx_lexYe\the\numexpr#1+1.#3.}\fi
    \xint_orthat{\calx_lexYh#1.#2.#3.}%
}%
\def\calx_lexYe #1.#2.%
{%
    \ifnum#1=\CALX_pten{\calx_ldigits-8} \xint_dothis{\expandafter\calx_lexYh
      \the\numexpr#1/10\expandafter.\expandafter0\expandafter.%
      \the\numexpr#2+1.}\fi
    \xint_orthat{\calx_lexYh#1.0.#2.}%
}%
```

g = k - P, e = floor(g/8), h = g - 8e (\calx_lexh<h> = h.10^h.10^(8-h).), D = P + h; blocks of M 10^h: A = floor(Ml/V), B = floor(Mh/V), V = 10^(8-h)

```
\def\calx_lexYh #1.#2.#3.%
   {\expandafter\calx_lexYi\the\numexpr(2*(#3-\calx_ldigits)-7)/16.#3.#1.#2.}%
\def\calx_lexYi #1.#2.%
{%
    \expandafter\expandafter\expandafter\calx_lexYk
    \csname calx_lexh\the\numexpr#2-\calx_ldigits-8*#1\endcsname#1.%
}%
\begingroup\catcode`F=12
\gdef\calx_lexYk #1.#2.#3.#4.#5.#6.%
{%
    \expanded{ F+\the\numexpr\calx_ldigits+#1.#4.%
      \the\numexpr100000000+(#6-(2*#6-#3+1)/(2*#3)*#3)*#2!%
      \the\numexpr100000000+(2*#6-#3+1)/(2*#3)+(#5-(2*#5-#3+1)/(2*#3)*#3)*#2!%
      \ifnum#5<#3 \else\the\numexpr100000000+(2*#5-#3+1)/(2*#3)!\fi}%
}%
\endgroup
\def\calx_lLtmp #1.%
{%
    \expandafter\edef\csname calx_lexh#1\endcsname{#1.\CALX_pten{#1}.%
      \CALX_pten{8-#1}.}%
    \ifnum#1<7 \expandafter\calx_lLtmp\the\numexpr#1+1\expandafter.\fi
}%
\calx_lLtmp 0.\let\calx_lLtmp\relax
```

33.12. Exponential, P1..P8

Front, k, i and R1 as P16 (table macros \calx_lexC<i>); stages k = 2, 4 on the short residual R (scale 10^-10, keys only, P > 16 receivers); t = R4 10^-10, U = 10^10 (e^t - 1) = R4 + R4^2/(2 10^10) (t^3/6 < 2.2e-14); A = c(100+j2) = a1 10^4 + a0, G = A (10^4+j4) = H 10^4 + g0, H = A + a1 j4, g0 = a0 j4; Z' = 10^12 e^r0 = G (10^10+U)/10^8 = H 10^6 + L', L' = 100 g0 + H U/10^4 + g0 U/10^8; normalized H', L'' (0 <= L'' < 10^6); M = round(Z'/10^(12-P)), exp x = M 10^(k-P): \calx_ldgoc. P >= 6: M = H' 10^(P-6) + round(L''/10^(12-P)); P <= 5: M = round(H'/10^(6-P)) exactly (the rounding

boundaries are multiples of 10^6: L'' irrelevant). Relative error < 2.2e-10 (R1 0.52, tables 2 x 0.5, U 0.5 units 10^-10): 0.022 ulp_8. R4.j4;j2;c;|k.

```
\def\calx_lexhV #1.#2;#3;#4;|%
{%
    \expandafter\calx_lexhW\the\numexpr#1+#1*#1/1000000000/20\expandafter.%
    \the\numexpr(2*#4*(100+#3)-9999)/20000\expandafter.%
    \the\numexpr#4*(100+#3).#2.%
}%
```

U, a1, A, j4: H = A + a1 j4, g0 = (A - 10^4 a1) j4

```
\def\calx_lexhW #1.#2.#3.#4.%
{%
    \expandafter\calx_lexhX\the\numexpr#3+#2*#4\expandafter.%
    \the\numexpr(#3-10000*#2)*#4.#1.%
}%
```

H, g0, U: L' -> H', L''

```
\def\calx_lexhX #1.#2.#3.%
{%
    \expandafter\calx_lexhY\the\numexpr100*#2+#1*#3/10000+#2*#3/100000000.#1.%
}%
\def\calx_lexhY #1.#2.%
{%
    \expandafter\calx_lexhZ\the\numexpr#2+(2*#1-999999)/2000000\expandafter.%
    \the\numexpr#1-1000000*((2*#1-999999)/2000000).%
}%
```

H', L'', k: frequent 10^5 <= H' < 10^6; M by P >= 6 / P <= 5

```
\def\calx_lexhZ #1.#2.#3.#4%
{%
    \ifx#4\calx_lexMk\else\xint_dothis{#4}\fi
    \ifnum#1<100000 \xint_dothis\calx_lexhR\fi
    \ifnum#1>999999 \xint_dothis\calx_lexhR\fi
    \xint_orthat\calx_lexhround#1.#2.#3.%
}%
\def\calx_lexhP #1.#2.#3.%
{%
    \expandafter\calx_lexhO\the\numexpr#1*\CALX_pten{\calx_ldigits-6}%
      +#2/\CALX_pten{12-\calx_ldigits}\expandafter.\the\numexpr#3-\calx_ldigits.%
}%
\def\calx_lexhQ #1.#2.#3.%
{%
    \expandafter\calx_lexhO\the\numexpr#1/\CALX_pten{6-\calx_ldigits}%
      \expandafter.\the\numexpr#3-\calx_ldigits.%
}%
\def\calx_lexhO #1.#2.{\expandafter\calx_ldgoc\the\numexpr#1\expandafter.%
   \the\numexpr(2*#2-7)/16.#2.+}%
\def\calx_lexhR #1.#2.#3.%
{%
    \expandafter\calx_lexhRr\expanded{\the\numexpr#3-12.%
      #1\expandafter\xint_gobble_i\the\numexpr1000000+#2.}%
}%
\def\calx_lexhRr #1.#2.{\CALX_fnround{\calx_ldigits}{\CALX_ffromraw{#2[#1]}}}%
```

rare path 10^4 <= |x| < 10^16: D = |x| - |K| ln10 with |K| = 8 K8, K8 = floor(Ze/8) - 25, Ze ~ |x|/ln10 (8 leading digits): D in (290, 650) exactly on the grid (ln10 floored at 10^-36, |K| < 2^31); the fast path on x' = s D, then e <- e + s K8 (K = s |K|). |x| >= 4944763850 (approx.): overflow (x > 0), underflow (x < 0).

```
\def\calx_lexHa 1#1!#2\xint:#3#4.%
{%
    \ifnum#1>49 \xint_dothis{\calx_lexOd#3}\fi
    \ifnum\numexpr#1*1000000+#4/100>49447638 \xint_dothis{\calx_lexOd#3}\fi
    \xint_orthat{\expandafter\calx_lexHb\the\numexpr
      (\ifnum#1>\xint_c_ (#1*1000000+#4/100)*434294482/1000000000*100\else
       #4*434294482/1000000000\fi-4)/8-25.#3#1.#4.}%
}%
\def\calx_lexOd #1#2.#3.#4.{\calx_lexO#1}%
```

K8, s, I1, I0, f1, f2, f3; |K| = k2 10^8 + k1 10^4 + k0; tail s K8.

```
\def\calx_lexHb #1.#2#3.#4.%
   {\expandafter\calx_lexHc\the\numexpr(2*#1-12499999)/25000000.#1.#2#3.#4.}%
\def\calx_lexHc #1.#2.#3#4.#5.#6.#7.#8.%
   {\expandafter\calx_lexHd\the\numexpr8*#2-100000000*#1.#1.#4.#5.#6.#7.#8.#3#2.}%
\def\calx_lexHd #1.#2.#3.#4.#5.#6.#7.%
{%
    \expandafter\calx_lexHe\the\numexpr(2*#1-9999)/20000\expandafter.%
    \the\numexpr#1-10000*((2*#1-9999)/20000).#2.#3.#4.#5.#6.#7.%
}%
```

k1, k0, k2, I1, I0, f1, f2, f3: D = |x| - |K| ln10, ln10 limbs 2|3025|8509| 2994|0456|8401|7991|4546|8436|4207 (floor 10^-36); pairs I0 (w=0), f1 (w=-2), f2 (w=-4), f3 (w=-6), I1 (10^8); columns w >= 0 merged (they cancel to D's integer part); w=-7 (+ w=-8) carry only

```
\def\calx_lexHe #1.#2.#3.#4.#5.#6.#7.#8.%
{%
    \expandafter\calx_lexHf\the\numexpr((#4-2*#3)*10000-2*#1-3025*#3)*10000%
      +#5-2*#2-3025*#1-8509*#3\expandafter\calx_lLRh
    \the\numexpr\calx_lLoffh-3025*#2-8509*#1-2994*#3\expandafter\calx_lLRh
    \the\numexpr\calx_lLoffh+#6-8509*#2-2994*#1-456*#3\expandafter\calx_lLRh
    \the\numexpr\calx_lLoffh-2994*#2-456*#1-8401*#3\expandafter\calx_lLRh
    \the\numexpr\calx_lLoffh+#7-456*#2-8401*#1-7991*#3\expandafter\calx_lLRh
    \the\numexpr\calx_lLoffh-8401*#2-7991*#1-4546*#3\expandafter\calx_lLRh
    \the\numexpr\calx_lLoffh+#8-7991*#2-4546*#1-8436*#3\expandafter\calx_lLRhz
    \the\numexpr\calx_lLoffh-4546*#2-8436*#1-4207*#3%
      +(-8436*#2-4207*#1)/10000\relax!%
}%
```

Dint, m1..m6, s, K8: the fast path on x' = s D, then the exponent (P <= 8 reader \calx_lexhG; also used by pow10)

```
\def\calx_lexHf #1!#2!#3!#4!#5!#6!#7!#8#9.%
{%
    \expandafter\calx_lexHz\romannumeral0%
    \csname calx_lex\calx_lexhkind G\expandafter\endcsname\expanded
      {1#6#7!1#4#5!1#2#3!\the\numexpr100000000+#1!100000000!}\xint:#8%
    \xint:#8#9.%
}%
```

E = 8e + D - 1: Overflow if E > 2^31 - 9 (the xint printer fails at E = 2^31 - 1), Underflow if E < -(2^31-1) + P + 7

```
\begingroup\catcode`F=12
```

```
\gdef\calx_lexHz F#1#2.#3.#4\xint:#5.%
{%
    \expandafter\calx_lexHy\the\numexpr#3#5.#2.#4\xint:%
}%
\gdef\calx_lexHy #1.#2.#3\xint:%
{%
    \ifnum\ifnum#1>268435440
      \numexpr8*(#1-268435455)+#2\relax\else0\fi>\xint_c_
      \xint_dothis{\calx_lexO+}\fi
    \ifnum\ifnum#1<-268435440 \numexpr8*(#1+268435455)+#2-\calx_ldigits\relax
      \else1\fi<\xint_c_i \xint_dothis{\calx_lexO-}\fi
    \xint_orthat{ F+#2.#1.#3}%
}%
\gdef\calx_lexO #1%
{%
    \if-#1\expandafter\xint_firstoftwo\else\expandafter\xint_secondoftwo\fi
    {\expandafter\CALX_sp\romannumeral`&&@%
       \CALX_signalcondition{Underflow}{exp(x) underflow}{F+0.0.}{F+0.0.}}%
    {\CALX_nsig{Overflow}{exp(x)}{+}}%
}%
\endgroup
```

33.13. Exponential, P17..P100

Exponential, P17..P100. Grid W = 8m, m = ceil(P/8) + 1 pairs, n = 2m limbs. x: integer block and m pairs MS first (signed); k, i, R1 as P16 (table macros \calx_lexB<i>); stages k = 2, 4, 6, 8 on R (keys only); t = x + Lv + Lu + Lt + Ls + Lb - k ln10 by log's zip adder \calx_lLGTx and ln10 chain template \calx_lLGcidef; e^t: P17..P32 an explicit series and product by fixed chains (below); P > 32: e^t = 1 + t E1(t), E1 = sum t^k/(k+1)!, by the kernel (schedules \calx_lexS: all levels negate their output, \calx_tZun / \calx_tZvn); Y = G e^t, G = c (100+j2)(10^4+j4)(10^6+j6)(10^8+j8) exact (6 limbs), one kernel level (\calx_lexT); exp x = 10^(k-W) Y, \CALX_ffromraw + fround.

```
\def\calx_lexgm{\numexpr(\calx_ldigits+3)/8+1\relax}% m (internal integer)
```

s, D, e, blocks: p = top block position; p >= 2 over/underflow; p < -m: |x| < 10^-W, result 1; p = 1 or integer block >= 10^4: rare path

```
\begingroup\catcode`F=12
\gdef\calx_lexgb #1;#2#3.%
{%
    \ifnum#3=\xint_c_ \xint_dothis\calx_lexgb_zero\fi
    \xint_orthat\calx_lexgb_regular #1;#2#3.%
}%
\gdef\calx_lexgb_zero #1;#2#3.#4.#5\xint:{ F+1.0.100000001!}%
\gdef\calx_lexgb_regular #1;#2#3.#4.#5\xint:%
{%
    \expandafter\calx_lexgc
      \the\numexpr#4+(#3+\xint_c_iii)/\xint_c_viii-\xint_c_i.#4.#1;#2#5\xint:%
}%
\gdef\calx_lexgc #1.%
{%
    \ifnum#1>\xint_c_i \xint_dothis\calx_lexgc_over\fi
    \ifnum#1<-\calx_lexgm \xint_dothis\calx_lexgc_small\fi
    \xint_orthat\calx_lexgc_regular #1.%
}%
```

```
\gdef\calx_lexgc_over #1.#2.#3;#4#5\xint:{\calx_lexO#4}%
\gdef\calx_lexgc_small #1.#2.#3;#4#5\xint:{ F+1.0.100000001!}%
\gdef\calx_lexgc_regular #1.#2.#3;#4#5\xint:%
{%
    \csname calx_l#3\ifnum#1>\xint_c_ Ja\else gG\fi
      \expandafter\endcsname\expanded{#4%
      \csname calx_lexgz\ifnum#1>\xint_c_ 0\else\the\numexpr-#1\fi\endcsname
      \expandafter\calx_lexgDp\the\numexpr-\calx_lexgm-#2.#4#5\xint:!%
      \csname calx_lexgz\ifnum#2>-\calx_lexgm\the\numexpr#2+\calx_lexgm\else0\fi
      \endcsname
      \noexpand\calx_lexQ\expandafter\noexpand\csname calx_l#3Mk\endcsname}%
}%
\endgroup
```

drop d low blocks, then reverse into signed items s<8 digits>! (MS first)

```
\def\calx_lexgDp #1.%
   {\ifnum#1>\xint_c_
    \expandafter\calx_lexgDq\else\expandafter\calx_lexgDr\fi#1.}%
\def\calx_lexgDq #1.#2#3!{\expandafter\calx_lexgDp\the\numexpr#1-1.#2}%
\def\calx_lexgDr #1.#2{\calx_lexgRw#2{}}%
\def\calx_lexgRw #1#2#3#4!%
   {\xint_gob_til_xint: #3\calx_lexgRe\xint:\calx_lexgRw#1{#1#4!#2}}%
\def\calx_lexgRe\xint:\calx_lexgRw#1#2{\calx_lexgRf#2}%
\def\calx_lexgRf #1!{}%
```

s, s xI, s f1, s f2 (rest s f3!..!s fm!\calx_lexQ): Z, then k, i as P16

```
\def\calx_lexgG #1#2!%
{%
    \ifnum#1#2>9999 \xint_dothis\calx_lexgG_large\fi
    \xint_orthat\calx_lexgG_regular #1#2!%
}%
\def\calx_lexgG_large #1#2!#3!#4!{\calx_lexJa#1+00000000!#2!#3!#4!}%
\def\calx_lexgG_regular #1#2!#3!#4!%
{%
    \expandafter\calx_lexK\the\numexpr
      (0#1(#2*100000+#3/1000))*434294482/1000000000.#1B#2!#3!#4!%
}%
```

stages (keys only): \calx_lexgTmp <S><R><next>{<short table>}{<j of R>}{<R part>}

```
\def\calx_lexgTmp #1#2#3#4#5#6%
{%
    \def#1##1.{\expandafter\expandafter\expandafter#2%
       \csname#4\the\numexpr#5\endcsname##1.}%
    \def#2##1!##2!##3!##4.##5.##6|{#6##4;##6|}%
}%
\calx_lexgTmp\calx_lexgSa\calx_lexgRa\calx_lexgSb{calx_lexu}%
   {(#1+#1*#1/1000000000/20)/100000000}%
   {\expandafter\calx_lexgSb\the\numexpr#5+100*#1+#2/1000000.}%
\calx_lexgTmp\calx_lexgSb\calx_lexgRb\calx_lexgSc{calx_lexv}%
   {(#1+#1*#1/1000000000/20)/1000000}%
   {\expandafter\calx_lexgSc\the\numexpr#5+100*#1+#2/1000000.}%
\calx_lexgTmp\calx_lexgSc\calx_lexgRc\calx_lexgSd{calx_lexw}%
   {#1/10000}%
   {\expandafter\calx_lexgSd\the\numexpr#5+100*#1+#2/1000000.}%
```

```
\calx_lexgTmp\calx_lexgSd\calx_lexgRd\calx_lexgV{calx_lexz}%
   {#1/100}{\calx_lexgV}%
\calx_lexgTmp\calx_lexhSa\calx_lexhRa\calx_lexhSb{calx_lexu}%
   {(#1+#1*#1/1000000000/20)/100000000}%
   {\expandafter\calx_lexhSb\the\numexpr#5+100*#1+#2/1000000.}%
\calx_lexgTmp\calx_lexhSb\calx_lexhRb\calx_lexhV{calx_lexv}%
   {(#1+#1*#1/1000000000/20)/1000000}%
   {\expandafter\calx_lexhV\the\numexpr#5+100*#1+#2/1000000.}%
\let\calx_lexgTmp\relax
```

j8;j6;j4;j2;a;c;|k.sxI! then s f1!..!s fm!\calx_lexQ: the adder stream; keys j8;100j6;j4;100j2;a;-k;sxI. then c.j2.j4.j6.j8.k.; P > 32: log's generic zip adder (\calx_lexgVb), P <= 32: log's fixed-arity adders (templates \calx_lLGTdef, \calx_lLGAdef, \calx_lLGBdef, chains \calx_lLGcadef, \calx_lLGcbdef), accs LS first: L24 f4!f3!f2!f1; L32 f4!f3!f2!f1!f5;

```
\def\calx_lexgV #1;#2;#3;#4;#5;#6;|#7.#8!%
{%
    \expandafter\calx_lexgVkernel\expanded{{#1;\the\numexpr100*#2\relax;#3;%
      \the\numexpr100*#4\relax;#5;-#7;#8.}{#6.#4.#3.#2.#1.#7.}}%
}%
\def\calx_lexgVb #1#2#3\calx_lexQ{\calx_lexTv#3;#1#2}%
\def\calx_lexgVp #1#2#3!#4!#5!#6!\calx_lexQ{\calx_lexNva#6!#5!#4!#3;#1#2}%
\def\calx_lexgVq #1#2#3!#4!#5!#6!#7!\calx_lexQ{\calx_lexNvb#6!#5!#4!#3!#7;#1#2}%
\calx_lLGTdef\calx_lexNva{calx_lLv}\calx_lexUva \calx_lLGAdef\calx_lexUva\calx_lexNua
\calx_lLGTdef\calx_lexNua{calx_lLu}\calx_lexUua \calx_lLGAdef\calx_lexUua\calx_lexNta
\calx_lLGTdef\calx_lexNta{calx_lLt}\calx_lexUta \calx_lLGAdef\calx_lexUta\calx_lexNsa
\calx_lLGTdef\calx_lexNsa{calx_lLs}\calx_lexUsa \calx_lLGAdef\calx_lexUsa\calx_lexNba
\calx_lLGTdef\calx_lexNba{calx_lLb}\calx_lexUba \calx_lLGAdef\calx_lexUba\calx_lexNca
\calx_lLGTdef\calx_lexNvb{calx_lLv}\calx_lexUvb \calx_lLGBdef\calx_lexUvb\calx_lexNub
\calx_lLGTdef\calx_lexNub{calx_lLu}\calx_lexUub \calx_lLGBdef\calx_lexUub\calx_lexNtb
\calx_lLGTdef\calx_lexNtb{calx_lLt}\calx_lexUtb \calx_lLGBdef\calx_lexUtb\calx_lexNsb
\calx_lLGTdef\calx_lexNsb{calx_lLs}\calx_lexUsb \calx_lLGBdef\calx_lexUsb\calx_lexNbb
\calx_lLGTdef\calx_lexNbb{calx_lLb}\calx_lexUbb \calx_lLGBdef\calx_lexUbb\calx_lexNcb
\calx_lLGcadef\calx_lexNca\calx_lexgW{#1!#2!#3!#4;#5;#6.}{#6+}{\calx_lLQ}%
\calx_lLGcbdef\calx_lexNcb\calx_lexgW{#1!#2!#3!#4!#5;#6;#7.}{#7+}%
\def\calx_lexTv{\calx_lLGTx\calx_lexTu{calx_lLv}}%
\def\calx_lexTu{\calx_lLGTx\calx_lexTt{calx_lLu}}%
\def\calx_lexTt{\calx_lLGTx\calx_lexTs{calx_lLt}}%
\def\calx_lexTs{\calx_lLGTx\calx_lexTb{calx_lLs}}%
\def\calx_lexTb{\calx_lLGTx\calx_lexch{calx_lLb}}%
\def\calx_lexch{\expandafter\calx_lexci\calx_lLzl\relax}%
\calx_lLGcidef\calx_lexci\calx_lexgW{#1\relax#2!#3;#4;#5.}{#5+}%
```

t = I!l1!..!ln!\calx_lLQ (|t| < 5.5e-9): P <= 32 the explicit back end (\calx_lexE<p|n><a|b>, sign of t, layout L = 24 or 32), P > 32 the kernel

```
\def\calx_lexgWsmall #1!%
{%
    \csname calx_lexE\ifnum#1<\xint_c_ n\else p\fi
      \calx_lexlayout\endcsname#1!%
}%
```

P > 32: U'' = 10^8 t (l3..ln, t < 0: top l3 - 10^4), kernel U list, schedule \calx_lexS

```
\def\calx_lexgWk #1!#2!#3!#4\calx_lLQ%
{%
    \ifnum#1<\xint_c_ \expandafter\calx_lexgWn\else\expandafter\calx_lexgWs\fi
```

```
    #4\calx_tZq!%
}%
\def\calx_lexgWn #1!{\expandafter\calx_lexgWs\the\numexpr#1-10000!}%
\def\calx_lexgWs #1\calx_tZq!%
{%
    \expandafter\expandafter\expandafter\calx_tZst
    \expandafter\calx_lexS\expandafter{\expanded{\calx_tZbt#1\calx_tZq!}}%
}%
```

```
{U}x_top.<n+1 limbs dddd!>\calx_tZq c.j2.j4.j6.j8.  (then k.):  a = c(100+j2) = a1 10^4 + a0; T
items -(e^t 10^W)
```

```
\def\calx_lexgF #1#2.#3\calx_tZq{\expandafter\calx_lexgFb\expanded{#2.#3}\calx_tZq}%
\def\calx_lexgFb #1.#2\calx_tZq#3.#4.#5.#6.#7.%
{%
    \expandafter\calx_lexgGa\expanded{\the\numexpr(2*#3*(100+#4)-9999)/20000.%
      \the\numexpr#3*(100+#4)-10000*((2*#3*(100+#4)-9999)/20000).#5.#6.#7.%
      \noexpand\calx_lexgGc{-#1.\calx_lexgTn#2\calx_tZq!}}%
}%
\def\calx_lexgTn #1!%
   {\calx_tZgob#1\calx_lexgTe\calx_tZq\noexpand\calx_tZn-#1.\calx_lexgTn}%
\def\calx_lexgTe\calx_tZq#1\calx_lexgTn{}%
```

```
a1.a0.j4.j6.j8.<cont>:  a b = p3!p2!p1!p0!, b = 100 10^8 + (100 j4 + j6) 10^4 + j4 j6 (shared by
P17..P100; <cont> = \calx_lexgGc (kernel path), \calx_lexPA, \calx_lexPB)
```

```
\def\calx_lexgGa #1.#2.#3.#4.#5.%
{%
    \expandafter\calx_lexgGb\the\numexpr100*#1\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+100*#2+#1*(100*#3+#4)\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+#2*(100*#3+#4)+#1*#3*#4\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+#2*#3*#4\relax!#5.%
}%
```

```
G = a b (10^8 + j8):  <cont>g5!..!g0! (6 limbs, top not normalized)
```

```
\def\calx_lexgGb #1!#2!#3!#4!#5.#6%
{%
    \expandafter#6\the\numexpr#1\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+#2\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+#3+#1*#5\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+#4+#2*#5\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+#3*#5\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+#4*#5\relax!%
}%
\def\calx_lexgGc #1!#2!#3!#4!#5!#6!#7%
{%
    \expandafter\calx_lexT\expandafter
    {\expanded{\calx_tZbt#1!#2!#3!#4!#5!#6!\calx_tZq!}}{#7}%
}%
```

```
{U}x_top.<n limbs dddd!>\calx_tZq k.  :  exp x = 10^(k-W) Y
```

```
\def\calx_lexgY #1#2.#3\calx_tZq
   {\expandafter\calx_lexgYb\expanded{#2.#3}\calx_tZq}%
\def\calx_lexgYb #1.#2\calx_tZq#3.%
{%
    \expandafter\calx_lexgYd\expanded{\ifnum#1>\xint_c_ #1\fi
      \calx_lexgYs#2\calx_tZq!}#3.%
```

```
}%
```

digits.k.<marker>: expm1 -> \calx_lemF, else \calx_lexgYa

```
\def\calx_lexgYd #1.#2.#3{\ifx#3\calx_lexMk\expandafter\xint_firstoftwo
   \else\expandafter\xint_secondoftwo\fi
   {\calx_lexgYe#1.#2.8*\calx_lexgm.}{\calx_lemF#1.#2.8*\calx_lexgm.#3}}%
\def\calx_lexgYe #1.#2.#3.{\expandafter\calx_lexgYa\the\numexpr#2-#3.#1.}%
\def\calx_lexgYs #1!{\calx_tZgob#1\calx_lexgYt\calx_tZq#1\calx_lexgYs}%
\def\calx_lexgYt\calx_tZq#1\calx_lexgYs{.}%
\def\calx_lexgYa #1.#2.%
   {\CALX_fnround{\calx_ldigits}{\CALX_ffromraw{#2[#1]}}}%
```

P17..P32: explicit back end, no kernel. t = I!0000!0000!l3!..!ln!\calx_lLQ (I = 0) or -1!9999!9999!..; s = floor(10^16 t) = u 10^4 + w (u = l3 or l3 - 10^4, w = l4); U = 10^W (e^t - 1) by ONE carry chain (floor limbs, signed top), braced; then G (\calx_lexgGa), Y' = G 10^(W-24) + G U 10^-24 (\calx_lexPA, \calx_lexPB), exp x = 10^(k-W) Y' (writer \calx_lexXa, \calx_lexXb). L = 24 (W = 32): u.s.w!l5!l6!l7!l8!\calx_lLQ (rho = l5l6, lambda = l7l8); U = s 1e16 + rho 1e8 + lambda + s^2/2 + s rho/1e8 + s^3/6e16, s^2/2 = (u^2/2) 1e8 + u w 1e4 + w^2/2

```
\def\calx_lexEpa 0!0000!0000!#1!#2!{\calx_lexEa#1.#1#2.#2!}%
\def\calx_lexEna -1!9999!9999!#1!#2!%
   {\expandafter\calx_lexEa
    \the\numexpr#1-10000\expandafter.\the\numexpr#1#2-100000000.#2!}%
\def\calx_lexEa #1.#2.#3!#4!#5!#6!#7!\calx_lLQ%
{%
    \expandafter\calx_lexEc\expandafter\calx_lexPA\expandafter{\the\numexpr
    0\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+#2\expandafter\calx_lLR
    \the\numexpr\calx_lLoff\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+#4#5+#1*#1/2\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+(#1*#1-2*(#1*#1/2))*5000+#1*#3\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+#6#7+#3*#3/2+#2*#4#5/100000000%
      +(#2*#2/100000000)*#2/600000000\relax!}%
}%
```

L = 32 (W = 40): u.w!l5!..!l10!\calx_lLQ -> h1.h0.Ls.u.w!r1!r0!lambda!mu! (s^2 = Hs 1e8 + Ls exactly, Hs = h1 1e4 + h0)

```
\def\calx_lexEpb 0!0000!0000!#1!#2!{\calx_lexEh#1.#2!}%
\def\calx_lexEnb -1!9999!9999!#1!#2!%
   {\expandafter\calx_lexEh\the\numexpr#1-10000.#2!}%
\def\calx_lexEh #1.#2!#3!#4!#5!#6!#7!#8!\calx_lLQ%
{%
    \expandafter\calx_lexEb
    \the\numexpr(#1*#1+2*#1*#2/10000)/10000\expandafter.%
    \the\numexpr#1*#1+2*#1*#2/10000-10000*((#1*#1+2*#1*#2/10000)/10000)%
    \expandafter.\the\numexpr(2*#1*#2-10000*(2*#1*#2/10000))*10000+#2*#2.%
    #1.#2!#3!#4!#5#6!#7#8!%
}%
```

U = s 1e24 + rho 1e16 + lambda 1e8 + mu + t^2/2 + t^3/6 + t^4/24 (units 10^-40): s rho = u r1 1e8 + (u r0 + w r1) 1e4 + w r0; s^3/6e8 = s Hs/6 + s Ls/6e8, s Hs/6 = (u h1/6) 1e8 + ((u h0 + w h1)/6) 1e4 + w h0/6

```
\def\calx_lexEb #1.#2.#3.#4.#5!#6!#7!#8!#9!%
{%
    \expandafter\calx_lexEc\expandafter\calx_lexPB\expandafter{\the\numexpr
    0\expandafter\calx_lLR
```

```
    \the\numexpr\calx_lLoff+#4*10000+#5\expandafter\calx_lLR
    \the\numexpr\calx_lLoff\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+#6#7+#4*#4/2\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+(#4*#4-2*(#4*#4/2))*5000+#4*#5\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+#8+#5*#5/2+#4*#6+#4*#1/6\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+(#5*#5-2*(#5*#5/2))*5000+#4*#7+#5*#6%
      +(#4*#2+#5*#1)/6\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+#9+#5*#7+#6#7*#6#7/200000000%
      +(#4*10000+#5)*#8/100000000+(#4*10000+#5)*#3/600000000%
      +(#4*#1-6*(#4*#1/6))*50000000/3%
      +(#4*#2+#5*#1-6*((#4*#2+#5*#1)/6))*5000/3+#5*#2/6%
      +(#1*10000+#2)*#6#7/200000000%
      +(#1*10000+#2)*(#1*10000+#2)/1200000000/2\relax!}%
}%
```

<cont>{U limbs}c.j2.j4.j6.j8. : a = c (100 + j2) = a1 10^4 + a0, then G

```
\def\calx_lexEc #1#2#3.#4.#5.#6.#7.%
{%
    \expandafter\calx_lexgGa\the\numexpr(2*#3*(100+#4)-9999)/20000\expandafter.%
    \the\numexpr#3*(100+#4)-10000*((2*#3*(100+#4)-9999)/20000).#5.#6.#7.#1#2%
}%
```

Y' = G 10^(W-24) + G U 10^-24: columns c = i + m (g_i u_m) >= W/4 - 2, c = W/4 - 3 guard (carry only); later partial sums passed as expression TEXT. L = 24: g5..g0, u5, u4, u3 (stream u2!u1!u0!k.)

```
\def\calx_lexPA #1!#2!#3!#4!#5!#6!#7!#8!#9!%
{%
    \expandafter\calx_lexXe\the\numexpr#1\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+#2\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+#3\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+#4+#1*#7\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+#5+#1*#8+#2*#7\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+#6+#1*#9+#2*#8+#3*#7%
    \calx_lexPAb#2*#9+#3*#8+#4*#7!#3*#9+#4*#8+#5*#7!#4*#9+#5*#8+#6*#7!%
    #1!#2!#3!%
}%
```

S1, S0, Sg (texts), g5, g4, g3, u2, u1, u0

```
\def\calx_lexPAb #1!#2!#3!#4!#5!#6!#7!#8!#9!%
{%
    \expandafter\calx_lLR\the\numexpr\calx_lLoff+#4*#7+#1%
    \expandafter\calx_lLR\the\numexpr\calx_lLoff+#4*#8+#5*#7+#2%
    \expandafter\calx_lLRz\the\numexpr\calx_lLoff+#4*#9+#5*#8+#6*#7+#3\relax!%
}%
```

L = 32: g5..g0, u7, u6, u5 (stream u4!..!u0!k.); column 9 left open

```
\def\calx_lexPB #1!#2!#3!#4!#5!#6!#7!#8!#9!%
{%
    \expandafter\calx_lexXf\the\numexpr#1\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+#2\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+#3\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+#4+#1*#7\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+#5+#1*#8+#2*#7\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+#6+#1*#9+#2*#8+#3*#7\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+#2*#9+#3*#8+#4*#7%
    \calx_lexPBb#3*#9+#4*#8+#5*#7!#4*#9+#5*#8+#6*#7\calx_lexPBc#6*#9!#5!%
```

```
    #5*#9+#6*#8\calx_lexPm#1!#2!#3!#4!%
}%
\let\calx_lexPm\relax
```

S8, bundle (S7, \calx_lexPBc, A5, g1, A6), g5..g2, u4, u3, u2: closes column 9; column 7 ends with S7 and \calx_lexPBc (A6 + B6 one parameter)

```
\def\calx_lexPBb #1!#2\calx_lexPm#3!#4!#5!#6!#7!#8!#9!%
{%
    +#3*#7\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+#1+#3*#8+#4*#7\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+#3*#9+#4*#8+#5*#7+#2+#4*#9+#5*#8+#6*#7!%
    #5*#9+#6*#8!#3!#4!#7!%
}%
```

A5, g1, A6 + B6, B5, g5, g4, u4, u1, u0

```
\def\calx_lexPBc #1!#2!#3!#4!#5!#6!#7!#8!#9!%
{%
    \expandafter\calx_lLR\the\numexpr\calx_lLoff+#5*#8+#3%
    \expandafter\calx_lLRz\the\numexpr\calx_lLoff+#5*#9+#6*#8+#2*#7+#4+#1\relax!%
}%
```

writer: T!q..!q0!k.<marker>: \calx_lexXe (L24), \calx_lexXf (L32) dispatch (expm1: \calx_lemXa, \calx_lemXb)

```
\def\calx_lexXe #1.#2{\ifx#2\calx_lexMk\calx_lexXj\fi\calx_lemXa#1.#2}%
\def\calx_lexXf #1.#2{\ifx#2\calx_lexMk\calx_lexXk\fi\calx_lemXb#1.#2}%
\def\calx_lexXj\fi\calx_lemXa#1.#2{\fi\calx_lexXa#1.}%
\def\calx_lexXk\fi\calx_lemXb#1.#2{\fi\calx_lexXb#1.}%
```

T!q..!q0!k. ; frequent 1000 <= T <= 9999 (Y' has W digits): pairs MS first P!..!P0!k. ; else \CALX_ffromraw (\calx_lexgYa)

```
\def\calx_lexXa #1!#2!#3!#4!#5!#6!#7!%
{%
    \ifnum\numexpr(2*#1-10999)*(2*#1-10999)>80982001
      \xint_dothis{\calx_lexXg32.#1#2#3#4#5#6#7}\fi
    \xint_orthat{\calx_lexXca#1#2!#3#4!#5#6!#7}%
}%
\def\calx_lexXb #1!#2!#3!#4!#5!#6!#7!#8!#9!%
{%
    \ifnum\numexpr(2*#1-10999)*(2*#1-10999)>80982001
      \xint_dothis{\calx_lexXg40.#1#2#3#4#5#6#7#8#9}\fi
    \xint_orthat{\calx_lexXcb#1#2!#3#4!#5#6!#7#8!#9}%
}%
\def\calx_lexXg #1.#2!#3.{\expandafter\calx_lexgYa\the\numexpr#3-#1.#2.}%
```

f = floor(k/8), h = k - 8f

```
\def\calx_lexXca #1!#2!#3!#4!#5.%
{%
    \expandafter\calx_lexXd\the\numexpr(2*#5-7)/16\expandafter.%
    \the\numexpr#5-8*((2*#5-7)/16).#1!#2!#3!#4!%
}%
\def\calx_lexXcb #1!#2!#3!#4!#5!#6.%
{%
    \expandafter\calx_lexXd\the\numexpr(2*#6-7)/16\expandafter.%
    \the\numexpr#6-8*((2*#6-7)/16).#1!#2!#3!#4!#5!%
}%
```

f.h. -> \calx_lexX<z|s|v|u><a|b> h.10^h.V.f. (V = 10^(8-h), \calx_lexh<h>): P = L (24, 32): v (h = 0), u (h > 0) direct; other P: z, s (\CALX_fround)

```
\def\calx_lexXd #1.#2.%
{%
    \csname calx_lexX\ifnum#2=\xint_c_ \calx_lexXzero\else\calx_lexXnonzero\fi
    \calx_lexlayout
    \expandafter\expandafter\expandafter\endcsname\csname calx_lexh#2\endcsname#1.%
}%
```

blocks of N = Y' 10^h (LS first), D = W + h, e = f - W/8; one rounding. floor(P/V) = (2P-V+1)/(2V)

```
\begingroup\catcode`F=12
\gdef\calx_lexXza #1.#2.#3.#4.#5!#6!#7!#8!%
   {\CALX_fnround\calx_ldigits
    {\expanded{F+32.\the\numexpr#4-4.1#8!1#7!1#6!1#5!}}}%
\gdef\calx_lexXzb #1.#2.#3.#4.#5!#6!#7!#8!#9!%
   {\CALX_fnround\calx_ldigits
    {\expanded{F+40.\the\numexpr#4-5.1#9!1#8!1#7!1#6!1#5!}}}%
\gdef\calx_lexXsa #1.#2.#3.#4.#5!#6!#7!#8!%
{%
    \CALX_fnround\calx_ldigits{\expanded{F+\the\numexpr32+#1.\the\numexpr#4-4.%
    \the\numexpr100000000+(#8-(2*#8-#3+1)/(2*#3)*#3)*#2!%
    \the\numexpr100000000+(#7-(2*#7-#3+1)/(2*#3)*#3)*#2+(2*#8-#3+1)/(2*#3)!%
    \the\numexpr100000000+(#6-(2*#6-#3+1)/(2*#3)*#3)*#2+(2*#7-#3+1)/(2*#3)!%
    \the\numexpr100000000+(#5-(2*#5-#3+1)/(2*#3)*#3)*#2+(2*#6-#3+1)/(2*#3)!%
    \the\numexpr100000000+(2*#5-#3+1)/(2*#3)!}}%
}%
\gdef\calx_lexXsb #1.#2.#3.#4.#5!#6!#7!#8!#9!%
{%
    \CALX_fnround\calx_ldigits{\expanded{F+\the\numexpr40+#1.\the\numexpr#4-5.%
    \the\numexpr100000000+(#9-(2*#9-#3+1)/(2*#3)*#3)*#2!%
    \the\numexpr100000000+(#8-(2*#8-#3+1)/(2*#3)*#3)*#2+(2*#9-#3+1)/(2*#3)!%
    \the\numexpr100000000+(#7-(2*#7-#3+1)/(2*#3)*#3)*#2+(2*#8-#3+1)/(2*#3)!%
    \the\numexpr100000000+(#6-(2*#6-#3+1)/(2*#3)*#3)*#2+(2*#7-#3+1)/(2*#3)!%
    \the\numexpr100000000+(#5-(2*#5-#3+1)/(2*#3)*#3)*#2+(2*#6-#3+1)/(2*#3)!%
    \the\numexpr100000000+(2*#5-#3+1)/(2*#3)!}}%
}%
```

P = L: M = (pairs above P0) + [P0 >= 5e7] (= round(Y'/10^8)), blocks of M 10^h, D = P + h, e = f - P/8; M1 + [P0 >= 5e7] = 10^8: the fround path

```
\gdef\calx_lexXva #1.#2.#3.#4.#5!#6!#7!#8!%
{%
    \ifnum\numexpr#7+#8/100000000>99999999
      \xint_dothis{\calx_lexXza#1.#2.#3.#4.#5!#6!#7!#8!}\fi
    \xint_orthat{\expanded{ F+24.\the\numexpr#4-3.%
      \the\numexpr1#7+#8/100000000!1#6!1#5!}}%
}%
\gdef\calx_lexXvb #1.#2.#3.#4.#5!#6!#7!#8!#9!%
{%
    \ifnum\numexpr#8+#9/100000000>99999999
      \xint_dothis{\calx_lexXzb#1.#2.#3.#4.#5!#6!#7!#8!#9!}\fi
    \xint_orthat{\expanded{ F+32.\the\numexpr#4-4.%
      \the\numexpr1#8+#9/100000000!1#7!1#6!1#5!}}%
}%
\gdef\calx_lexXua #1.#2.#3.#4.#5!#6!#7!#8!%
```

```
{%
    \ifnum\numexpr#7+#8/100000000>99999999
      \xint_dothis{\calx_lexXsa#1.#2.#3.#4.#5!#6!#7!#8!}\fi
    \xint_orthat
    {\expandafter\calx_lexXw\the\numexpr#7+#8/100000000.#1.#2.#3.#4.#5!#6!}%
}%
\gdef\calx_lexXw #1.#2.#3.#4.#5.#6!#7!%
{%
    \expanded{ F+\the\numexpr24+#2.\the\numexpr#5-3.%
    \the\numexpr100000000+(#1-(2*#1-#4+1)/(2*#4)*#4)*#3!%
    \the\numexpr100000000+(#7-(2*#7-#4+1)/(2*#4)*#4)*#3+(2*#1-#4+1)/(2*#4)!%
    \the\numexpr100000000+(#6-(2*#6-#4+1)/(2*#4)*#4)*#3+(2*#7-#4+1)/(2*#4)!%
    \the\numexpr100000000+(2*#6-#4+1)/(2*#4)!}%
}%
\gdef\calx_lexXub #1.#2.#3.#4.#5!#6!#7!#8!#9!%
{%
    \ifnum\numexpr#8+#9/100000000>99999999
      \xint_dothis{\calx_lexXsb#1.#2.#3.#4.#5!#6!#7!#8!#9!}\fi
    \xint_orthat{\expandafter\calx_lexXx\the\numexpr#8+#9/100000000.%
      #1.#2.#3.#4.#5!#6!#7!}%
}%
\gdef\calx_lexXx #1.#2.#3.#4.#5.#6!#7!#8!%
{%
    \expanded{ F+\the\numexpr32+#2.\the\numexpr#5-4.%
    \the\numexpr100000000+(#1-(2*#1-#4+1)/(2*#4)*#4)*#3!%
    \the\numexpr100000000+(#8-(2*#8-#4+1)/(2*#4)*#4)*#3+(2*#1-#4+1)/(2*#4)!%
    \the\numexpr100000000+(#7-(2*#7-#4+1)/(2*#4)*#4)*#3+(2*#8-#4+1)/(2*#4)!%
    \the\numexpr100000000+(#6-(2*#6-#4+1)/(2*#4)*#4)*#3+(2*#7-#4+1)/(2*#4)!%
    \the\numexpr100000000+(2*#6-#4+1)/(2*#4)!}%
}%
\endgroup
```

rare path P > 16, 10^4 <= |x| < 10^16 (as P16: K = s 8 K8, D = |x| - |K| ln10 in (290, 650), the fast path on s D, e <- e + s K8): s, s I1 (block 10^8), s I0, then s f1!..!s fm!\calx_lexQ. D by one chain: columns w = 2..-(n+1), X_w - k0 l_(-w) - k1 l_(1-w) - k2 l_(2-w) (ln10 limbs l_j, \calx_lexLx: j = 3..31, floor 10^-124); the last column only for the carry.

```
\def\calx_lexJa #1#2!#3!%
{%
    \ifnum#1#2>49 \xint_dothis\calx_lexJa_over\fi
    \ifnum\numexpr(#1#2)*1000000+(#1#3)/100>49447638
      \xint_dothis\calx_lexJa_over\fi
    \xint_orthat\calx_lexJa_regular #1#2!#3!%
}%
\def\calx_lexJa_over #1#2!#3!#4\calx_lexQ#5{\calx_lexO#1}%
\def\calx_lexJa_regular #1#2!#3!#4\calx_lexQ#5%
{%
    \expandafter\calx_lexJb\the\numexpr
      (\ifnum#1#2>\xint_c_
       ((#1#2)*1000000+(#1#3)/100)*434294482/1000000000*100%
       \else(#1#3)*434294482/1000000000\fi-4)/8-25.#1#2!#3!#4\calx_lexQ#5%
}%
```

K8, s, s I1, s I0, pairs: k2 = floor(8 K8/10^8), rem, k1, k0; X limbs

```
\def\calx_lexJb #1.#2#3!#4!#5\calx_lexQ%
```

```
{%
    \expandafter\calx_lexJc\expanded{\the\numexpr(2*#1-12499999)/25000000.%
      \the\numexpr8*#1-100000000*((2*#1-12499999)/25000000).%
      \the\numexpr#2#3!\calx_lexJs#4!\calx_lexJl#5\calx_lexQ!}0!\calx_lLQ!#2#1.%
}%
\def\calx_lexJs #1#2#3#4#5#6#7#8#9!{#2#3#4#5!#6#7#8#9!}%
\def\calx_lexJl #1#2!%
   {\ifx\calx_lexQ#1\else\calx_lexJs#1#2!\expandafter\calx_lexJl\fi}%
\def\calx_lexJc #1.#2.%
{%
    \expandafter\calx_lexJd\the\numexpr(2*#2-9999)/20000\expandafter.%
    \the\numexpr#2-10000*((2*#2-9999)/20000).#1.%
}%
```

k1, k0, k2, X2, X1, X0, rest X (\calx_lLQ!): the chain

```
\def\calx_lexJd{\expandafter\calx_lexJe\calx_lexLx\relax}%
\def\calx_lexJe #1\relax#2.#3.#4.#5!#6!#7!#8\calx_lLQ!%
{%
    \expandafter\calx_lexJf\the\numexpr#5-2*#4\expandafter\calx_lLRh
    \the\numexpr\calx_lLoffh+#6-2*#2-3025*#4\expandafter\calx_lLRh
    \the\numexpr\calx_lLoffh+#7-2*#3-3025*#2-8509*#4%
    \calx_lexJw#8\calx_lLQ!;3025.8509.#3.#2.#4.#10!0!\relax
}%
```

X_w, rest X, l_(-w), l_(1-w), k0, k1, k2, l_(2-w) (stream)

```
\def\calx_lexJw #1!#2;#3.#4.#5.#6.#7.#8!%
{%
    \calx_lLGgt#1\calx_lexJz\calx_lLQ\expandafter\calx_lLRh
    \the\numexpr\calx_lLoffh+#1-#5*#3-#6*#4-#7*#8\calx_lexJw#2;#4.#8.#5.#6.#7.%
}%
\def\calx_lexJz\calx_lLQ#1\relax{\relax!\calx_lLQ}%
```

c2!l1!l0!m1!..m_(n+1)!\calx_lLQ s K8.<marker> : the fast path on s D (the marker passed to it), then e + s K8

```
\def\calx_lexJf #1!#2!#3!#4\calx_lLQ#5#6.#7%
{%
    \expandafter\calx_lexHz\romannumeral0\expandafter\calx_lexgG\expanded
      {#5#5#2#3!\calx_lexJp#5#4\calx_lLQ!}\calx_lexQ#7\xint:#5#6.%
}%
\def\calx_lexJp #1#2!#3!%
   {\ifx\calx_lLQ#3\else#1#2#3!\expandafter\calx_lexJp\expandafter#1\fi}%
```

Short tables constructed once: F1!c2!c3!j. keyed by j: \calx_lexu<j> = Ls<100j>, \calx_lexv<j> = Lt<j>, \calx_lexw<j> = Lu<100j>

```
\def\calx_lexMs #1!#2!#3!#4\calx_lLQ#5#6{\expandafter\def\csname#5#6\endcsname
   {#1!#2!#3!#6.}}%
\def\calx_lexMt #1.#2.#3.#4.#5.%
{%
    \edef\calx_lLtmpb{\the\numexpr#3*#4}%
    \expandafter\expandafter\expandafter
    \calx_lexMs\csname#2\calx_lLtmpb\endcsname
      \calx_lLQ{#1}{#4}%
    \ifnum#4>#5 \xint_afterfi{\expandafter\calx_lexMt\expanded{#1.#2.#3.%
      \the\numexpr#4-1.#5.}}\fi
}%
```

table macros \calx_lexA<i> (i = 0..99) from \calx_lexAlist (calxlogdata.tex) and the entries \calx_lLb<a>: #1 k, #2 s x12, #3 s x34, #4 s x56, #5 s xI. R1 = 100 A2 + (-2994k + (s x34 + c2 - 456k)/10^4)/100, A2 = A1 10^4 + s x12 + F1 - 8509k, A1 = (s xI - 2k) 10^4 - 3025k (|A1| < 3.7e4, |A2| < 5e6). list entries i.key.c. (key of \calx_lLb<key>: 0 = zero entry, 1 = \calx_lLz (ln10), else a; c the bootstrap multiplier)

```
\expandafter\let\csname calx_lLb1\endcsname\calx_lLz
\def\calx_lexMaA #1!#2!#3!#4\calx_lLQ#5.#6.#7.%
{%
    \XINT_global\expandafter\edef\csname calx_lexA#5\endcsname##1.##2.##3.##4.##5.%
    {%
        \noexpand\expandafter\noexpand\calx_lexSa\noexpand\the\noexpand\numexpr
        (((##5-2*##1)*10000-3025*##1)*10000+##2+#1-8509*##1)*100%
        +(-2994*##1+(##3+#2-456*##1)/10000)/100.#1.#2.#3.%
        #7;|##2.##3.##4.##1.##5.##1.%
    }%
}%
\def\calx_lexLd #1.#2.#3.%
{%
    \ifnum#1<\xint_c_ \expandafter\xint_gobble_i\else
    \edef\calx_lexcurr{#2.#3.}%
    \ifx\calx_lexcurr\calx_lexprev
        \XINT_global\expandafter
            \let\csname calx_lex\calx_lexfamily#1\expandafter\endcsname
            \csname calx_lex\calx_lexfamily\calx_lexindex\endcsname
    \else
        \expandafter\expandafter\expandafter\calx_lexMa\csname calx_lLb#2\endcsname
          \calx_lLQ#1.#2.#3.%
        \let\calx_lexprev\calx_lexcurr
        \def\calx_lexindex{#1}%
    \fi\fi
    \calx_lexLd
}%
```

P > 16 table macros \calx_lexB<i>: #1 k, #2 s xI, #3 s f1, #4 s f2 (R1 as P16)

```
\def\calx_lexMaB #1!#2!#3!#4\calx_lLQ#5.#6.#7.%
{%
    \XINT_global\expandafter\edef\csname calx_lexB#5\endcsname##1.##2!##3!##4!%
    {%
        \noexpand\expandafter\noexpand\calx_lexgSa\noexpand\the\noexpand\numexpr
        (((##2-2*##1)*10000-3025*##1)*10000+##3+#1-8509*##1)*100%
        +(-2994*##1+(##4+#2-456*##1)/10000)/100.#6;%
        #7;|##1.##2!##3!##4!%
    }%
}%
```

P <= 8 table macros \calx_lexC<i>: as \calx_lexA<i>, state R1.c;|k.

```
\def\calx_lexMaC #1!#2!#3!#4\calx_lLQ#5.#6.#7.%
{%
    \XINT_global\expandafter\edef\csname calx_lexC#5\endcsname##1.##2.##3.##4.##5.%
    {%
        \noexpand\expandafter\noexpand\calx_lexhSa\noexpand\the\noexpand\numexpr
        (((##5-2*##1)*10000-3025*##1)*10000+##2+#1-8509*##1)*100%
        +(-2994*##1+(##3+#2-456*##1)/10000)/100.%
        #7;|##1.%
```

```
    }%
}%
```

zero items +00000000! (k = 0..16) for the P > 16 layout

```
\def\calx_lLtmp #1.#2%
{%
    \expandafter\def\csname calx_lexgz#1\endcsname{#2}%
    \ifnum#1<16 \xint_afterfi{\expandafter\calx_lLtmp\the\numexpr#1+1.%
       {#2+00000000!}}\fi
}%
\calx_lLtmp0.{}%
```

schedules, P = 33..100 (P17..P32: explicit back end): n = 2m, K = (n-4)/2; start -C_K (scale 10^8); levels k = K-1..1 (b = m = n-4-2k, n_k = n-2-2k, C_k = 1/(k+1)!), level 0 (C_0 = 1, n-1 limbs), e^t level (C = 10^W, n+1 limbs, floor carries); \calx_lexT: Y = G e^t 10^-24 (U = G, 6 limbs; T = -e^t, n+1 items)

```
\def\calx_lexgK #1#2{\calx_ttcoef{calx_tcd\ifodd\numexpr#1+1\relax s\else c\fi}%
   {\ifodd\numexpr#1+1\relax(#1)/2\else(#1+1)/2\fi}{#2}}%
```

#1 b, #2 m; the following braced n goes directly to \calx_ttcoef. The coefficient continuation follows n.

```
\def\calx_lexgLv #1.#2.#3.#4.#5#6%
   {\ifnum#4=\xint_c_i\calx_tZlevx\calx_tZv{#1}{#2}{#3}{#5}#6\else
    \calx_tZlevx\calx_tZvn{#1}{#2}{#3}{#5}#6\fi}%
\def\calx_lexgSl #1.%
{%
    \ifnum#1>\xint_c_
      \expandafter\calx_lexgLv\the\numexpr\calx_lLWa-4-2*#1\expandafter.%
        \the\numexpr\calx_lLWa-4-2*#1\expandafter.\the\numexpr\calx_lLWa-2-2*#1%
        \expandafter.%
        \the\numexpr\ifnum\numexpr#1-\calx_lLWb\relax=\xint_c_ 1\else0\fi.%
        {\expandafter\calx_lLWrv\expanded{\calx_lexgK{#1}{\calx_lLWa-2-2*#1}%
          {\calx_lLWa-2-2*#1}\calx_ttq}\relax}\calx_tZun
      \expandafter\calx_lexgSl\the\numexpr#1-1.%
    \else
      \expandafter\calx_lexgLv\the\numexpr\calx_lLWa-4\expandafter.%
        \the\numexpr\calx_lLWa-4\expandafter.%
        \the\numexpr\calx_lLWa-1\expandafter.%
        \the\numexpr\ifnum\calx_lLWb=\xint_c_ 1\else0\fi.%
        {1!\expandafter\calx_lLWzr\the\numexpr\calx_lLWa-2.}\calx_tZun
      \expandafter\calx_lexgLv\the\numexpr\calx_lLWa-2\expandafter.%
        \the\numexpr\calx_lLWa-1\expandafter.\the\numexpr\calx_lLWa+1.0.%
        {1!\expandafter\calx_lLWzr\calx_lLWa.}\calx_tZuf
      \noexpand\calx_lexgF
    \fi
}%
\def\calx_lexgSt #1!#2!{-#2.\noexpand\calx_tZn-#1}%
\def\calx_lbuildex #1%
{%
    \edef\calx_lLWa{\the\numexpr2*((#1+3)/8)+2}% n
    \edef\calx_lLWb{\the\numexpr(\calx_lLWa-4)/2-1}% first level
    \XINT_global\edef\calx_lexS
    {%
      {\expandafter\calx_lexgSt\expanded{\calx_lexgK{\calx_lLWb+1}{2}{2}%
```

```
        \calx_ttq}}%
      {\expandafter\calx_lexgSl\calx_lLWb.}%
  }%
  \XINT_global\edef\calx_lexT##1##2%
  {%
    \noexpand\calx_tZd##2\noexpand\calx_tZq
      {##1\expandafter\calx_lLWzr\the\numexpr\calx_lLWa-6.}% U pad
    \expandafter\calx_lexgLv\expandafter6\expandafter.\the\numexpr
      \calx_lLWa+1\expandafter.\the\numexpr\calx_lLWa+1.1.%
      {\expandafter\calx_lLWzr\the\numexpr\calx_lLWa+1.}\calx_tZuf
    \noexpand\calx_lexgY
  }%
}%
```

33.14. \calxFpPowTen, \calxRawPowTen

pow10(x). x rounded as exp (P+14 digits, sliced at the grid); |x| = 8K + y, K = floor(n/8) (n integer part of |x|), y = d + phi, d = n mod 8 = (block I0) mod 8; 10^x = 10^(8sK) exp(s y ln10): x' = y ln10 (< 18.43) on exp's grid by one short product, then exp's fast path and e <- e + sK (\calx_lexHf / \calx_lexJf -> \calx_lexHz: Overflow, Underflow). Integer x (P <= 16 path): 10^(s n) written directly (\calx_lexHy).

```
\def\calx_lPowTen{\romannumeral0\expandafter\calx_lpwa
    \romannumeral0\CALX_fround{\calx_ldigits+14}}%
\def\calxRawPowTen #1{\romannumeral0\CALX_ftoraw{\calxFpPowTen{#1}}}%
\def\calx_lpwnfl #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\calxFpPowTen#3}}%
}%
\def\calx_lpwnex #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\calxRawPowTen#3}}%
}%
\expandafter\let\csname XINT_flexpr_func_pow10\endcsname\calx_lpwnfl
\expandafter\let\csname XINT_expr_func_pow10\endcsname\calx_lpwnex
\def\calx_lpwa #1#2{\csname calx_lpwt#1#2\endcsname}%
\expandafter\let
\csname calx_lpwtI+\expandafter\endcsname\csname calx_lextI+\endcsname
\expandafter\let
\csname calx_lpwtI-\expandafter\endcsname\csname calx_lextI-\endcsname
\expandafter\let
\csname calx_lpwtI?\expandafter\endcsname\csname calx_lextI?\endcsname
\expandafter\def
\csname calx_lpwtF+\endcsname{\calx_lexb pw;+}%
\expandafter\def
\csname calx_lpwtF-\endcsname{\calx_lexb pw;-}%
```

P <= 16 reader (layout of \calx_lexc): f3, f2, f1, I0, I1, rest, s; I1 > 21: |x| > 2.2e9, over/underflow; floor(I0/8) = (2 I0 - 7)/16

```
\def\calx_lpwG 1#1!1#2!1#3!1#4!1#5!%
```

```
{%
    \ifnum#5>21 \xint_dothis\calx_lpwG_over\fi
    \ifnum\numexpr#1+#2+#3\relax=\xint_c_
      \xint_dothis\calx_lpwG_int\fi
    \xint_orthat\calx_lpwG_regular 1#1!1#2!1#3!1#4!1#5!%
}%
\def\calx_lpwG_over 1#1!1#2!1#3!1#4!1#5!#6\xint:#7{\calx_lexO#7}%
\def\calx_lpwG_int 1#1!1#2!1#3!1#4!1#5!#6\xint:#7%
{%
    \expandafter\calx_lpwI\the\numexpr#4-8*((2*#4-7)/16)\expandafter.%
        \the\numexpr#5*12500000+(2*#4-7)/16.#7%
}%
\def\calx_lpwG_regular 1#1!1#2!1#3!1#4!1#5!#6\xint:#7%
{%
    \expandafter\calx_lpwP\expanded{\the\numexpr#4-8*((2*#4-7)/16)!%
      \calx_lpwh#3\calx_lpwh#2\calx_lpwh#1#7\the\numexpr#5*12500000+(2*#4-7)/16.}%
}%
\let\calx_lpwhG\calx_lpwG
\def\calx_lpwh #1#2#3#4#5#6#7#8{#1#2#3#4!#5#6#7#8!}%
```

d, y1..y6 (stream s K.): x' 10^24 = y 10^24 ln10, columns c0..c6, guard c7 (ln10 limbs 2|3025|8509|2994|0456|8401|7991|4546; dropped columns < 5 units 10^-24) -> \calx_lexHf X!l1!..!l6!sK.

```
\def\calx_lpwP #1!#2!#3!#4!#5!#6!#7!%
{%
    \expandafter\calx_lexHf\the\numexpr2*#1\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+3025*#1+2*#2\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+8509*#1+3025*#2+2*#3\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+2994*#1+8509*#2+3025*#3+2*#4\expandafter\calx_lLR
    \the\numexpr\calx_lLoff+456*#1+2994*#2+8509*#3+3025*#4+2*#5%
    \expandafter\calx_lLR
    \the\numexpr\calx_lLoff+8401*#1+456*#2+2994*#3+8509*#4+3025*#5+2*#6%
    \expandafter\calx_lLR
    \the\numexpr\calx_lLoff+7991*#1+8401*#2+456*#3+2994*#4+8509*#5+3025*#6%
      +2*#7\expandafter\calx_lLRz
    \the\numexpr\calx_lLoff+4546*#1+7991*#2+8401*#3+456*#4+2994*#5+8509*#6%
      +3025*#7\relax!%
}%
```

integer x = s n, n = 8K + d: 10^(s n) as e.D.block for \calx_lexHy

```
\def\calx_lpwI #1.#2.#3%
{%
    \expandafter\calx_lexHy\expanded{\if-#3%
      \ifnum#1=\xint_c_ \the\numexpr-#2.1.100000001!%
      \else\the\numexpr-#2-1.\the\numexpr9-#1.%
        \the\numexpr100000000+\CALX_pten{8-#1}!\fi
    \else\the\numexpr#2.\the\numexpr#1+1.\the\numexpr100000000+\CALX_pten{#1}!%
    \fi}\xint:%
}%
```

P > 16 readers (layout of \calx_lexgc: s, signed items MS first, \calx_lexQ): p <= 0 (I0 first), p = 1 (I1, I0)

```
\def\calx_lpwgG #1#2#3!{\calx_lpwS#1.00000000.#3.}%
\def\calx_lpwJa #1#2#3!#4#5!#6\calx_lexQ#7%
   {\ifnum#3>21 \xint_dothis{\calx_lexO#1}\fi
```

```
    \xint_orthat{\calx_lpwS#1.#3.#5.#6\calx_lexQ#7}}%
\def\calx_lpwS #1.#2.#3.%
{%
    \expandafter\calx_lpwV\the\numexpr#2*12500000+(2*#3-7)/16\expandafter.%
    \the\numexpr#3-8*((2*#3-7)/16).#1%
}%
```

K.d.s f-items\calx_lexQ: U list (d, limbs of the fraction items), kernel level \calx_lpwT (U = y, T = -ln10, C = 0, floor carries); stream s K.

```
\def\calx_lpwV #1.#2.#3#4\calx_lexQ%
{%
    \expandafter\calx_lpwT\expandafter
    {\expanded{\expandafter\calx_tZbt
     \expanded{#2!\calx_lpwL#4\calx_lexQ}\calx_tZq!}}%
    #3#1.%
}%
\def\calx_lpwL #1{\ifx\calx_lexQ#1\else\expandafter\calx_lpwLs\fi}%
\def\calx_lpwLs #1#2#3#4#5#6#7#8!{#1#2#3#4!#5#6#7#8!\calx_lpwL}%
```

{U}0.<n+1 limbs>\calx_tZq s K. -> \calx_lexJf 0!0000!<x' limbs>0000!

```
\def\calx_lpwF #1#2.#3\calx_tZq
    {\expandafter\calx_lexJf\expanded{0!0000!#3}0000!\calx_lLQ}%
```

load time: \calx_lpwT #1 (U list), P = 17..100, n = W/4: T = -ln10 on n+2 limbs (2, 3025, 8509, \calx_lexLx), level b = n+1, m = n+2, n+2 limbs (the kernel's top output column must be >= b+m-1: top limb 0)

```
\edef\calx_lpwlx{3025!8509!\calx_lexLx}%
\def\calx_lpwk #1.#2!{\ifnum#1>\xint_c_ \noexpand\calx_tZn-#2.\expandafter\calx_lpwk
   \the\numexpr#1-1\expandafter.\else\expandafter\calx_lpwz\fi}%
\def\calx_lpwz #1\relax{}%
\def\calx_lbuildpw #1%
{%
    \edef\calx_lLWa{\the\numexpr2*((#1+3)/8)+2}% n
    \XINT_global\edef\calx_lpwT##1%
    {%
      \noexpand\calx_tZd-2.\expandafter\calx_lpwk\the\numexpr\calx_lLWa+1%
        \expandafter.\calx_lpwlx\relax\noexpand\calx_tZq{##1}%
      \expandafter\calx_lexgLv\the\numexpr\calx_lLWa+1\expandafter.\the\numexpr
        \calx_lLWa+2\expandafter.\the\numexpr\calx_lLWa+2.1.%
        {\expandafter\calx_lLWzr\the\numexpr\calx_lLWa+2.}\calx_tZuf
      \noexpand\calx_lpwF
    }%
}%
```

33.15. \calxFpExpMinusOne, \calxRawExpMinusOne

expm1(x) = exp(x) - 1. x rounded to P digits. E = decimal exponent: E <= -5: x E1(x), E1 = sum x^k/(k+1)!, log1p's C machinery (kind em); E >= 3: exp(x) (x > 0) or -1; else exp's front with kind em: readers \calx_lemG (P <= 16), \calx_lemgG (P > 16): |x| >= T_P = (P+4) ln10 (up): exp (x > 0) or -1; 0 < x < 0.04: bucket-0 table macro with k = 0; else exp's normal branch; the writers then call \calx_lemF (marker \calx_lemMk): N = Y' - 10^(W'-k) exact, ONE rounding.

```
\chardef\calx_lexMk=1 \chardef\calx_lemMk=2 \let\calx_lpwMk\calx_lexMk
\chardef\calx_leuMk=3 \chardef\calx_lxuMk=4 % unrounded expm1, exp
\def\calx_lExpMinusOne{\romannumeral0\expandafter\calx_lema
```

```
    \romannumeral0\CALX_fround{\calx_ldigits}}%
\def\calxRawExpMinusOne #1{\romannumeral0\CALX_ftoraw{\calxFpExpMinusOne{#1}}}%
\def\calx_lemnfl #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\calxFpExpMinusOne#3}}%
}%
\def\calx_lemnex #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\calxRawExpMinusOne#3}}%
}%
\expandafter\let\csname XINT_flexpr_func_expm1\endcsname\calx_lemnfl
\expandafter\let\csname XINT_expr_func_expm1\endcsname\calx_lemnex
\def\calx_lema #1#2{\csname calx_lemt#1#2\endcsname}%
\def\calx_lemT{\numexpr(\calx_ldigits+4)*23026/10000+1\relax}% T_P
\begingroup\catcode`F=12
\expandafter\gdef\csname calx_lemtI+\endcsname#1\xint:{ I+}%
\expandafter\gdef\csname calx_lemtI-\endcsname#1\xint:{ F-1.0.100000001!}%
\expandafter\gdef\csname calx_lemtI?\endcsname#1\xint:{ I?}%
\expandafter\gdef\csname calx_lemtF+\endcsname{\calx_lemb+}%
\expandafter\gdef\csname calx_lemtF-\endcsname{\calx_lemb-}%
```

s D.e.blocks; D + 8e = E + 1

```
\gdef\calx_lemb #1#2.#3.%
{%
    \ifnum#2=\xint_c_ \xint_dothis\calx_lemb_zero\fi
    \ifnum\numexpr#2+8*#3\relax<-\xint_c_iii
      \xint_dothis\calx_lemb_small\fi
    \ifnum\numexpr#2+8*#3\relax>\xint_c_iii
      \xint_dothis\calx_lemb_large\fi
    \xint_orthat\calx_lemb_regular #1#2.#3.%
}%
\gdef\calx_lemb_zero #1#2.#3.#4\xint:{ F+0.0.}%
\gdef\calx_lemb_small #1#2.#3.#4\xint:{\calx_llpC eml;#1#2.#3.#4\xint:}%
\gdef\calx_lemb_large #1#2.#3.#4\xint:
    {\calx_lemL#1{\calx_lexb ex;#1#2.#3.#4\xint:}}%
\gdef\calx_lemb_regular #1#2.#3.#4\xint:{\calx_lexb em;#1#2.#3.#4\xint:}%
```

s{exp's computation}: x > 0: the computation, x < 0: -1

```
\gdef\calx_lemL #1%
   {\if-#1\expandafter\xint_firstoftwo\else\expandafter\xint_secondoftwo\fi
    {\calx_lemM}{\xint_firstofone}}%
\gdef\calx_lemM #1{ F-1.0.100000001!}%
\endgroup
```

P <= 16 reader (layout of \calx_lexc, |x| < 1000): f3, f2, f1, xI, rest, s

```
\def\calx_lemG 1#1!1#2!1#3!1#4!#5\xint:#6%
{%
    \ifnum#4<\calx_lemT\else
      \xint_dothis{\calx_lemL#6{\calx_lexG1#1!1#2!1#3!1#4!#5\xint:#6}}\fi
    \ifnum\numexpr#61*(#4*100+#3/1000000)\relax<4
```

```
      \xint_dothis{\if-#6\expandafter\calx_lexGn\else\expandafter\calx_lemA\fi
        #1.#2.#3.#4.#6A\calx_lemMk}\fi
    \xint_orthat{\calx_lexGn#1.#2.#3.#4.#6A\calx_lemMk}%
}%
\let\calx_lemhG\calx_lemG
```

0 < x < 0.04: bucket 0 (c = 10^4, no constant) with k = 0

```
\def\calx_lemA #1.#2.#3.#4.#5#6%
   {\csname calx_lexA0\endcsname 0.#5#3.#5#2.#5#1.#5#4.}%
```

P > 16 reader (layout of \calx_lexgc): s, s xI, s f1, s f2, rest\calx_lexQ \calx_lemMk

```
\def\calx_lemgG #1#2!#3!%
{%
    \ifnum#1#2<\calx_lemT\else\xint_dothis\calx_lemgG_large\fi
    \ifnum\numexpr#2*100+#3/1000000\relax<4
      \xint_dothis\calx_lemgG_small\fi
    \xint_orthat\calx_lemgG_regular #1#2!#3!%
}%
\def\calx_lemgG_large #1#2!#3!#4!{\calx_lemgL#1#2!#3!#4!}%
\def\calx_lemgG_small #1#2!#3!#4!%
{%
    \if-#1\expandafter\calx_lexgG\else\expandafter\calx_lemgA\fi
       #1#2!#3!#4!%
}%
\def\calx_lemgG_regular #1#2!#3!#4!{\calx_lexgG#1#2!#3!#4!}%
\def\calx_lemgA #1#2!#3!#4!{\csname calx_lexB0\endcsname 0.#2!#3!#4!}%
```

|x| >= T_P: x > 0 exp (marker of exp's writer), x < 0 -1

```
\def\calx_lemgL #1#2\calx_lexQ#3{\calx_lemL#1{\calx_lexgG#1#2\calx_lexQ\calx_lexMk}}%
\let\calx_lemJa\calx_lemgL
```

finisher: digits of Y'.k.W'. -> exp x - 1 = round_P(Y' 10^(k-W') - 1), the subtraction exact (\CALX_fsubx, native 1), ONE rounding

```
\def\calx_lemF #1.#2.#3.{\expandafter\calx_lemZ\the\numexpr#2-#3.#1.}%
\def\calx_lemZ #1.#2.#3%
{%
    \ifcase#3\or\or\expandafter\calx_lemZr\or\expandafter\calx_lemZu
    \else\expandafter\calx_lemZx\fi{#2[#1]}%
}%
\begingroup\catcode`F=12
\gdef\calx_lemZr #1%
   {\CALX_fnround\calx_ldigits{\CALX_fsubx{\CALX_ffromraw{#1}}{F+1.0.100000001!}}}%
\gdef\calx_lemZu #1{\CALX_fsubx{\CALX_ffromraw{#1}}{F+1.0.100000001!}}%
\endgroup
\def\calx_lemZx{\CALX_ffromraw}%
```

P17..P32 writers: T!q..!q1! then q0!k. -> digits.k.W.

```
\def\calx_lemXa #1!#2!#3!#4!#5!#6!#7!{\calx_lemXc#1#2#3#4#5#6#7}%
\def\calx_lemXb #1!#2!#3!#4!#5!#6!#7!#8!#9!{\calx_lemXc#1#2#3#4#5#6#7#8#9}%
\def\calx_lemXc #1!#2.#3{\calx_lemF#1.#2.8*\calx_lexgm.#3}%
```

series |x| < 10^-4: U = -10^4 x (lists of log1p, sign swapped), schedules \calx_lemS (log's unity levels, C_k = 1/(k+1)! by \calx_lexgK)

```
\expandafter\let\csname calx_lemU+%
     \expandafter\endcsname\csname calx_llpU-\endcsname
\expandafter\let\csname calx_lemU-%
```

```
     \expandafter\endcsname\csname calx_llpU+\endcsname
\def\calx_lbuildem #1%
{%
    \edef\calx_lLWa{\the\numexpr(#1+\ifnum#1>64 6\else5\fi)/4}%
    \edef\calx_lLWm{\the\numexpr\calx_lLWa-1}%
    \XINT_global\edef\calx_lemS##1%
    {%
      \noexpand\calx_tZd\expandafter\calx_tZhd\expanded{\calx_lexgK
        {\calx_lLWa-1}{1}{1}\calx_ttq}\relax.\noexpand\calx_tZq{##1}%
      \expandafter\calx_lLWbl\the\numexpr\calx_lLWa-2.%
    }%
}%
\let\calx_lLtmpb\relax
```

33.16. \calxFpPow, \calxRawPow

Powers a**b, a^b, pow(a, b) of \xintfloateval. y = exp(z), z = b' ln a': a' = a rounded to P digits, b' = b rounded to P + 14 digits (|z| < 5e9 is the only range that matters: |z| rho_b <= 2.5e-(P+4)); ln a' by the unrounded internal log at P' = P + 2 + g (log1p's \calx_llpLP), 10^g >= |z| (bound from the exponents, g <= 10); z = b' ln a' exact (\CALX_fmulx), then exp's entry (x at P + 14 digits, one rounding). The integer-exponent path is in calx.sty; in this scientific path integer b gives the sign for a < 0. Specials: IEEE-754 pow table (a = 1 or b = 0: 1).

```
\def\calx_lPow #1#2{\romannumeral0\expandafter\calx_lppa
    \romannumeral0\CALX_fround{\calx_ldigits+14}{#2}{#1}}%
\def\calxRawPow #1#2{\romannumeral0\CALX_ftoraw{\calxFpPow{#1}{#2}}}%
\def\XINT_flexpr_func_pow #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:two
    {\romannumeral`&&@\calxFpPow#3}}%
}%
```

b'\xint:{a} -> a'\xint:b'\xint:, dispatch on <tag a><sign a><tag b>

```
\def\calx_lppa #1\xint:#2%
   {\expandafter\calx_lppb\romannumeral0\CALX_fround{\calx_ldigits}{#2}#1\xint:}%
\def\calx_lppb #1#2#3\xint:#4{\csname calx_lppt#1#2#4\endcsname#3\xint:}%
```

d = digits of 24 (|E_a| + 1) (|E_a| < 4166666, else 11): |z| < 10^(E_b + 1) 2.303 (|E_a| + 1) < 10^(E_b + d); g = E_b + d in [0, 10]

```
\def\calx_lppc #1.#2.%
{%
    \expandafter\calx_lppd\the\numexpr\calx_ldigits+\xint_c_ii+\expandafter\calx_lppg
    \the\numexpr#2+\ifnum\ifnum#1<\xint_c_ -\fi#1<4166666
      \CALX_ndig{24*(\ifnum#1<\xint_c_ -\fi#1+1)}\else 11\fi.%
    .%
}%
\def\calx_lppg #1.{\ifnum#1<\xint_c_ 0\else\ifnum#1>10 10\else#1\fi\fi}%
```

P'.b'\xint:+<a>\xint: -> exp(b' ln a')

```
\def\calx_lppd #1.#2\xint:#3\xint:%
{%
    \expandafter\calx_lexa\romannumeral0\CALX_fround{\calx_ldigits+14}%
      {\CALX_fmulx{#2}{\calx_llpLP\numexpr4*((#1+1)/4)\relax.#3\xint:}}%
}%
```

```
\def\calx_lppz #1%
   {\if-#1\xint_dothis{\CALX_nsig{DivisionByZero}{0 ** b, b < 0}{+}}\fi
    \xint_orthat{ F+0.0.}}%
```

T\xint:b\xint:<even><odd><not integer>: T = trunc(b) (\CALX_ti_f), b an integer iff T = b (\CALX_fcmp); parity from T's block at 10^0

```
\def\calx_lppi #1\xint:#2\xint:#3#4#5%
   {\ifcase\CALX_fcmp{#1}{#2} \xint_afterfi{\calx_lppj#1\xint:#3#4}%
    \else\xint_afterfi{#5}\fi}%
```

|a| ** b, negated (b odd); not an integer: InvalidOperation

```
\def\calx_lppE{\csname calx_lpptF+F\endcsname}%
\def\calx_lppO #1\xint:#2\xint:%
   {\expandafter\CALX_fopp_a\romannumeral0\csname calx_lpptF+F\endcsname
    #1\xint:#2\xint:\xint:}%
\def\calx_lppx #1\xint:#2\xint:%
   {\CALX_nsig{InvalidOperation}{a ** b, a < 0, b not an integer}{}}%
\def\calx_lppP{ I+}%
\def\calx_lppM{ I-}%
```

|a| vs 1, b = I+, I-, I?

```
\def\calx_lppy #1#2#3%
   {\if ?#1\xint_dothis{ I?}\fi\if#1#2\xint_dothis{ I+}\fi
    \xint_orthat{ F+0.0.}}%
\begingroup\catcode`F=12
\gdef\calx_lppj F#1#2.#3.#4!%
{%
    \ifnum#3>\xint_c_ \xint_dothis\calx_lppj_positive\fi
    \ifodd#4 \xint_dothis\calx_lppj_odd\fi
    \xint_orthat\calx_lppj_regular F#1#2.#3.#4!%
}%
\gdef\calx_lppj_positive F#1#2.#3.#4!#5\xint:#6#7{#6}%
\gdef\calx_lppj_odd F#1#2.#3.#4!#5\xint:#6#7{#7}%
\gdef\calx_lppj_regular F#1#2.#3.#4!#5\xint:#6#7{#6}%
```

a >= 0, b finite: <Da>.<ea>.<blocks>\xint:<sb><Db>.<eb>.<blocks>\xint:

```
\expandafter\gdef\csname calx_lpptF+F\endcsname #1.#2.#3\xint:#4#5.#6.#7\xint:%
{%
    \ifnum#5=\xint_c_ \xint_dothis{ F+1.0.100000001!}\fi
    \ifnum#1=\xint_c_ \xint_dothis{\calx_lppz#4}\fi
    \xint_orthat{\expandafter\calx_lppc\the\numexpr#1-\xint_c_i+\xint_c_viii*#2%
      \expandafter.\the\numexpr#5-\xint_c_i+\xint_c_viii*#6.%
      F#4#5.#6.#7\xint:+#1.#2.#3\xint:}%
}%
\expandafter\gdef\csname calx_lpptF-F\endcsname #1.#2.#3\xint:#4#5.#6.#7\xint:%
{%
    \ifnum#5=\xint_c_ \xint_dothis{ F+1.0.100000001!}\fi
    \xint_orthat
      {\expandafter\calx_lppi\romannumeral0\CALX_ti_f F#4#5.#6.#7\xint:%
      \xint:F#4#5.#6.#7\xint:\calx_lppE\calx_lppO\calx_lppx
      #1.#2.#3\xint:#4#5.#6.#7\xint:}%
}%
\expandafter\gdef\csname calx_lpptF+I\endcsname #1\xint:#2\xint:%
{%
    \ifcase\CALX_fcmp{F+#1}{\CALX_fonec} \xint_afterfi{ F+1.0.100000001!}%
```

```
    \or\xint_afterfi{\calx_lppy#2+-}%
    \else\xint_afterfi{\calx_lppy#2-+}\fi
}%
\expandafter\gdef\csname calx_lpptI+F\endcsname\xint:#1#2.%
{%
    \ifnum#2=\xint_c_ \xint_dothis\calx_lppt_posinf_zero\fi
    \if-#1\xint_dothis\calx_lppt_posinf_negative\fi
    \xint_orthat\calx_lppt_posinf_positive #1#2.%
}%
\gdef\calx_lppt_posinf_zero #1#2.#3\xint:{ F+1.0.100000001!}%
\gdef\calx_lppt_posinf_negative #1#2.#3\xint:{ F+0.0.}%
\gdef\calx_lppt_posinf_positive #1#2.#3\xint:{ I+}%
\expandafter\gdef\csname calx_lpptI-F\endcsname\xint:#1#2.%
{%
    \ifnum#2=\xint_c_ \xint_dothis\calx_lppt_neginf_zero\fi
    \if-#1\xint_dothis\calx_lppt_neginf_negative\fi
    \xint_orthat\calx_lppt_neginf_regular #1#2.%
}%
\gdef\calx_lppt_neginf_zero #1#2.#3.#4\xint:{ F+1.0.100000001!}%
\gdef\calx_lppt_neginf_negative #1#2.#3.#4\xint:{ F+0.0.}%
\gdef\calx_lppt_neginf_regular #1#2.#3.#4\xint:%
   {\expandafter\calx_lppi\romannumeral0\CALX_ti_f F#1#2.#3.#4\xint:%
      \xint:F#1#2.#3.#4\xint:\calx_lppP\calx_lppM\calx_lppP}%
\expandafter\gdef\csname calx_lpptI?F\endcsname\xint:#1#2.%
{%
    \ifnum#2=\xint_c_ \xint_dothis\calx_lppt_nan_zero\fi
    \xint_orthat\calx_lppt_nan_regular #1#2.%
}%
\gdef\calx_lppt_nan_zero #1#2.#3\xint:{ F+1.0.100000001!}%
\gdef\calx_lppt_nan_regular #1#2.#3\xint:{ I?}%
\expandafter\gdef\csname calx_lpptI+I\endcsname\xint:#1\xint:%
   {\if ?#1\xint_dothis{ I?}\fi\if-#1\xint_dothis{ F+0.0.}\fi
    \xint_orthat{ I+}}%
\expandafter\gdef\csname calx_lpptI?I\endcsname\xint:#1\xint:{ I?}%
\endgroup
\expandafter\let\csname calx_lpptF-I\expandafter\endcsname
   \csname calx_lpptF+I\endcsname
\expandafter\let\csname calx_lpptI-I\expandafter\endcsname
   \csname calx_lpptI+I\endcsname
```

Unrounded exp and expm1 (used by the hyperbolic functions): kinds xu (exp, |x| < 10^4) and eu (expm1, |x| < 1000) of exp's front; the writers pass the digits of Y' to \calx_lemF with the marker \calx_lxuMk or \calx_leuMk, the finisher \calx_lemZ then gives Y' 10^(k-W') or Y' 10^(k-W') - 1 exactly (no rounding); expm1 |x| < 10^-4: log1p's C machinery with the output letter u (\calx_tshou: no rounding).

```
\def\calx_lxuG 1#1!1#2!1#3!1#4!#5\xint:#6{\calx_lexGn#1.#2.#3.#4.#6A\calx_lxuMk}%
\let\calx_lxuhG\calx_lxuG \let\calx_lxugG\calx_lexgG \let\calx_lxuJa\calx_lexJa
\def\calx_leuG 1#1!1#2!1#3!1#4!#5\xint:#6%
{%
    \ifnum#4<\calx_lemT\else
      \xint_dothis{\calx_lemL#6{\calx_lexG1#1!1#2!1#3!1#4!#5\xint:#6}}\fi
    \ifnum\numexpr#61*(#4*100+#3/1000000)\relax<4
      \xint_dothis{\if-#6\expandafter\calx_lexGn\else\expandafter\calx_lemA\fi
```

```
        #1.#2.#3.#4.#6A\calx_leuMk}\fi
    \xint_orthat{\calx_lexGn#1.#2.#3.#4.#6A\calx_leuMk}%
}%
\let\calx_leuhG\calx_leuG \let\calx_leugG\calx_lemgG \let\calx_leuJa\calx_lemJa
\begingroup\catcode`F=12
\gdef\calx_lhyX #1{\expandafter\calx_lhyXa\romannumeral`&&@#1\xint:}%
\gdef\calx_lhyXa F#1#2.#3.#4\xint:{\calx_lexb xu;#1#2.#3.#4\xint:}%
\gdef\calx_lhyU #1{\expandafter\calx_lhyUa\romannumeral`&&@#1\xint:}%
\gdef\calx_lhyUa F#1#2.#3.%
{%
    \ifnum\numexpr#2+\xint_c_viii*#3<-\xint_c_iii
      \xint_dothis\calx_lhyUa_small\fi
    \xint_orthat\calx_lhyUa_regular F#1#2.#3.%
}%
\gdef\calx_lhyUa_small F#1#2.#3.#4\xint:{\calx_llpC emu;#1#2.#3.#4\xint:}%
\gdef\calx_lhyUa_regular F#1#2.#3.#4\xint:{\calx_lexb eu;#1#2.#3.#4\xint:}%
\endgroup
\def\calx_lhyln{\calx_llpLP\calx_ldigits+\xint_c_viii.+1.0.100000002!\xint:}%
```

33.17. \calxFpSinh, \calxFpCosh, \calxFpTanh, \calxFpCoth, \calxFpSech, \calxFpCsch, \calxFpAsinh, \calxFpAcosh, \calxFpAtanh, \calxFpAcoth, \calxFpAsech, \calxFpAcsch

Hyperbolic functions and their inverses (and the \calxRaw... variants, expression functions sinh, cosh, ...), composed from exp, expm1, log1p, the internal log and calxfp arithmetic at Q = P + 4 digits, with one final rounding; formulas chosen to avoid cancellation. x is rounded to P digits, a = |x|, E its decimal exponent, u = expm1(x), w = expm1(-x):

    sinh E < 0: (u - w)/2; E <= 2: (e^x - e^-x)/2; else sign e^(a-ln2)
    cosh (e^x + e^-x)/2; E > 2: e^(a-ln2)
    tanh (u - w)/(u + w + 2); E > 2: sign 1; coth the inverse ratio
    sech 2/(e^x + e^-x); csch 2/(u - w) or 2/(e^x - e^-x); E > 2: e^(ln2-a)
    asinh sign log1p(a + a^2/(1 + sqrt(1 + a^2))); E >= Q: log1p(2a)
    acosh log1p(t + sqrt(t(t+2))), t = x - 1; E >= Q: log1p(2x)
    atanh sign log1p(q/(1 + sqrt(1 + q))), q = 2a/(1 - a)
    acoth the same with q = 2/(a - 1)
    asech acosh(1/x) with t = (1 - x)/x; E < -Q/2: log1p((2 - x)/x)
    acsch asinh(1/a) with sign; E >= Q/2: 1/x; E < -Q/2: log1p((2-a)/a)

asinh, atanh: E < -Q/2 gives x. log1p is applied to its Q-digit argument without rounding it to P digits (its three paths accept it). Entry \calx_lhyd<code><tag><sign> -> \csname calx_lhy<code><tag><sign> with Q. in front (Q = P + 4, P < 8 counts as 8); codes S C T K E H (sinh .. csch), s c t k e h (asinh .. acsch); the F macros of both signs come from the template \calx_lhyT.

```
\edef\calx_lhytwo{\romannumeral0\CALX_ffromraw{2[0]}}%
\edef\calx_lhyup{\romannumeral0\CALX_ffromraw{1[0]}}%
\edef\calx_lhyum{\romannumeral0\CALX_ffromraw{-1[0]}}%
\edef\calx_lhyzero{\romannumeral0\CALX_ffromraw{0[0]}}%
\expandafter\let\csname calx_lhyu+\endcsname\calx_lhyup
\expandafter\let\csname calx_lhyu-\endcsname\calx_lhyum
\def\calx_lhyd #1#2#3%
   {\expandafter\calx_lhye\the\calx_lhyQ.#1#2#3}%
\def\calx_lhye #1.#2#3#4%
   {\csname calx_lhy#2\calx_lhykind#3#4\endcsname#1.#3#4}%
```

Infinities and NaN: \calx_lhyIp (the input), \calx_lhyIa (I+), \calx_lhyIo (sign 1), \calx_lhyIz (0), \calx_lhyIn (InvalidOperation).

```
\def\calx_lhyIp #1.#2#3\xint:{ #2#3}%
\def\calx_lhyIa #1.#2#3\xint:{ #2+}%
\def\calx_lhyIo #1.#2#3\xint:%
   {\expandafter\expandafter\expandafter\space\csname calx_lhyu#3\endcsname}%
\def\calx_lhyIz #1\xint:{\expandafter\space\calx_lhyzero}%
\def\calx_lhyIn #1\xint:%
   {\CALX_nsig{InvalidOperation}{inverse hyperbolic function of inf}{}}%
\def\calx_lhyIs #1#2#3%
{%
    \expandafter\let\csname calx_lhy#1I+\endcsname#2%
    \expandafter\let\csname calx_lhy#1I-\endcsname#3%
    \expandafter\let\csname calx_lhy#1I?\endcsname\calx_lhyIp
    \expandafter\let\csname calx_lhy#1wI+\endcsname#2%
    \expandafter\let\csname calx_lhy#1wI-\endcsname#3%
    \expandafter\let\csname calx_lhy#1wI?\endcsname\calx_lhyIp
}%
\calx_lhyIs S\calx_lhyIp\calx_lhyIp \calx_lhyIs C\calx_lhyIa\calx_lhyIa
\calx_lhyIs T\calx_lhyIo\calx_lhyIo \calx_lhyIs K\calx_lhyIo\calx_lhyIo
\calx_lhyIs E\calx_lhyIz\calx_lhyIz \calx_lhyIs H\calx_lhyIz\calx_lhyIz
\calx_lhyIs s\calx_lhyIp\calx_lhyIp \calx_lhyIs c\calx_lhyIp\calx_lhyIn
\calx_lhyIs t\calx_lhyIn\calx_lhyIn \calx_lhyIs k\calx_lhyIz\calx_lhyIz
\calx_lhyIs e\calx_lhyIn\calx_lhyIn \calx_lhyIs h\calx_lhyIz\calx_lhyIz
```

Helpers (e^x, u, w: unrounded \calx_lhyX, \calx_lhyU; ln 2 \calx_lhyln at P + 8). \calx_lhyNp, \calx_lhyNm: sign. \calx_lhyP<f>{x1}{x2}<cont>: <cont> f(x2)\xint:f(x1)\xint:. \calx_lhyLp{w}: log1p(w) (w >= 0.1: the internal log at Q of round_Q(1 + w)); \calx_lhyLm{w}: -log1p(w). \calx_lhyS{b}{Q}<L>: <L> of b + b^2/(1 + sqrt(1 + b^2)). \calx_lhyq q\xint:Q.<L>: <L> of q/(1 + sqrt(1 + q)). \calx_lhyW{t}{Q}: t + sqrt(t(t+2)).

```
\let\calx_lhyNp\xint_firstofone
\def\calx_lhyNm #1{\expandafter\CALX_fopp_a\romannumeral0#1\xint:}%
\def\calx_lhyP #1#2#3#4%
   {\expandafter\calx_lhyPa\romannumeral0#1{#2}\xint:#1{#3}#4}%
\def\calx_lhyPa #1\xint:#2#3#4%
   {\expandafter#4\romannumeral0#2{#3}\xint:#1\xint:}%
\def\calx_lhyLp #1{\expandafter\calx_lhyLa\romannumeral`&&@#1\xint:}%
\def\calx_lhyLa #1#2#3.#4.%
{%
    \ifnum\numexpr#3+\xint_c_viii*#4>-\xint_c_i
      \xint_dothis\calx_lhyLa_large\fi
    \xint_orthat\calx_lhyLa_regular #1#2#3.#4.%
}%
\def\calx_lhyLa_large #1#2#3.#4.#5\xint:%
   {\expandafter\calx_lhyLc
      \the\numexpr\calx_ldigits+\xint_c_iv.#1#2#3.#4.#5\xint:}%
\def\calx_lhyLa_regular #1#2#3.#4.#5\xint:{\calx_llpa#1#2#3.#4.#5\xint:}%
\def\calx_lhyLc #1.#2\xint:%
   {\CALX_fnround{\calx_ldigits}{\expandafter\calx_lhyLd
    \romannumeral0\CALX_fadd{#1}{\calx_lhyup}{#2}\xint:#1.}}%
\begingroup\catcode`F=12
\gdef\calx_lhyLd F#1\xint:#2.{\calx_llpLP#2.#1\xint:}%
\endgroup
```

```
\def\calx_lhyLm #1%
   {\expandafter\CALX_fopp_a\romannumeral0\calx_lhyLp{#1}\xint:}%
\def\calx_lhyS #1#2%
   {\expandafter\calx_lhySa\romannumeral0\CALX_fmul{#2}{#1}{#1}\xint:%
    {#1}{#2}}%
\def\calx_lhySa #1\xint:#2#3#4%
{%
    #4{\CALX_fadd{#3}{#2}{\CALX_fdiv{#3}{#1}{\CALX_fadd{#3}{\CALX_fonec}%
      {\CALX_fsqrt{#3}{\CALX_fadd{#3}{\CALX_fonec}{#1}}}}}}%
}%
\def\calx_lhyb #1\xint:#2.{\calx_lhyS{#1}{#2}}%
\def\calx_lhyq #1\xint:#2.#3%
{%
    #3{\CALX_fdiv{#2}{#1}{\CALX_fadd{#2}{\CALX_fonec}{\CALX_fsqrt{#2}%
      {\CALX_fadd{#2}{\CALX_fonec}{#1}}}}}%
}%
\def\calx_lhyW #1#2%
   {\CALX_fadd{#2}{#1}{\CALX_fsqrt{#2}{\CALX_fmul{#2}{#1}%
      {\CALX_faddx{#1}{\calx_lhytwo}}}}}%
```

Final ratios: w\xint:u\xint:Q. (tanh, coth, csch), e^-x\xint:e^x\xint:Q. (sech, csch).

```
\def\calx_lhyTt #1\xint:#2\xint:#3.%
   {\CALX_fnround{\calx_ldigits}{\CALX_fdiv{#3}{\CALX_fsubx{#2}{#1}}%
      {\CALX_faddx{\CALX_faddx{#2}{#1}}{\calx_lhytwo}}}}%
\def\calx_lhyKt #1\xint:#2\xint:#3.%
   {\CALX_fnround{\calx_ldigits}{\CALX_fdiv{#3}%
      {\CALX_faddx{\CALX_faddx{#2}{#1}}{\calx_lhytwo}}{\CALX_fsubx{#2}{#1}}}}%
\def\calx_lhyEt #1\xint:#2\xint:#3.%
   {\CALX_fnround{\calx_ldigits}{\CALX_fdiv{#3}{\calx_lhytwo}%
      {\CALX_faddx{#2}{#1}}}}%
\def\calx_lhyHt #1\xint:#2\xint:#3.%
   {\CALX_fnround{\calx_ldigits}{\CALX_fdiv{#3}{\calx_lhytwo}%
      {\CALX_fsubx{#2}{#1}}}}%
```

\calx_lhyT<sign><opposite><sign 1><log1p wrapper><sign>: the F macros. Parameters: Q.F<sign>D.e.blocks\xint:, v = D + 8e = E + 1.

```
\def\calx_lhyT #1#2#3#4#5%
{%
\expandafter\def\csname calx_lhySF#1\endcsname ##1.##2##3##4.##5.%
{%
    \ifnum##4=\xint_c_ \xint_dothis{\csname calx_lhySF#1a\endcsname}\fi
    \ifnum\numexpr##4+\xint_c_viii*##5>\xint_c_iii
      \xint_dothis{\csname calx_lhySF#1b\endcsname}\fi
    \ifnum\numexpr##4+\xint_c_viii*##5<\xint_c_i
      \xint_dothis{\csname calx_lhySF#1c\endcsname}\fi
    \xint_orthat{\csname calx_lhySF#1d\endcsname}##1.##2##3##4.##5.%
}%
\expandafter\def\csname calx_lhySF#1a\endcsname ##1.##2##3##4.##5.##6\xint:%
{%
 ##2+0.0.%
}%
\expandafter\def\csname calx_lhySF#1b\endcsname ##1.##2##3##4.##5.##6\xint:%
{%
#5{\calx_lExp{\CALX_fsubx{##2+##4.##5.##6}%
```

```
        {\calx_lhyln}}}%
}%
\expandafter\def\csname calx_lhySF#1c\endcsname ##1.##2##3##4.##5.##6\xint:%
{%
\CALX_fnround{\calx_ldigits}{\CALX_fhalf{\CALX_fsubx
        {\calx_lhyU{##2#1##4.##5.##6}}{\calx_lhyU{##2#2##4.##5.##6}}}}%
}%
\expandafter\def\csname calx_lhySF#1d\endcsname ##1.##2##3##4.##5.##6\xint:%
{%
\CALX_fnround{\calx_ldigits}{\CALX_fhalf{\CALX_fsubx
        {\calx_lhyX{##2#1##4.##5.##6}}{\calx_lhyX{##2#2##4.##5.##6}}}}%
}%
\expandafter\def\csname calx_lhyCF#1\endcsname ##1.##2##3##4.##5.%
{%
    \ifnum##4=\xint_c_ \xint_dothis{\csname calx_lhyCF#1a\endcsname}\fi
    \ifnum\numexpr##4+\xint_c_viii*##5>\xint_c_iii
      \xint_dothis{\csname calx_lhyCF#1b\endcsname}\fi
    \xint_orthat{\csname calx_lhyCF#1c\endcsname}##1.##2##3##4.##5.%
}%
\expandafter\def\csname calx_lhyCF#1a\endcsname ##1.##2##3##4.##5.##6\xint:%
{%
\expandafter\space\calx_lhyup%
}%
\expandafter\def\csname calx_lhyCF#1b\endcsname ##1.##2##3##4.##5.##6\xint:%
{%
\calx_lExp{\CALX_fsubx{##2+###4.##5.##6}{\calx_lhyln}}%
}%
\expandafter\def\csname calx_lhyCF#1c\endcsname ##1.##2##3##4.##5.##6\xint:%
{%
\CALX_fnround{\calx_ldigits}{\CALX_fhalf{\CALX_faddx
        {\calx_lhyX{##2#1##4.##5.##6}}{\calx_lhyX{##2#2##4.##5.##6}}}}%
}%
\expandafter\def\csname calx_lhyTF#1\endcsname ##1.##2##3##4.##5.%
{%
    \ifnum##4=\xint_c_ \xint_dothis{\csname calx_lhyTF#1a\endcsname}\fi
    \ifnum\numexpr##4+\xint_c_viii*##5>\xint_c_iii
      \xint_dothis{\csname calx_lhyTF#1b\endcsname}\fi
    \xint_orthat{\csname calx_lhyTF#1c\endcsname}##1.##2##3##4.##5.%
}%
\expandafter\def\csname calx_lhyTF#1a\endcsname ##1.##2##3##4.##5.##6\xint:%
{%
 ##2+0.0.%
}%
\expandafter\def\csname calx_lhyTF#1b\endcsname ##1.##2##3##4.##5.##6\xint:%
{%
\expandafter\space#3%
}%
\expandafter\def\csname calx_lhyTF#1c\endcsname ##1.##2##3##4.##5.##6\xint:%
{%
\calx_lhyP\calx_lhyU{##2#1##4.##5.##6}%
      {##2#2##4.##5.##6}\calx_lhyTt##1.%
}%
```

```
\expandafter\def\csname calx_lhyKF#1\endcsname ##1.##2##3##4.##5.%
{%
    \ifnum##4=\xint_c_
      \xint_dothis{\csname calx_lhyKF#1a\endcsname}\fi
    \ifnum\numexpr##4+\xint_c_viii*##5>\xint_c_iii
      \xint_dothis{\csname calx_lhyKF#1b\endcsname}\fi
    \xint_orthat{\csname calx_lhyKF#1c\endcsname}##1.##2##3##4.##5.%
}%
\expandafter\def\csname calx_lhyKF#1a\endcsname ##1.##2##3##4.##5.##6\xint:%
{%
\CALX_nsig{DivisionByZero}{coth(0)}{+}%
}%
\expandafter\def\csname calx_lhyKF#1b\endcsname ##1.##2##3##4.##5.##6\xint:%
{%
\expandafter\space#3%
}%
\expandafter\def\csname calx_lhyKF#1c\endcsname ##1.##2##3##4.##5.##6\xint:%
{%
\calx_lhyP\calx_lhyU{##2#1##4.##5.##6}%
      {##2#2##4.##5.##6}\calx_lhyKt##1.%
}%
\expandafter\def\csname calx_lhyEF#1\endcsname ##1.##2##3##4.##5.%
{%
    \ifnum##4=\xint_c_ \xint_dothis{\csname calx_lhyEF#1a\endcsname}\fi
    \ifnum\numexpr##4+\xint_c_viii*##5>\xint_c_iii
      \xint_dothis{\csname calx_lhyEF#1b\endcsname}\fi
    \xint_orthat{\csname calx_lhyEF#1c\endcsname}##1.##2##3##4.##5.%
}%
\expandafter\def\csname calx_lhyEF#1a\endcsname ##1.##2##3##4.##5.##6\xint:%
{%
\expandafter\space\calx_lhyup%
}%
\expandafter\def\csname calx_lhyEF#1b\endcsname ##1.##2##3##4.##5.##6\xint:%
{%
\calx_lExp{\CALX_fsubx{\calx_lhyln}{##2+##4.##5.##6}}%
}%
\expandafter\def\csname calx_lhyEF#1c\endcsname ##1.##2##3##4.##5.##6\xint:%
{%
\calx_lhyP\calx_lhyX{##2#1##4.##5.##6}%
      {##2#2##4.##5.##6}\calx_lhyEt##1.%
}%
\expandafter\def\csname calx_lhyHF#1\endcsname ##1.##2##3##4.##5.%
{%
    \ifnum##4=\xint_c_
      \xint_dothis{\csname calx_lhyHF#1a\endcsname}\fi
    \ifnum\numexpr##4+\xint_c_viii*##5>\xint_c_iii
      \xint_dothis{\csname calx_lhyHF#1b\endcsname}\fi
    \ifnum\numexpr##4+\xint_c_viii*##5<\xint_c_i
      \xint_dothis{\csname calx_lhyHF#1c\endcsname}\fi
    \xint_orthat{\csname calx_lhyHF#1d\endcsname}##1.##2##3##4.##5.%
}%
\expandafter\def\csname calx_lhyHF#1a\endcsname ##1.##2##3##4.##5.##6\xint:%
```

```
{%
\CALX_nsig{DivisionByZero}{csch(0)}{+}%
}%
\expandafter\def\csname calx_lhyHF#1b\endcsname ##1.##2##3##4.##5.##6\xint:%
{%
#5{\calx_lExp{\CALX_fsubx{\calx_lhyln}%
        {##2+##4.##5.##6}}}%
}%
\expandafter\def\csname calx_lhyHF#1c\endcsname ##1.##2##3##4.##5.##6\xint:%
{%
\calx_lhyP\calx_lhyU{##2#1##4.##5.##6}%
        {##2#2##4.##5.##6}\calx_lhyHt##1.%
}%
\expandafter\def\csname calx_lhyHF#1d\endcsname ##1.##2##3##4.##5.##6\xint:%
{%
\calx_lhyP\calx_lhyX{##2#1##4.##5.##6}%
      {##2#2##4.##5.##6}\calx_lhyHt##1.%
}%
\expandafter\def\csname calx_lhysF#1\endcsname ##1.##2##3##4.##5.%
{%
    \ifnum##4=\xint_c_ \xint_dothis{\csname calx_lhysF#1a\endcsname}\fi
    \ifnum\numexpr##4+\xint_c_viii*##5<\numexpr\xint_c_i-##1/\xint_c_ii\relax
      \xint_dothis{\csname calx_lhysF#1b\endcsname}\fi
    \ifnum\numexpr##4+\xint_c_viii*##5>##1
      \xint_dothis{\csname calx_lhysF#1c\endcsname}\fi
    \xint_orthat{\csname calx_lhysF#1d\endcsname}##1.##2##3##4.##5.%
}%
\expandafter\def\csname calx_lhysF#1a\endcsname ##1.##2##3##4.##5.##6\xint:%
{%
 ##2+0.0.%
}%
\expandafter\def\csname calx_lhysF#1b\endcsname ##1.##2##3##4.##5.##6\xint:%
{%
 ##2#1##4.##5.##6%
}%
\expandafter\def\csname calx_lhysF#1c\endcsname ##1.##2##3##4.##5.##6\xint:%
{%
#4{\CALX_faddx{##2+##4.##5.##6}{##2+##4.##5.##6}}%
}%
\expandafter\def\csname calx_lhysF#1d\endcsname ##1.##2##3##4.##5.##6\xint:%
{%
\calx_lhyS{##2+##4.##5.##6}{##1}#4%
}%
\expandafter\def\csname calx_lhytF#1\endcsname ##1.##2##3##4.##5.%
{%
    \ifnum##4=\xint_c_ \xint_dothis{\csname calx_lhytF#1a\endcsname}\fi
    \ifnum\numexpr##4+\xint_c_viii*##5<\numexpr\xint_c_i-##1/\xint_c_ii\relax
      \xint_dothis{\csname calx_lhytF#1b\endcsname}\fi
    \xint_orthat{\csname calx_lhytF#1c\endcsname}##1.##2##3##4.##5.%
}%
\expandafter\def\csname calx_lhytF#1a\endcsname ##1.##2##3##4.##5.##6\xint:%
{%
```

```
 ##2+0.0.%
}%
\expandafter\def\csname calx_lhytF#1b\endcsname ##1.##2##3##4.##5.##6\xint:%
{%
 ##2#1##4.##5.##6%
}%
\expandafter\def\csname calx_lhytF#1c\endcsname ##1.##2##3##4.##5.##6\xint:%
{%
\expandafter\calx_lhyta\romannumeral0\CALX_fsub{##1}%
      {\calx_lhyup}{##2+##4.##5.##6}\xint:{##2+##4.##5.##6}##1.#4#1%
}%
\expandafter\def\csname calx_lhykF#1\endcsname ##1.##2##3##4.##5.##6\xint:%
{%
    \expandafter\calx_lhyka\romannumeral0\CALX_fsub{##1}%
      {##2+##4.##5.##6}{\calx_lhyup}\xint:##1.#4#1%
}%
\expandafter\def\csname calx_lhyhF#1\endcsname ##1.##2##3##4.##5.%
{%
    \ifnum##4=\xint_c_
      \xint_dothis{\csname calx_lhyhF#1a\endcsname}\fi
    \ifnum\numexpr##4+\xint_c_viii*##5>\numexpr##1/\xint_c_ii\relax
      \xint_dothis{\csname calx_lhyhF#1b\endcsname}\fi
    \ifnum\numexpr##4+\xint_c_viii*##5<\numexpr\xint_c_i-##1/\xint_c_ii\relax
      \xint_dothis{\csname calx_lhyhF#1c\endcsname}\fi
    \xint_orthat{\csname calx_lhyhF#1d\endcsname}##1.##2##3##4.##5.%
}%
\expandafter\def\csname calx_lhyhF#1a\endcsname ##1.##2##3##4.##5.##6\xint:%
{%
\CALX_nsig{DivisionByZero}{acsch(0)}{+}%
}%
\expandafter\def\csname calx_lhyhF#1b\endcsname ##1.##2##3##4.##5.##6\xint:%
{%
\CALX_fdiv{\calx_ldigits}{\calx_lhyup}%
        {##2#1##4.##5.##6}%
}%
\expandafter\def\csname calx_lhyhF#1c\endcsname ##1.##2##3##4.##5.##6\xint:%
{%
#4{\CALX_fdiv{##1}{\CALX_fsub{##1}{\calx_lhytwo}%
        {##2+##4.##5.##6}}{##2+##4.##5.##6}}%
}%
\expandafter\def\csname calx_lhyhF#1d\endcsname ##1.##2##3##4.##5.##6\xint:%
{%
\expandafter\calx_lhyb\romannumeral0\CALX_fdiv{##1}%
      {\calx_lhyup}{##2+##4.##5.##6}\xint:##1.#4%
}%
}%
\calx_lhyT+-\calx_lhyup\calx_lhyLp\calx_lhyNp
\calx_lhyT-+\calx_lhyum\calx_lhyLm\calx_lhyNm
```

atanh, acoth: d = 1 - a, a - 1 (at Q digits, exact near |x| = 1); \calx_lhyta d\xint:{a}Q.<L><sign>, \calx_lhyka d\xint:Q.<L><sign>.

```
\def\calx_lhyta #1#2#3.#4.#5\xint:#6#7.#8#9%
{%
```

```
    \if-#2\xint_dothis
      {\CALX_nsig{InvalidOperation}{atanh(x), |x| > 1}{}}\fi
    \ifnum#3=\xint_c_
      \xint_dothis{\CALX_nsig{DivisionByZero}{atanh(+-1)}{#9}}\fi
    \xint_orthat{\expandafter\calx_lhyq\romannumeral0\CALX_fdiv{#7}%
      {\CALX_faddx{#6}{#6}}{#1#2#3.#4.#5}\xint:#7.#8}%
}%
\def\calx_lhyka #1#2#3.#4.#5\xint:#6.#7#8%
{%
    \if-#2\xint_dothis
      {\CALX_nsig{InvalidOperation}{acoth(x), |x| < 1}{}}\fi
    \ifnum#3=\xint_c_
      \xint_dothis{\CALX_nsig{DivisionByZero}{acoth(+-1)}{#8}}\fi
    \xint_orthat{\expandafter\calx_lhyq\romannumeral0\CALX_fdiv{#6}%
      {\calx_lhytwo}{#1#2#3.#4.#5}\xint:#6.#7}%
}%
```

acosh (x >= 1) and asech (0 < x <= 1): t = x - 1, t = (1 - x)/x.

```
\expandafter\def\csname calx_lhycF+\endcsname #1.#2#3#4.#5.#6\xint:%
{%
    \ifnum\numexpr#4+\xint_c_viii*#5>#1
      \xint_dothis{\calx_lhyLp{\CALX_faddx{#2+#4.#5.#6}{#2+#4.#5.#6}}}\fi
    \xint_orthat{\expandafter\calx_lhyca\romannumeral0\CALX_fsub{#1}%
      {#2+#4.#5.#6}{\calx_lhyup}\xint:#1.}%
}%
\expandafter\def\csname calx_lhycF-\endcsname #1\xint:%
   {\CALX_nsig{InvalidOperation}{acosh(x), x < 1}{}}%
\def\calx_lhyca #1#2#3.#4.#5\xint:#6.%
{%
    \if-#2\xint_dothis{\CALX_nsig{InvalidOperation}{acosh(x), x < 1}{}}\fi
    \ifnum#3=\xint_c_ \xint_dothis{ #1+0.0.}\fi
    \xint_orthat{\calx_lhyLp{\calx_lhyW{#1#2#3.#4.#5}{#6}}}%
}%
\expandafter\def\csname calx_lhyeF+\endcsname #1.#2#3#4.#5.#6\xint:%
{%
    \ifnum#4=\xint_c_
      \xint_dothis{\CALX_nsig{DivisionByZero}{asech(0)}{+}}\fi
    \ifnum\numexpr#4+\xint_c_viii*#5<\numexpr\xint_c_i-#1/\xint_c_ii\relax
      \xint_dothis{\calx_lhyLp{\CALX_fdiv{#1}{\CALX_fsub{#1}{\calx_lhytwo}%
        {#2+#4.#5.#6}}{#2+#4.#5.#6}}}\fi
    \xint_orthat{\expandafter\calx_lhyea\romannumeral0\CALX_fsub{#1}%
      {\calx_lhyup}{#2+#4.#5.#6}\xint:{#2+#4.#5.#6}#1.}%
}%
\expandafter\def\csname calx_lhyeF-\endcsname #1\xint:%
   {\CALX_nsig{InvalidOperation}{asech(x), x < 0}{}}%
\def\calx_lhyea #1#2#3.#4.#5\xint:#6#7.%
{%
    \if-#2\xint_dothis{\CALX_nsig{InvalidOperation}{asech(x), x > 1}{}}\fi
    \ifnum#3=\xint_c_ \xint_dothis{ #1+0.0.}\fi
    \xint_orthat{\expandafter\calx_lhyeb\romannumeral0\CALX_fdiv{#7}%
      {#1#2#3.#4.#5}{#6}\xint:#7.}%
}%
\def\calx_lhyeb #1\xint:#2.{\calx_lhyLp{\calx_lhyW{#1}{#2}}}%
```

P <= 8 (codes <c>w, \calx_lhye adds w). x = X8 10^(E-7), X8 the 8 leading digits (\calx_lhzx: one or two blocks, D <= P + 7); u = x^2 at scale 2e9: U2 (\calx_lhzU, abs. error <= 0.5 unit). Results built as V = |y| 10^-z (V >= 10^9, error a few units: 0.05 ulp is >= 5 units) and written by \calx_lhyv + \calx_lhywb (log P <= 8 writer \calx_llgUw), or as a factor F = 1 +- t (two-part native, \calx_lhynb) and one product/quotient at Q digits.

```
|x| < 1    sinh x(1 + u A), A = 1/6 + u A' (A' at 2e11);
           cosh 1 + u/2 + u^2 B' (B' at 4e10); csch (1/x)(1 - (S-1)/S)
|x| < 0.5  tanh x(1 - Delta), Delta = uK/C, K = 1/3 + u K' (K' at 5e10);
           coth (1/x)(1 + Delta/(1 - Delta)); sech 1 - Cm/(1 + Cm)
|x| < 1000 otherwise: one call of the P <= 8 exp kernel (kind xw,
           marker \calx_lxwMk: e^a native from Z' = H' 10^6 + L'', relative
           error < 2.2e-10), then Q-digit operations on E = e^a, E^2
asinh, atanh |x| < 0.1: x(1 - u Ka), x(1 + u Kc) (series to u^4)
```

Other inputs, and acosh, acoth, asech, acsch: the general code.

```
\def\calx_lhzx #1.1#2!#3\relax
   {\ifx\relax#3\relax#2*\CALX_pten{8-#1}\else\calx_lhzy#1.#2.#3\fi}%
\def\calx_lhzy #1.#2.1#3!{#3*\CALX_pten{16-#1}+#2/\CALX_pten{#1-8}}%
\def\calx_lhzU #1.#2.%
   {\ifnum#2>-3 #1*#1/(5*\CALX_pten{4-2*#2})\else
    #1*#1/500000000/\CALX_pten{-2*#2-4}\fi}%
\def\calx_lhzAp #1{(1666666667+#1*(39682540+#1*(551146+#1*(5010%
   +#1*32/2000000000)/2000000000)/2000000000)/2000000000)}%
\def\calx_lhzBp #1{(1666666667+#1*(55555556+#1*(992063+#1*(11023%
   +#1*84/2000000000)/2000000000)/2000000000)/2000000000)}%
\def\calx_lhzKp #1{(1666666667+#1*(59523810+#1*(1102293+#1*(12526%
   +#1*96/2000000000)/2000000000)/2000000000)/2000000000)}%
\def\calx_lhzKb #1{(1500000000-#1*(892857143-#1*607638889/2000000000)%
   /2000000000)}%
\def\calx_lhzKc #1{(2000000000+#1*(1428571429+#1*1111111111/2000000000)%
   /2000000000)}%
```

\calx_lhyv<final>R.H.: H 10^8 + R normalized (-10^8 < R < 2.09e9) -> <final>H'.1<8 digits>. ; \calx_lhywb: the writer (stream: sign z.), \calx_lhynb: the native (stream: z.).

```
\def\calx_lhyv #1#2.#3.%
   {\expandafter\calx_lhyva\the\numexpr(#2+50000000)/100000000-1.#3.#2.#1}%
\def\calx_lhyva #1.#2.#3.#4%
   {\expandafter#4\the\numexpr#2+#1\expandafter.%
    \the\numexpr100000000+#3-100000000*#1.}%
\def\calx_lhywb #1.1#2.{\calx_llgUw#1!!#2!!}%
\def\calx_lhynb #1.1#2.#3.{\CALX_ffromraw{#1#2[#3]}}%
```

series: \calx_lhzS X8.E.<sign> (sinh), \calx_lhzC U2. (cosh), \calx_lhzE U2. (sech), \calx_lhzT U2.<x>.Q.<L> (tanh, coth: <L> = \calx_lhzTt, \calx_lhzKt), \calx_lhzH U2.<x>.Q. (csch), \calx_lhzs, \calx_lhzt (asinh, atanh: X8.E.<sign>)

```
\def\calx_lhzS #1.#2.#3%
{%
    \expandafter\calx_lhzSa\the\numexpr\calx_lhzU#1.#2.\relax.#1.#2.#3%
}%
\def\calx_lhzSa #1.#2.#3.#4%
{%
    \expandafter\calx_lhzSb\the\numexpr#1*5/6%
      +#1*(#1*\calx_lhzAp{#1}/2000000000)/2000000000/20%
    \expandafter.\the\numexpr(#2-500000)/1000000\expandafter.%
```

```
    \the\numexpr#3-9.#2.#4%
}%
\def\calx_lhzSb #1.#2.#3.#4.#5%
{%
    \expandafter\calx_lhyv\expandafter\calx_lhywb
    \the\numexpr100*(#4-1000000*#2)+#4*#1/100000000.#2.#5#3.%
}%
\def\calx_lhzC #1.%
{%
    \expandafter\calx_lhyv\expandafter\calx_lhywb
    \the\numexpr#1/4+#1*(#1*\calx_lhzBp{#1}/2000000000)/2000000000/40.10.+-9.%
}%
\def\calx_lhzCm #1{#1*5/2+#1*(#1*\calx_lhzBp{#1}/2000000000)/2000000000/4}%
\def\calx_lhzE #1.%
{%
    \expandafter\calx_lhzEa\the\numexpr\calx_lhzCm{#1}.%
}%
\def\calx_lhzEa #1.%
{%
    \expandafter\calx_lhyv\expandafter\calx_lhywb
    \the\numexpr2000000000-#1*1000000000/(1000000000+#1/10).80.+-10.%
}%
\def\calx_lhzT #1.%
{%
    \expandafter\calx_lhzTa\the\numexpr\calx_lhzCm{#1}\expandafter.%
    \the\numexpr#1*5/3+#1*(#1*\calx_lhzKp{#1}/2000000000)/2000000000/5.%
}%
\def\calx_lhzTa #1.#2.%
{%
    \expandafter\calx_lhzTb\the\numexpr#2-#2*#1/(1000000000+#1/10)/10.%
}%
\def\calx_lhzTb #1.#2#3.#4{#4#1.{#2}#3.}%
\def\calx_lhzTt #1.#2#3.%
{%
    \CALX_fnround{\calx_ldigits}{\CALX_fmul{#3}{#2}%
      {\expandafter\calx_lhyv\expandafter\calx_lhynb
       \the\numexpr1000000000-#1.90.-10.}}%
}%
\def\calx_lhzKt #1.#2#3.%
{%
    \CALX_fnround{\calx_ldigits}{\CALX_fmul{#3}{\CALX_fdiv{#3}{\calx_lhyup}{#2}}%
      {\expandafter\calx_lhyv\expandafter\calx_lhynb\the\numexpr
       #1*1000000000/(1000000000-#1/10).100.-10.}}%
}%
\def\calx_lhzH #1.%
{%
    \expandafter\calx_lhzHa\the\numexpr#1*5/6%
      +#1*(#1*\calx_lhzAp{#1}/2000000000)/2000000000/20.%
}%
\def\calx_lhzHa #1.#2#3.%
{%
    \CALX_fnround{\calx_ldigits}{\CALX_fmul{#3}{\CALX_fdiv{#3}{\calx_lhyup}{#2}}%
```

```
      {\expandafter\calx_lhyv\expandafter\calx_lhynb\the\numexpr
       2000000000-#1*1000000000/(1000000000+#1/10).80.-10.}}%
}%
\def\calx_lhzs #1.#2.#3%
{%
    \expandafter\calx_lhzsa\the\numexpr\calx_lhzU#1.#2.\relax.#1.#2.#3%
}%
\def\calx_lhzsa #1.#2.#3.#4%
{%
    \expandafter\calx_lhzsb\the\numexpr#1*5/6%
      -#1*(#1*\calx_lhzKb{#1}/2000000000)/2000000000/2%
    \expandafter.\the\numexpr(#2-50000)/100000\expandafter.%
    \the\numexpr#3-10.#2.#4%
}%
\def\calx_lhzsb #1.#2.#3.#4.#5%
{%
    \expandafter\calx_lhyv\expandafter\calx_lhywb
    \the\numexpr1000*(#4-100000*#2)+1000000000-#4*#1/10000000\expandafter.%
    \the\numexpr#2-10.#5#3.%
}%
\def\calx_lhzt #1.#2.#3%
{%
    \expandafter\calx_lhzta\the\numexpr\calx_lhzU#1.#2.\relax.#1.#2.#3%
}%
\def\calx_lhzta #1.#2.#3.#4%
{%
    \expandafter\calx_lhzSb\the\numexpr#1*5/3%
      +#1*(#1*\calx_lhzKc{#1}/2000000000)/2000000000%
    \expandafter.\the\numexpr(#2-500000)/1000000\expandafter.%
    \the\numexpr#3-9.#2.#4%
}%
```

exp kernel P <= 8, kind xw: \calx_lxwMk H'.L''.k.<cont> -> <cont> E\xint: (E = e^a native); continuations <cont>Q.<data>: \calx_lhzXS (sinh, cosh: <fsubx|faddx><sign>), \calx_lhzXT (tanh, coth, sech, csch: <final><final for E >= 10^8><sign 1 or 2>); E >= 10^8: E^-1 negligible.

```
\def\calx_lxwhG 1#1!1#2!1#3!1#4!#5\xint:#6%
   {\calx_lexGn#1.#2.#3.#4.#6C\calx_lxwMk}%
\def\calx_lxwMk #1.#2.#3.%
   {\expandafter\calx_lxwMa\the\numexpr#3-12\expandafter.%
    \the\numexpr1000000+#2.#1.}%
\def\calx_lxwMa #1.1#2.#3.#4%
   {\expandafter#4\romannumeral0\CALX_ffromraw{#3#2[#1]}\xint:}%
\begingroup\catcode`F=12
\gdef\calx_lhzXS F#1#2.#3.%
{%
    \ifnum\numexpr#2+\xint_c_viii*#3>\xint_c_viii
      \xint_dothis\calx_lhzXS_large\fi
    \xint_orthat\calx_lhzXS_regular
    F#1#2.#3.%
}%
\gdef\calx_lhzXS_large F#1#2.#3.#4\xint:#5.#6#7%
   {#7{\CALX_fnround{\calx_ldigits}{\CALX_fhalf{F#1#2.#3.#4}}}}%
\gdef\calx_lhzXS_regular F#1#2.#3.#4\xint:#5.#6#7%
```

```
    {#7{\CALX_fnround{\calx_ldigits}{\CALX_fhalf{#6{F#1#2.#3.#4}%
       {\CALX_fdiv{#5}{\calx_lhyup}{F#1#2.#3.#4}}}}}}%
\gdef\calx_lhzXT F#1#2.#3.%
{%
    \ifnum\numexpr#2+\xint_c_viii*#3>\xint_c_viii
      \xint_dothis\calx_lhzXT_large\fi
    \xint_orthat\calx_lhzXT_regular
    F#1#2.#3.%
}%
\gdef\calx_lhzXT_large F#1#2.#3.#4\xint:#5.#6#7#8%
   {#7{F#1#2.#3.#4}#8}%
\gdef\calx_lhzXT_regular F#1#2.#3.#4\xint:#5.#6#7#8%
   {\expandafter#6\romannumeral0\CALX_fmul{#5}{F#1#2.#3.#4}%
      {F#1#2.#3.#4}\xint:{F#1#2.#3.#4}#5.#8}%
\endgroup
```

finals: normal <E^2>\xint:{E}Q.<c> (c = sign 1 or 2), E >= 10^8: {E}<c>

```
\def\calx_lhzXTa #1\xint:#2#3.#4%
   {\CALX_fnround{\calx_ldigits}{\CALX_fdiv{#3}{\CALX_fmulx{#4}%
      {\CALX_fsubx{#1}{\calx_lhyup}}}}{\CALX_faddx{#1}{\calx_lhyup}}}}%
\def\calx_lhzXKa #1\xint:#2#3.#4%
   {\CALX_fnround{\calx_ldigits}{\CALX_fdiv{#3}{\CALX_fmulx{#4}%
      {\CALX_faddx{#1}{\calx_lhyup}}}}{\CALX_fsubx{#1}{\calx_lhyup}}}}%
\def\calx_lhzXTb #1{\expandafter\space}%
\def\calx_lhzXEa #1\xint:#2#3.#4%
   {\CALX_fnround{\calx_ldigits}{\CALX_fdiv{#3}{\CALX_fmulx{#4}{#2}}}%
      {\CALX_faddx{#1}{\calx_lhyup}}}}%
\def\calx_lhzXHa #1\xint:#2#3.#4%
   {\CALX_fnround{\calx_ldigits}{\CALX_fdiv{#3}{\CALX_fmulx{#4}{#2}}}%
      {\CALX_fsubx{#1}{\calx_lhyup}}}}%
\def\calx_lhzXEb #1#2{\CALX_fnround{\calx_ldigits}{\CALX_fdiv{12}{#2}{#1}}}%
\def\calx_lhzu #1#2.#3.{\expandafter#1\the\numexpr\calx_lhzU#2.#3.\relax.}%
```

\calx_lhzW<sign><opposite><sign 1><sign wrapper><sign 2>: the P <= 8 F macros; v = D + 8e = E + 1.

```
\edef\calx_lhymtwo{\romannumeral0\CALX_ffromraw{-2[0]}}%
\def\calx_lhzW #1#2#3#4#5%
{%
\expandafter\def\csname calx_lhySwF#1\endcsname ##1.##2##3##4.##5.%
{%
    \ifnum##4=\xint_c_
      \xint_dothis{\csname calx_lhySwF#1a\endcsname ##1.##2##3##4.##5.}\fi
    \ifnum\numexpr##4+\xint_c_viii*##5<-\xint_c_iv
      \xint_dothis{\csname calx_lhySwF#1b\endcsname ##1.##2##3##4.##5.}\fi
    \ifnum\numexpr##4+\xint_c_viii*##5<\xint_c_i
      \xint_dothis{\csname calx_lhySwF#1c\endcsname ##1.##2##3##4.##5.}\fi
    \ifnum\numexpr##4+\xint_c_viii*##5<\xint_c_iv
      \xint_dothis{\csname calx_lhySwF#1d\endcsname ##1.##2##3##4.##5.}\fi
    \xint_orthat{\csname calx_lhySwF#1e\endcsname ##1.##2##3##4.##5.}%
}%
\expandafter\def\csname calx_lhySwF#1a\endcsname ##1.##2##3##4.##5.##6\xint:%
{%
     ##2+0.0.%
}%
\expandafter\def\csname calx_lhySwF#1b\endcsname ##1.##2##3##4.##5.##6\xint:%
```

```
{%
     ##2#1##4.##5.##6%
}%
\expandafter\def\csname calx_lhySwF#1c\endcsname ##1.##2##3##4.##5.##6\xint:%
{%
    \expandafter\calx_lhzS
        \the\numexpr\calx_lhzx##4.##6\relax\expandafter.%
        \the\numexpr##4-\xint_c_i+\xint_c_viii*##5.#1%
}%
\expandafter\def\csname calx_lhySwF#1d\endcsname ##1.##2##3##4.##5.##6\xint:%
{%
    \calx_lexb xw;+##4.##5.##6\xint:\calx_lhzXS##1.%
        \CALX_fsubx#4%
}%
\expandafter\def\csname calx_lhySwF#1e\endcsname ##1.##2##3##4.##5.##6\xint:%
{%
    \csname calx_lhySF#1\endcsname##1.##2##3##4.##5.##6\xint:%
}%
%
\expandafter\def\csname calx_lhyCwF#1\endcsname ##1.##2##3##4.##5.%
{%
    \ifnum##4=\xint_c_
      \xint_dothis{\csname calx_lhyCwF#1a\endcsname ##1.##2##3##4.##5.}\fi
    \ifnum\numexpr##4+\xint_c_viii*##5<-\xint_c_iv
      \xint_dothis{\csname calx_lhyCwF#1b\endcsname ##1.##2##3##4.##5.}\fi
    \ifnum\numexpr##4+\xint_c_viii*##5<\xint_c_i
      \xint_dothis{\csname calx_lhyCwF#1c\endcsname ##1.##2##3##4.##5.}\fi
    \ifnum\numexpr##4+\xint_c_viii*##5<\xint_c_iv
      \xint_dothis{\csname calx_lhyCwF#1d\endcsname ##1.##2##3##4.##5.}\fi
    \xint_orthat{\csname calx_lhyCwF#1e\endcsname ##1.##2##3##4.##5.}%
}%
\expandafter\def\csname calx_lhyCwF#1a\endcsname ##1.##2##3##4.##5.##6\xint:%
{%
    \expandafter\space\calx_lhyup%
}%
\expandafter\def\csname calx_lhyCwF#1b\endcsname ##1.##2##3##4.##5.##6\xint:%
{%
    \expandafter\space\calx_lhyup%
}%
\expandafter\def\csname calx_lhyCwF#1c\endcsname ##1.##2##3##4.##5.##6\xint:%
{%
    \expandafter\calx_lhzu\expandafter\calx_lhzC
        \the\numexpr\calx_lhzx##4.##6\relax\expandafter.%
        \the\numexpr##4-\xint_c_i+\xint_c_viii*##5.%
}%
\expandafter\def\csname calx_lhyCwF#1d\endcsname ##1.##2##3##4.##5.##6\xint:%
{%
    \calx_lexb xw;+##4.##5.##6\xint:\calx_lhzXS##1.%
        \CALX_faddx\calx_lhyNp%
}%
\expandafter\def\csname calx_lhyCwF#1e\endcsname ##1.##2##3##4.##5.##6\xint:%
{%
```

```
    \csname calx_lhyCF#1\endcsname##1.##2##3##4.##5.##6\xint:%
}%
%
\expandafter\def\csname calx_lhyTwF#1\endcsname ##1.##2##3##4.##5.%
{%
    \ifnum##4=\xint_c_ \xint_dothis{\csname calx_lhyTwF#1a\endcsname}\fi
    \ifnum\numexpr##4+\xint_c_viii*##5<-\xint_c_iv
      \xint_dothis{\csname calx_lhyTwF#1b\endcsname}\fi
    \xint_orthat{\csname calx_lhyTwF#1r\endcsname}##1.##2##3##4.##5.%
}%
\expandafter\def\csname calx_lhyTwF#1a\endcsname ##1.##2##3##4.##5.##6\xint:%
{%
 ##2+0.0.%
}%
\expandafter\def\csname calx_lhyTwF#1b\endcsname ##1.##2##3##4.##5.##6\xint:%
{%
 ##2#1##4.##5.##6%
}%
\expandafter\def\csname calx_lhyTwF#1r\endcsname ##1.##2##3##4.##5.##6\xint:%
{%
    \ifnum\numexpr(##4+\xint_c_viii*##5)*100000000%
      +\calx_lhzx##4.##6\relax<50000000
      \xint_dothis{\expandafter\calx_lhzu\expandafter\calx_lhzT
        \the\numexpr\calx_lhzx##4.##6\relax\expandafter.%
        \the\numexpr##4-\xint_c_i+\xint_c_viii*##5.%
        {##2#1##4.##5.##6}##1.\calx_lhzTt}\fi
    \ifnum\numexpr##4+\xint_c_viii*##5<\xint_c_iv
      \xint_dothis{\calx_lexb xw;+##4.##5.##6\xint:\calx_lhzXT##1.%
        \calx_lhzXTa\calx_lhzXTb#3}\fi
    \xint_orthat{\csname calx_lhyTF#1\endcsname##1.##2##3##4.##5.##6\xint:}%
}%
\expandafter\def\csname calx_lhyKwF#1\endcsname ##1.##2##3##4.##5.%
{%
    \ifnum##4=\xint_c_
      \xint_dothis{\csname calx_lhyKwF#1a\endcsname}\fi
    \ifnum\numexpr##4+\xint_c_viii*##5<-\xint_c_iv
      \xint_dothis{\csname calx_lhyKwF#1b\endcsname}\fi
    \xint_orthat{\csname calx_lhyKwF#1r\endcsname}##1.##2##3##4.##5.%
}%
\expandafter\def\csname calx_lhyKwF#1a\endcsname ##1.##2##3##4.##5.##6\xint:%
{%
\CALX_nsig{DivisionByZero}{coth(0)}{+}%
}%
\expandafter\def\csname calx_lhyKwF#1b\endcsname ##1.##2##3##4.##5.##6\xint:%
{%
\CALX_fdiv{\calx_ldigits}{\calx_lhyup}{##2#1##4.##5.##6}%
}%
\expandafter\def\csname calx_lhyKwF#1r\endcsname ##1.##2##3##4.##5.##6\xint:%
{%
    \ifnum\numexpr(##4+\xint_c_viii*##5)*100000000%
      +\calx_lhzx##4.##6\relax<50000000
      \xint_dothis{\expandafter\calx_lhzu\expandafter\calx_lhzT
```

```
3005          \the\numexpr\calx_lhzx##4.##6\relax\expandafter.%
3006          \the\numexpr##4-\xint_c_i+\xint_c_viii*##5.%
3007          {##2#1##4.##5.##6}##1.\calx_lhzKt}\fi
3008      \ifnum\numexpr##4+\xint_c_viii*##5<\xint_c_iv
3009        \xint_dothis{\calx_lexb xw;+##4.##5.##6\xint:\calx_lhzXT##1.%
3010          \calx_lhzXKa\calx_lhzXTb#3}\fi
3011      \xint_orthat{\csname calx_lhyKF#1\endcsname##1.##2##3##4.##5.##6\xint:}%
3012  }%
3013  \expandafter\def\csname calx_lhyEwF#1\endcsname ##1.##2##3##4.##5.%
3014  {%
3015      \ifnum##4=\xint_c_ \xint_dothis{\csname calx_lhyEwF#1a\endcsname}\fi
3016      \ifnum\numexpr##4+\xint_c_viii*##5<-\xint_c_iv
3017        \xint_dothis{\csname calx_lhyEwF#1b\endcsname}\fi
3018      \xint_orthat{\csname calx_lhyEwF#1r\endcsname}##1.##2##3##4.##5.%
3019  }%
3020  \expandafter\def\csname calx_lhyEwF#1a\endcsname ##1.##2##3##4.##5.##6\xint:%
3021  {%
3022  \expandafter\space\calx_lhyup%
3023  }%
3024  \expandafter\def\csname calx_lhyEwF#1b\endcsname ##1.##2##3##4.##5.##6\xint:%
3025  {%
3026  \expandafter\space\calx_lhyup%
3027  }%
3028  \expandafter\def\csname calx_lhyEwF#1r\endcsname ##1.##2##3##4.##5.##6\xint:%
3029  {%
3030      \ifnum\numexpr(##4+\xint_c_viii*##5)*100000000%
3031        +\calx_lhzx##4.##6\relax<50000000
3032        \xint_dothis{\expandafter\calx_lhzu\expandafter\calx_lhzE
3033          \the\numexpr\calx_lhzx##4.##6\relax\expandafter.%
3034          \the\numexpr##4-\xint_c_i+\xint_c_viii*##5.}\fi
3035      \ifnum\numexpr##4+\xint_c_viii*##5<\xint_c_iv
3036        \xint_dothis{\calx_lexb xw;+##4.##5.##6\xint:\calx_lhzXT##1.%
3037          \calx_lhzXEa\calx_lhzXEb\calx_lhytwo}\fi
3038      \xint_orthat{\csname calx_lhyEF#1\endcsname##1.##2##3##4.##5.##6\xint:}%
3039  }%
3040  \expandafter\def\csname calx_lhyHwF#1\endcsname ##1.##2##3##4.##5.%
3041  {%
3042      \ifnum##4=\xint_c_
3043        \xint_dothis{\csname calx_lhyHwF#1a\endcsname}\fi
3044      \ifnum\numexpr##4+\xint_c_viii*##5<-\xint_c_iv
3045        \xint_dothis{\csname calx_lhyHwF#1b\endcsname}\fi
3046      \xint_orthat{\csname calx_lhyHwF#1r\endcsname}##1.##2##3##4.##5.%
3047  }%
3048  \expandafter\def\csname calx_lhyHwF#1a\endcsname ##1.##2##3##4.##5.##6\xint:%
3049  {%
3050  \CALX_nsig{DivisionByZero}{csch(0)}{+}%
3051  }%
3052  \expandafter\def\csname calx_lhyHwF#1b\endcsname ##1.##2##3##4.##5.##6\xint:%
3053  {%
3054  \CALX_fdiv{\calx_ldigits}{\calx_lhyup}{##2#1##4.##5.##6}%
3055  }%
3056  \expandafter\def\csname calx_lhyHwF#1r\endcsname ##1.##2##3##4.##5.##6\xint:%
```

```
{%
    \ifnum\numexpr##4+\xint_c_viii*##5<\xint_c_i
      \xint_dothis{\expandafter\calx_lhzu\expandafter\calx_lhzH
        \the\numexpr\calx_lhzx##4.##6\relax\expandafter.%
        \the\numexpr##4-\xint_c_i+\xint_c_viii*##5.%
        {##2#1##4.##5.##6}##1.}\fi
    \ifnum\numexpr##4+\xint_c_viii*##5<\xint_c_iv
      \xint_dothis{\calx_lexb xw;+##4.##5.##6\xint:\calx_lhzXT##1.%
        \calx_lhzXHa\calx_lhzXEb#5}\fi
    \xint_orthat{\csname calx_lhyHF#1\endcsname##1.##2##3##4.##5.##6\xint:}%
}%
\expandafter\def\csname calx_lhyswF#1\endcsname ##1.##2##3##4.##5.%
{%
    \ifnum##4=\xint_c_
      \xint_dothis{\csname calx_lhyswF#1a\endcsname ##1.##2##3##4.##5.}\fi
    \ifnum\numexpr##4+\xint_c_viii*##5<-\xint_c_iv
      \xint_dothis{\csname calx_lhyswF#1b\endcsname ##1.##2##3##4.##5.}\fi
    \ifnum\numexpr##4+\xint_c_viii*##5<\xint_c_
      \xint_dothis{\csname calx_lhyswF#1c\endcsname ##1.##2##3##4.##5.}\fi
    \xint_orthat{\csname calx_lhyswF#1d\endcsname ##1.##2##3##4.##5.}%
}%
\expandafter\def\csname calx_lhyswF#1a\endcsname ##1.##2##3##4.##5.##6\xint:%
{%
     ##2+0.0.%
}%
\expandafter\def\csname calx_lhyswF#1b\endcsname ##1.##2##3##4.##5.##6\xint:%
{%
     ##2#1##4.##5.##6%
}%
\expandafter\def\csname calx_lhyswF#1c\endcsname ##1.##2##3##4.##5.##6\xint:%
{%
    \expandafter\calx_lhzs
        \the\numexpr\calx_lhzx##4.##6\relax\expandafter.%
        \the\numexpr##4-\xint_c_i+\xint_c_viii*##5.#1%
}%
\expandafter\def\csname calx_lhyswF#1d\endcsname ##1.##2##3##4.##5.##6\xint:%
{%
    \csname calx_lhysF#1\endcsname##1.##2##3##4.##5.##6\xint:%
}%
%
\expandafter\def\csname calx_lhytwF#1\endcsname ##1.##2##3##4.##5.%
{%
    \ifnum##4=\xint_c_
      \xint_dothis{\csname calx_lhytwF#1a\endcsname ##1.##2##3##4.##5.}\fi
    \ifnum\numexpr##4+\xint_c_viii*##5<-\xint_c_iv
      \xint_dothis{\csname calx_lhytwF#1b\endcsname ##1.##2##3##4.##5.}\fi
    \ifnum\numexpr##4+\xint_c_viii*##5<\xint_c_
      \xint_dothis{\csname calx_lhytwF#1c\endcsname ##1.##2##3##4.##5.}\fi
    \xint_orthat{\csname calx_lhytwF#1d\endcsname ##1.##2##3##4.##5.}%
}%
\expandafter\def\csname calx_lhytwF#1a\endcsname ##1.##2##3##4.##5.##6\xint:%
{%
```

```
      ##2+0.0.%
}%
\expandafter\def\csname calx_lhytwF#1b\endcsname ##1.##2##3##4.##5.##6\xint:%
{%
      ##2#1##4.##5.##6%
}%
\expandafter\def\csname calx_lhytwF#1c\endcsname ##1.##2##3##4.##5.##6\xint:%
{%
    \expandafter\calx_lhzt
        \the\numexpr\calx_lhzx##4.##6\relax\expandafter.%
        \the\numexpr##4-\xint_c_i+\xint_c_viii*##5.#1%
}%
\expandafter\def\csname calx_lhytwF#1d\endcsname ##1.##2##3##4.##5.##6\xint:%
{%
    \csname calx_lhytF#1\endcsname##1.##2##3##4.##5.##6\xint:%
}%
%
}%
\calx_lhzW+-\calx_lhyup\calx_lhyNp\calx_lhytwo
\calx_lhzW-+\calx_lhyum\calx_lhyNm\calx_lhymtwo
\def\calx_lhzL #1%
{%
    \expandafter\let\csname calx_lhy#1wF+\expandafter\endcsname
      \csname calx_lhy#1F+\endcsname
    \expandafter\let\csname calx_lhy#1wF-\expandafter\endcsname
      \csname calx_lhy#1F-\endcsname
}%
\calx_lhzL c\calx_lhzL k\calx_lhzL e\calx_lhzL h%
```

Public macros and expression functions (\calx_lhymk <Name><code><name>).

```
\def\calx_lhymk #1#2#3%
{%
    \expandafter\def\csname calx_l#1\endcsname##1%
      {\romannumeral0\expandafter\calx_lhyd\expandafter#2%
       \romannumeral0\CALX_fround{\calx_ldigits}{##1}}%
    \expandafter\edef\csname calxRaw#1\endcsname##1%
      {\noexpand\romannumeral0\noexpand\CALX_ftoraw
       {\expandafter\noexpand\csname calxFp#1\endcsname{##1}}}%
    \expandafter\edef\csname XINT_flexpr_func_#3\endcsname##1##2##3%
      {\noexpand\expandafter##1\noexpand\expandafter##2%
       \noexpand\expandafter{\noexpand\romannumeral`&&@%
       \noexpand\XINT:NEhook:f:one:from:one{\noexpand\romannumeral`&&@%
       \expandafter\noexpand\csname calxFp#1\endcsname##3}}}%
    \expandafter\edef\csname XINT_expr_func_#3\endcsname##1##2##3%
      {\noexpand\expandafter##1\noexpand\expandafter##2%
       \noexpand\expandafter{\noexpand\romannumeral`&&@%
       \noexpand\XINT:NEhook:f:one:from:one{\noexpand\romannumeral`&&@%
       \expandafter\noexpand\csname calxRaw#1\endcsname##3}}}%
}%
\calx_lhymk{Sinh}S{sinh}\calx_lhymk{Cosh}C{cosh}\calx_lhymk{Tanh}T{tanh}%
\calx_lhymk{Coth}K{coth}\calx_lhymk{Sech}E{sech}\calx_lhymk{Csch}H{csch}%
\calx_lhymk{Asinh}s{asinh}\calx_lhymk{Acosh}c{acosh}%
\calx_lhymk{Atanh}t{atanh}\calx_lhymk{Acoth}k{acoth}%
```

```
\calx_lhymk{Asech}e{asech}\calx_lhymk{Acsch}h{acsch}%
```

33.18. Aliases

```
\let\XINT_flexpr_func_ln\XINT_flexpr_func_log
\let\XINT_expr_func_ln\XINT_expr_func_log
\let\XINT_flexpr_func_sh\XINT_flexpr_func_sinh
\let\XINT_flexpr_func_ch\XINT_flexpr_func_cosh
\let\XINT_flexpr_func_th\XINT_flexpr_func_tanh
\let\XINT_flexpr_func_argsh\XINT_flexpr_func_asinh
\let\XINT_flexpr_func_argch\XINT_flexpr_func_acosh
\let\XINT_flexpr_func_argth\XINT_flexpr_func_atanh
\let\XINT_flexpr_func_argcoth\XINT_flexpr_func_acoth
\let\XINT_flexpr_func_argsech\XINT_flexpr_func_asech
\let\XINT_flexpr_func_argcsch\XINT_flexpr_func_acsch
\xintFor #1 in {sh, ch, th, argsh, argch, argth, argcoth, argsech, argcsch}\do{%
    \expandafter\let\csname XINT_expr_func_#1\expandafter\endcsname
                    \csname XINT_expr_func_#1\endcsname
}%
```

33.19. Reconstruction at the scientific precision

The working precision is the next multiple of four. Only the schedules for that precision (and the extra logarithm precisions for powers) are constructed. The output wrappers are selected here, not during evaluation. All installed state follows \XINT_global. A request at the installed output precision is a no-op, except when local state must be made global.

```
\def\calx_ldgoc{\calx_tdgoP\calx_ldigits.}%
\def\calx_lclear #1#2{%
    \XINT_global\expandafter\let\csname calx_l#1#2\endcsname\undefined
}%
\def\calx_lZulloop #1.#2#3{%
    \ifnum#1>16 #2{#1}\fi
    \ifnum#3>0 \expandafter\calx_lZulloop\the\numexpr#1+4\expandafter.\expandafter
        #2\expandafter{\the\numexpr#3-1\expandafter}\fi
}%
\def\calx_lclearZul#1{\calx_lclear{Zul}{#1}}%
```

The largest internal logarithm precision is P+12. A generic layout g uses g+2 pairs, including the first pair (integer part and first eight decimal places). The fixed P <= 16 path uses three pairs. The full constants remain on disk. Only the prefix used by a layout is retained; the original arithmetic and carry chains are unchanged.

```
\def\calx_lpairdef #1#2%
{%
    \XINT_global\expandafter\edef\csname#1\endcsname
      {\expandafter\calx_lpairtake\the\calx_lpaircount.#2\calx_lLQ}%
}%
\def\calx_lpairtake #1.#2!%
{%
    #2!%
    \ifnum#1>1
      \xint_dothis{\expandafter\calx_lpairtake\the\numexpr#1-1.}\fi
    \xint_orthat\calx_lpairend
}%
```

```
\def\calx_lpairend #1\calx_lLQ{}%
\def\calx_lreadcoeff{%
    \ifnum\calx_lpaircount=\numexpr(\calx_ldigits+7)/8+2\relax\else
        \XINT_global\mathchardef\calx_lpaircount=\numexpr(\calx_ldigits+7)/8+2\relax
        \let\calx_lreadnewpairs\empty
    \fi
    \edef\calx_lrestore{\XINTrestorecatcodes\catcode63=\the\catcode63\relax}%
    \XINTsetcatcodes
    \input calxlogdata.tex\relax
    \calx_lrestore\let\calx_lrestore\undefined
    \let\calx_lreadnewpairs\undefined
    \let\calx_tcoeffonly\empty
    \calx_treadData
    \let\calx_tcoeffonly\undefined
}%
\def\calx_lfreecoeff #1.{%
    \expandafter\let\csname calx_lcdl#1\endcsname\undefined
    \ifnum#1<30 \expandafter\calx_lfreecoeff\the\numexpr#1+1\expandafter.\fi
}%
```

Generic residual schedules: g = (P-5)/8 through (P+7)/8, restricted to g >= 4. The two smaller layouts have their own fixed kernels.

```
\def\calx_lZlgloop #1{%
    \expandafter\calx_lZlgstep\the\numexpr
      \ifnum\calx_ldigits<36 4\else(\calx_ldigits-5)/8\fi\expandafter.%
      \the\numexpr(\calx_ldigits+7)/8.#1}%
\def\calx_lZlgstep #1.#2.#3{%
    \ifnum#1>#2\else
        #3{#1}%
        \xint_afterfi{\expandafter\calx_lZlgstep\the\numexpr#1+1.#2.#3}%
    \fi
}%
\def\calx_lclearZlg #1{%
    \ifnum#1<\numexpr(\calx_lnewdigits-5)/8\relax
        \calx_lclear{Zlg}{#1}%
    \else\ifnum#1>\numexpr(\calx_lnewdigits+7)/8\relax
        \calx_lclear{Zlg}{#1}%
    \fi\fi
}%
\def\calx_luseZlg #1{%
    \ifcsname calx_lZlg#1\endcsname\else\calx_lbuildZlg{#1}\fi
}%
```

Consecutive buckets with identical keys and multipliers share their replacement text via \let. A is also needed by the unrounded exponential and expm1 at low precision; C is only used at P <= 8.

```
\def\calx_lbuildfamily #1{%
    \def\calx_lexfamily{#1}%
    \let\calx_lexprev\undefined
    \expandafter\let\expandafter\calx_lexMa\csname calx_lexMa#1\endcsname
    \expandafter\calx_lexLd\calx_lexAlist
    \let\calx_lexMa\undefined
    \let\calx_lexprev\undefined \let\calx_lexcurr\undefined
    \let\calx_lexindex\undefined \let\calx_lexfamily\undefined
}%
```

```
\def\calx_lclearfamily #1{%
    \def\calx_lexfamily{#1}%
    \expandafter\calx_lclearex\calx_lexAlist
    \let\calx_lexfamily\undefined
}%
\def\calx_lclearex #1.#2.#3.%
{%
    \ifnum#1<\xint_c_ \expandafter\xint_gobble_i\else
        \calx_lclear{ex\calx_lexfamily}{#1}%
    \fi
    \calx_lclearex
}%
\def\calx_lusefamily #1{%
    \ifcsname calx_lex#10\endcsname\else\calx_lbuildfamily#1\fi
}%
\def\calx_ldropfamily #1{%
    \ifcsname calx_lex#10\endcsname\calx_lclearfamily#1\fi
}%
\def\calx_lbuildfamilies{%
    \ifnum\calx_ldigits>16
        \calx_ldropfamily A\calx_ldropfamily C\calx_lusefamily B%
    \else
        \calx_ldropfamily B\calx_lusefamily A%
        \ifnum\calx_ldigits<9 \calx_lusefamily C\else\calx_ldropfamily C\fi
    \fi
}%
\def\calx_lbuildshort{%
    \calx_lexMt calx_lexu.calx_lLs.100.11.-11.%
    \calx_lexMt calx_lexv.calx_lLt.1.56.-56.%
    \calx_lexMt calx_lexw.calx_lLu.100.51.-51.%
    \calx_lexMt calx_lexz.calx_lLv.1.51.-51.%
    \let\calx_lLtmpb\undefined
}%
\def\calx_lbuildworking{%
    \ifnum\calx_ldigits>8 \ifnum\calx_ldigits<17
        \XINT_global\edef\calx_lexq{\the\numexpr\CALX_pten{16-\calx_ldigits}.%
            \CALX_pten{\calx_ldigits-8}.\ifnum\calx_ldigits=16 1\else0\fi.}%
    \else \XINT_global\let\calx_lexq\undefined\fi
    \else \XINT_global\let\calx_lexq\undefined\fi
    \ifnum\calx_ldigits>16
        \XINT_global\expandafter\let\expandafter\calx_lLGsa
            \csname calx_lLGsa\the\numexpr(\calx_ldigits-5)/8\endcsname
        \XINT_global\let\calx_llga\calx_llga_high
        \XINT_global\let\calx_lexb\calx_lexgb
        \XINT_global\mathchardef\calx_llpBw=\numexpr(\calx_ldigits+3)/8+1\relax
        \XINT_global\def\calx_llpBkind{g}%
    \else
        \XINT_global\let\calx_lexb\calx_lexb_small
        \XINT_global\mathchardef\calx_llpBw=3
        \XINT_global\def\calx_llpBkind{s}%
        \ifnum\calx_ldigits<9 \XINT_global\let\calx_llga\calx_llga_low
        \else \XINT_global\let\calx_llga\calx_llga_mid\fi
```

```
    \fi
    \XINT_global\mathchardef
        \calx_llpP=\ifnum\calx_ldigits<20 20\else\calx_ldigits\fi\relax
    \ifnum\calx_ldigits<9
        \XINT_global\def\calx_lexhkind{h}\XINT_global\def\calx_lhykind{w}%
        \XINT_global\let\calx_lexY\calx_lexY_low
        \XINT_global\mathchardef\calx_lhyQ=12
    \else
        \XINT_global\let\calx_lexhkind\empty \XINT_global\let\calx_lhykind\empty
        \XINT_global\let\calx_lexY\calx_lexY_mid
        \XINT_global\mathchardef\calx_lhyQ=\numexpr\calx_ldigits+4\relax
    \fi
    \ifnum\calx_ldigits>5 \XINT_global\let\calx_lexhround\calx_lexhP
    \else \XINT_global\let\calx_lexhround\calx_lexhQ\fi
    \ifnum\calx_ldigits>32
        \XINT_global\let\calx_lexgVkernel\calx_lexgVb
        \XINT_global\let\calx_lexgW\calx_lexgWk
    \else
        \XINT_global\let\calx_lexgW\calx_lexgWsmall
        \ifnum\calx_ldigits>24 \XINT_global\let\calx_lexgVkernel\calx_lexgVq
        \else \XINT_global\let\calx_lexgVkernel\calx_lexgVp\fi
    \fi
    \ifnum\calx_ldigits>24 \XINT_global\def\calx_lexlayout{b}%
    \else \XINT_global\def\calx_lexlayout{a}\fi
    \ifnum\numexpr\calx_ldigits-8*((\calx_ldigits+3)/8)\relax=0
        \XINT_global\def\calx_lexXzero{v}\XINT_global\def\calx_lexXnonzero{u}%
    \else
        \XINT_global\def\calx_lexXzero{z}\XINT_global\def\calx_lexXnonzero{s}%
    \fi
    \calx_lZlgloop\calx_luseZlg
    \let\calx_lLWfin\calx_lLWfinish
    \expandafter\calx_lZulloop\the\calx_ldigits.\calx_lbuildZul{3}%
    \let\calx_lLWfin\empty
    \edef\calx_lbuildp{\ifnum\calx_ldigits<20 20\else\the\calx_ldigits\fi}%
    \calx_lbuildlp{\calx_lbuildp}%
    \let\calx_lLWco\calx_lexgK
    \calx_lbuildem{\calx_lbuildp}%
    \def\calx_lLWco{\calx_ttcoef{calx_lcdl}}%
    \ifnum\calx_ldigits>32 \calx_lbuildex{\the\calx_ldigits}%
    \else \XINT_global\let\calx_lexS\undefined
          \XINT_global\let\calx_lexT\undefined\fi
    \ifnum\calx_ldigits>16 \calx_lbuildpw{\the\calx_ldigits}%
    \else \XINT_global\let\calx_lpwT\undefined\fi
    \XINT_global\edef\calx_lltk{\romannumeral0\CALX_fnround{\calx_ldigits+12}%
        {\CALX_ffromraw{\calx_lLzinvraw}}}%
    \let\calx_lLWa\undefined \let\calx_lLWb\undefined
    \let\calx_lLWm\undefined \let\calx_lLWp\undefined
    \let\calx_lLWfin\undefined \let\calx_lbuildp\undefined
}%
\def\calx_linstall#1{%
    \ifnum\calx_ldigits=\calx_loutdigits
       \XINT_global\expandafter\let\csname calxFp#1\expandafter\endcsname
```

```
            \csname calx_l#1\endcsname
    \else
       \XINT_global\expandafter\edef\csname calxFp#1\endcsname##1{%
          \noexpand\romannumeral0\noexpand\CALX_fnround\noexpand\calx_loutdigits
          {\expandafter\noexpand\csname calx_l#1\endcsname{##1}}}%
    \fi
}%
```

If a local installation is replaced globally, an undefined table may hide a saved outer meaning. Clear all precision-dependent table keys before reconstruction so that none of those meanings can reappear.

```
\def\calx_lpromote{%
    \calx_lclearall 20.%
    \calx_lclearallZlg 4.%
    \calx_lclearfamily A\calx_lclearfamily B\calx_lclearfamily C%
    \mathchardef\calx_ldigits=0
    \mathchardef\calx_loutdigits=0
    \mathchardef\calx_lpaircount=0
}%
\def\calx_lclearall #1.{%
    \calx_lclearZul{#1}%
    \ifnum#1<112 \expandafter\calx_lclearall\the\numexpr#1+4\expandafter.\fi
}%
\def\calx_lclearallZlg #1.{%
    \calx_lclear{Zlg}{#1}%
    \ifnum#1<13 \expandafter\calx_lclearallZlg\the\numexpr#1+1\expandafter.\fi
}%
\def\calx_lreload{%
    \ifxintglobaldefs
        \ifnum\calx_lscope=0 \calx_lpromote\fi
    \fi
    \ifnum\calx_loutdigits=\XINTdigits\else
        \calx_lreload_a
    \fi
}%
\def\calx_lreload_a{%
    \XINT_global\mathchardef\calx_loutdigits=\XINTdigits
    \mathchardef\calx_lnewdigits=\numexpr4*((\calx_loutdigits+1)/4)\relax
    \ifnum\calx_ldigits=\calx_lnewdigits\else
        \ifnum\calx_ldigits>0
            \expandafter\calx_lZulloop\the\calx_ldigits.\calx_lclearZul{3}%
            \calx_lZlgloop\calx_lclearZlg
        \fi
        \XINT_global\let\calx_ldigits\calx_lnewdigits
        \calx_lreadcoeff
        \ifdefined\calx_lbuildshort
            \calx_lbuildshort
            \let\calx_lbuildshort\undefined
            \let\calx_lexMs\undefined \let\calx_lexMt\undefined
        \fi
        \calx_lbuildfamilies
        \calx_lbuildworking
        \calx_lfreecoeff 0.\calx_tfreecoeff 0.%
```

```
    \fi
    \calx_linstall{Log}%
    \calx_linstall{LogOnePlus}%
    \calx_linstall{LogTen}%
    \calx_linstall{Exp}%
    \calx_linstall{PowTen}%
    \calx_linstall{ExpMinusOne}%
    \calx_linstall{Sinh}%
    \calx_linstall{Cosh}%
    \calx_linstall{Tanh}%
    \calx_linstall{Coth}%
    \calx_linstall{Sech}%
    \calx_linstall{Csch}%
    \calx_linstall{Asinh}%
    \calx_linstall{Acosh}%
    \calx_linstall{Atanh}%
    \calx_linstall{Acoth}%
    \calx_linstall{Asech}%
    \calx_linstall{Acsch}%
    \ifnum\calx_ldigits=\calx_loutdigits
        \XINT_global\let\calx_lSciPow\calx_lPow
    \else
        \XINT_global\def\calx_lSciPow##1##2{\romannumeral0\CALX_fnround\calx_loutdigits
            {\calx_lPow{##1}{##2}}}%
    \fi
    \XINT_global\mathchardef\calx_lscope=\ifxintglobaldefs 1%
        \else\ifnum\currentgrouplevel=0 1\else 0\fi\fi\relax
}%
```

33.20. \xintreloadcalxlog, \xintreloadxintlog

```
\def\xintreloadcalxlog{%
    \ifnum\XINTdigits>100
        \xintMessage{calxlog}{Warning}{Cannot reload at \the\XINTdigits\space digits.
            Maximum is 100; retaining \the\calx_loutdigits\space digits.}%
    \else\expandafter\calx_lreload\fi
}%
\let\xintreloadxintlog\xintreloadcalxlog
\xintreloadcalxlog
```

---

34. Library calxlog data tables

Added at calx 2.0a1.

calxlogdata.tex -- table constants shared by log and potential exp. Full table: floor(log(factor)*10^112); native P16 cache: 24 places. P1..P8 (s47): log tables at 10^-12, see calxlog.tex (\calx_llgULoadT/V).

The fixed data are loaded once; the logarithm pairs are read whenever the required number of pairs changes. Reciprocal coefficients are loaded temporarily at reconstruction. The full precision is kept here. This file is loaded from calxlog.tex in a context having done \XINTsetcatcodes, in particular the exclamation mark has catcode 12.

The definition of \ver@calxlogdata.tex control sequence done also under existence of LaTeX \ProvidesFile is due to a miniltx feature (under investigation).

```
\ifnum\calx_ldigits=0
\ifdefined\ProvidesFile
  \ProvidesFile{calxlogdata.tex}[2026/11/26 v2.0a2 calx log data (JFB)]%
\fi
\expandafter\def\csname ver@calxlogdata.tex\endcsname
  {2026/11/26 v2.0a2 calx log data (JFB)}%
\calx_llgULoadT
10.909.0.954.1018.4805,11.833.0.1827.2163.6815,12.769.0.2626.6430.9476,%
13.714.0.3368.7231.6643,14.666.0.4064.6560.8442,15.625.0.4700.362.9246,%
16.588.0.5310.2833.1084,17.555.0.5887.8716.5236,18.526.0.6424.5406.6244,%
19.500.0.6931.4718.560,20.476.0.7423.3742.4751,21.454.0.7896.5808.941,%
22.434.0.8347.1074.4882,23.416.0.8770.7001.8721,24.400.0.9162.9073.1874,%
25.384.0.9571.1272.6394,26.370.0.9942.5227.3344,27.357.1.300.1949.7202,%
28.344.1.671.1362.1609,29.333.1.996.1278.9002,30.322.1.1332.373.3438,%
31.312.1.1647.5209.1173,32.303.1.1940.2247.3473,33.294.1.2241.7551.1643,%
34.285.1.2552.6609.8713,35.277.1.2837.3777.2795,36.270.1.3093.3331.9984,%
37.263.1.3356.124.6804,38.256.1.3625.7783.4503,39.250.1.3862.9436.1120,%
40.243.1.4146.9383.5642,41.238.1.4354.8460.5311,42.232.1.4610.1790.7316,%
43.227.1.4828.526.1501,44.222.1.5050.7789.7110,45.217.1.5278.5792.5442,%
46.212.1.5511.6900.4310,47.208.1.5702.1719.9281,48.204.1.5896.3528.5138,%
49.200.1.6094.3791.2434,50.196.1.6296.4061.9752,51.192.1.6502.5990.6954,%
52.188.1.6713.1331.6152,53.185.1.6873.9945.3904,54.181.1.7092.5824.7716,%
55.178.1.7259.7172.8690,56.175.1.7429.6930.5059,57.172.1.7602.6080.2169,%
58.169.1.7778.5656.4059,59.166.1.7957.6749.626,60.163.1.8140.507.8175,%
61.161.1.8263.5091.3998,62.158.1.8451.6024.5955,63.156.1.8578.9927.1733,%
64.153.1.8773.1735.7590,65.151.1.8904.7544.2167,66.149.1.9038.897.3037,%
67.147.1.9173.2269.2203,68.144.1.9379.4197.9406,69.142.1.9519.2822.1381,%
70.140.1.9661.1285.6373,71.138.1.9805.159.3825,72.136.1.9951.39.3246,%
73.135.2.24.8050.544,74.133.2.174.615.760,75.131.2.325.5795.5781,%
76.129.2.479.4287.4620,77.128.2.557.2501.5063,78.126.2.714.7337.2031,%
79.125.2.794.4154.1680,80.123.2.955.7092.3610,81.121.2.1119.6473.3385,%
82.120.2.1202.6353.6200,83.119.2.1286.3178.5871,84.117.2.1455.8134.4184,%
85.116.2.1541.6508.7876,86.114.2.1715.5683.588,87.113.2.1803.6746.270,%
88.112.2.1892.5640.7687,89.111.2.1982.2507.7670,90.109.2.2164.739.6753,%
91.108.2.2256.2405.1858,92.107.2.2349.2644.4520,93.106.2.2443.1618.4870,%
94.105.2.2537.9492.8825,95.104.2.2633.6437.9841,96.103.2.2730.2629.753,%
97.102.2.2827.8246.5698,98.101.2.2926.3476.2141,99.100.2.3025.8509.2994,%
0.0.0.0.0.0,%
\calx_llgULoadV
0.0.0.0,1.9.9950.333,2.19.9800.2663,3.29.9550.8980,4.39.9202.1270,%
```

```
5.49.8754.1511,6.59.8207.1678,7.69.7561.3736,8.79.6816.9649,9.89.5974.1371,%
10.99.5033.853,11.109.3994.38,12.119.2857.865,13.129.1622.5267,%
14.139.290.5169,15.148.8861.2494,16.158.7334.9156,17.168.5711.7066,%
18.178.3991.8128,19.188.2175.4241,20.198.262.7296,21.207.8253.9183,%
22.217.6149.1782,23.227.3948.6969,24.237.1652.6617,25.246.9261.2590,%
26.256.6774.6749,27.266.4193.946,28.276.1516.7033,29.285.8745.6852,%
30.295.5880.2242,31.305.2920.5035,32.314.9866.7059,33.324.6719.138,%
34.334.3477.6086,35.344.142.6717,36.353.6714.3837,37.363.3192.9247,%
38.372.9578.4744,39.382.5871.2117,40.392.2071.3153,41.401.8178.9633,%
42.411.4194.3331,43.421.117.6019,44.430.5948.9460,45.440.1688.5417,%
46.449.7336.5643,47.459.2893.1888,48.468.8358.5899,49.478.3732.9414,%
50.487.9016.4169,51.497.4209.1895,52.506.9311.4316,53.516.4323.3152,%
54.525.9245.119,55.535.4076.6928,56.544.8818.5284,57.554.3470.6888,%
58.563.8033.3436,59.573.2506.6619,60.582.6890.8124,61.592.1185.9632,%
62.601.5392.2820,63.610.9509.9360,64.620.3539.919,65.629.7479.9161,%
66.639.1332.5744,67.648.5097.2320,68.657.8774.538,69.667.2363.2043,%
70.676.5864.8474,71.685.9279.1466,72.695.2606.2649,73.704.5846.3649,%
74.713.8999.6087,75.723.2066.1580,76.732.5046.1740,77.741.7939.8174,%
78.751.747.2487,79.760.3468.6276,80.769.6104.1136,81.778.8653.8657,%
82.788.1118.424,83.797.3496.8019,84.806.5790.3017,85.815.7998.6992,%
86.825.122.1512,87.834.2160.8139,88.843.4114.8434,89.852.5984.3951,%
90.861.7769.6241,91.870.9470.6851,92.880.1087.7323,93.889.2620.9194,%
94.898.4070.4000,95.907.5436.3268,96.916.6718.8526,97.925.7918.1293,%
98.934.9034.3087,99.944.67.5421,100.953.1017.9804,101.962.1885.7741,%
-1.0.0.0,%
% log10 (s47): 1/ln10 truncated to 130 digits
\def\calx_lLzinvraw{434294481903251827651128918916605082294397005803666 56%
                    61144537831658646492088707747292249493384317483187061%
                    06744766303733641679287 1[-130]}%
% exp (s49): bucket table i.key.c. (gen_expdata.py, exp_bounds.py)
\def\calx_lexAlist{%
0.0.10000.1.10.9524.2.10.9524.3.10.9524.4.11.8696.5.11.8696.6.11.8696.7.11.%
8696.%
8.12.8000.9.12.8000.10.12.8000.11.13.7407.12.13.7407.13.13.7407.14.13.7407.%
15.14.%
6897.16.14.6897.17.14.6897.18.15.6452.19.15.6452.20.16.6061.21.16.6061.22.16.%
6061.%
23.17.5714.24.17.5714.25.17.5714.26.18.5405.27.18.5405.28.19.5128.29.19.5128.%
30.20.%
4878.31.20.4878.32.21.4651.33.21.4651.34.22.4444.35.22.4444.36.23.4255.37.23.%
4255.%
38.24.4082.39.24.4082.40.25.3922.41.26.3774.42.26.3774.43.27.3636.44.27.3636.%
45.28.%
3509.46.29.3390.47.29.3390.48.30.3279.49.31.3175.50.31.3175.51.32.3077.52.33.%
2985.53.%
34.2899.54.35.2817.55.35.2817.56.36.2740.57.37.2667.58.38.2597.59.39.2532.60.%
40.2469.%
61.41.2410.62.42.2353.63.43.2299.64.44.2247.65.45.2198.66.46.2151.67.47.2105.%
68.48.2062.69.49.2020.70.50.1980.71.51.1942.72.53.1869.73.54.1835.74.55.1802.%
75.56.1770.76.58.1709.77.59.1681.78.60.1653.79.62.1600.80.63.1575.81.65.1527.%
82.66.1504.83.68.1460.84.69.1439.85.71.1399.86.73.1361.87.74.1342.88.76.1307.%
89.78.1274.90.80.1242.91.82.1212.92.84.1183.93.86.1156.94.88.1130.95.90.1105.%
```

```
96.92.1081.97.94.1058.98.96.1036.99.1.1000.-1.0.0.}%
% ln10 limbs 3..31 (10^-12 .. 10^-124, floor), exp rare path P > 16
\def\calx_lexLx{2994!0456!8401!7991!4546!8436!4207!6011!0148!8628!7729!7603!%
3327!9009!6757!2609!6773!5248!0235!9972!0508!9598!2983!4196!7784!0422!8624!%
8633!4095!}%
% calxlogdata_s50.tex -- generated by calx/t/s50/gen_logdata.py (s50: 128 places).
% Floors at 10^-128: F1!c2!...!c16!, F1 = floor(10^8 value) (see generator).
\@namedef{calx_lLz}{230258509!29940456!84017991!45468436!42076011!01488628!%
77297603!33279009!67572609!67735248!02359972!05089598!29834196!77840422!%
86248633!40952546!}%
\def\calx_lLzraw{230258509299404568401799145468436420760110148862872976033327%
900967572609677352480235997205089598298341967784042286248633409525 46[-128]}%
\fi
\ifdefined\calx_lreadnewpairs
\calx_lpairdef{calx_lLb0}{0!0!0!0!0!0!0!0!0!0!0!0!0!0!0!0!}%
\calx_lpairdef{calx_lLs0}{0!0!0!0!0!0!0!0!0!0!0!0!0!0!0!0!}%
\calx_lpairdef{calx_lLt0}{0!0!0!0!0!0!0!0!0!0!0!0!0!0!0!0!}%
\calx_lpairdef{calx_lLu0}{0!0!0!0!0!0!0!0!0!0!0!0!0!0!0!0!}%
\calx_lpairdef{calx_lLv0}{0!0!0!0!0!0!0!0!0!0!0!0!0!0!0!0!}%
\calx_lpairdef{calx_lLb10}{4877016!43694293!36438707!09756716!44757540!30750384!%
86035967!62043359!32455608!94122546!68018852!73358077!49141346!88747895!%
70469625!61742890!}%
\calx_lpairdef{calx_lLb11}{13972194!31751373!64678175!44301696!52712039!97184836!%
45331971!42500516!42479531!64743535!51508681!01872546!03600091!89139520!%
03589419!21760533!}%
\calx_lpairdef{calx_lLb12}{22314355!13142097!55766295!09030983!45033746!01085548!%
00721367!12878724!87391743!76826833!34184072!24100342!23571596!33409805!%
74191432!35296475!}%
\calx_lpairdef{calx_lLb13}{30015959!39628935!41371602!37944702!72769808!91876679!%
20550559!77179025!65849303!27944414!71217655!17752346!84864281!78978044!%
07633012!93937294!}%
\calx_lpairdef{calx_lLb14}{37149855!85448914!96543805!99399681!89888761!22427518!%
37928342!35216155!62356195!60013023!92061052!65723651!62595342!90221061!%
60995397!73700552!}%
\calx_lpairdef{calx_lLb15}{43819493!27310832!55733785!23617541!65471959!89052525!%
87033936!52180080!48059180!90773175!48005880!42744461!37180218!14169936!%
47502086!74779789!}%
\calx_lpairdef{calx_lLb16}{50071029!00248977!04817722!77652256!23878508!06925355!%
80183145!51133066!87643331!59412358!58614086!13722162!08351011!51923350!%
90024935!80227559!}%
\calx_lpairdef{calx_lLb17}{55966578!91854643!54500117!66979777!15388453!04460706!%
78467197!45551708!20457419!05438256!30576210!61284747!03442015!02755367!%
29888029!74325560!}%
\calx_lpairdef{calx_lLb18}{61526064!19028740!83843502!98392889!93635403!73783385!%
91440702!15199881!06299991!51115278!55047640!96677472!78725725!60456750!%
81784009!02261093!}%
\calx_lpairdef{calx_lLb19}{66786937!33756767!67986862!91636132!69403096!71722039!%
36786625!96453530!54332445!52642539!67010711!99040292!36043514!69578056!%
66497065!89571724!}%
\calx_lpairdef{calx_lLb20}{71784979!32003171!43767373!15022836!68141283!92775092!%
81408601!40285319!06563729!45287167!16068233!53055328!44436231!53978467!%
64816121!21726166!}%
```

```
\calx_lpairdef{calx_lLb21}{76550284!27520857!22079452!35855135!97223254!86076623!%
03479508!91977063!29076333!22551495!29152721!67824181!56882994!97868036!%
88895202!88481475!}%
\calx_lpairdef{calx_lLb22}{81103022!12166621!22291359!73094298!52375408!16244659!%
55947074!21185140!50917687!25443990!51525273!51620964!26902069!12749814!%
46642489!12169377!}%
\calx_lpairdef{calx_lLb23}{85449033!09687082!26296357!98266962!59085363!46698238!%
50227657!71156537!20128458!02659279!27164323!16440050!10743189!86672852!%
03627352!27378869!}%
\calx_lpairdef{calx_lLb24}{89599802!86063926!33176801!01931048!64087813!94244530!%
97917766!54000859!81706601!01518340!81170410!67394256!72104538!72969006!%
54874881!45408646!}%
\calx_lpairdef{calx_lLb25}{93598336!52198911!48142168!88721877!53280688!50480075!%
98288331!95894431!65902435!80586428!14044440!15203884!61520194!68823765!%
87778395!83479224!}%
\calx_lpairdef{calx_lLb26}{97444964!60476872!10641858!17145218!08581931!34662578!%
70624733!57868416!64120200!72981389!21533933!36838570!80873774!24117445!%
33718380!40540525!}%
\calx_lpairdef{calx_lLb27}{101170091!66788132!83562812!83506306!31280675!41982951!%
47774249!12975559!06502724!73045868!44803505!49996134!88285946!69427376!%
68909526!64125448!}%
\calx_lpairdef{calx_lLb28}{104725399!63929676!20764806!63044528!66252664!79332717!%
51347477!45058339!66362409!74508416!15005122!89535267!18789244!00996677!%
02162762!56866023!}%
\calx_lpairdef{calx_lLb29}{108175517!16016867!92076597!63351454!52461264!83519360!%
57568162!49413339!21649630!25439840!29981857!57955190!72827527!02024514!%
46181009!88988077!}%
\calx_lpairdef{calx_lLb30}{111504659!51315344!84010607!67124674!69902575!43934652!%
50955719!23890832!90556597!96991184!30913325!65498134!15405297!64766107!%
66634102!11947625!}%
\calx_lpairdef{calx_lLb31}{114727746!06493907!13823006!34456878!91564792!94122051!%
97069740!07506130!64747201!55751901!39378259!44569530!63764548!84831726!%
37907428!82654800!}%
\calx_lpairdef{calx_lLb32}{117862999!66541409!08983344!16223132!13675758!70500426!%
86830325!33457619!77682025!65026056!28072227!16147167!61053562!13153719!%
62144607!72580399!}%
\calx_lpairdef{calx_lLb33}{120898534!61494802!74665839!95672020!62843402!08761854!%
35223157!95083908!14517952!40222728!93142549!13153263!44630582!89778697!%
78853610!08020156!}%
\calx_lpairdef{calx_lLb34}{123821924!30545272!41382373!16744899!81989005!09735283!%
89796460!38937306!08689493!94901281!04120082!55717158!20503389!78163285!%
02889390!06510058!}%
\calx_lpairdef{calx_lLb35}{126691260!40998101!36602710!03563519!50675363!25570532!%
32307251!55592916!33338911!22138741!68328657!88830543!13517080!21975014!%
11685930!40074170!}%
\calx_lpairdef{calx_lLb36}{129462717!25940668!27321552!11461505!06641646!69157481!%
49835892!90296523!79178828!19947837!83571288!43535953!29862341!55317854!%
20985311!81541225!}%
\calx_lpairdef{calx_lLb37}{132163084!77941684!66523927!03065545!90012459!21543184!%
39550103!19619124!38031021!00215630!85368194!47354286!24346438!93659442!%
60026574!18280197!}%
\calx_lpairdef{calx_lLb38}{134822816!03134342!91716791!19282203!96007702!58742440!%
```

```
15192416!64018339!77126079!26178152!68340118!78696728!90953963!39272763!%
51849691!85438726!}%
\calx_lpairdef{calx_lLb39}{137357558!87121159!30100300!07344429!78715647!56954390!%
16357346!94784516!39873148!59048283!97347590!76159433!01835517!39687333!%
96150110!71201216!}%
\calx_lpairdef{calx_lLb40}{139877188!26310032!32766847!61636955!29816283!82434501!%
95495733!24122362!03343596!49805311!39953812!72566078!93743078!79268703!%
65498755!30737716!}%
\calx_lpairdef{calx_lLb41}{142295834!54914820!56587563!04534227!68877860!44701870!%
07109042!85814416!91021317!12724799!40020678!20726813!92710829!51310055!%
31388185!09720306!}%
\calx_lpairdef{calx_lLb42}{144689398!32488202!53179391!33853912!92983588!05568252!%
40597413!15379891!96509934!12678166!43197429!20744181!06928471!13853263!%
86480267!59124234!}%
\calx_lpairdef{calx_lLb43}{147011084!72130011!87939051!23091934!46935236!12985341!%
12873515!82159491!99850488!81873920!60797210!20537383!71474139!90511721!%
18861140!10500975!}%
\calx_lpairdef{calx_lLb44}{149298909!97908535!66266969!02134847!43945566!84390979!%
07371037!91104010!48498913!20873902!97253958!63455178!92767582!31389494!%
73967503!00391652!}%
\calx_lpairdef{calx_lLb45}{151503723!70126160!64124935!71743262!52839495!77459709!%
32510562!50554467!07591405!58752431!97289572!32246874!68282158!37947875!%
66455601!26248869!}%
\calx_lpairdef{calx_lLb46}{153665224!27084526!86318311!75063049!63246716!20483066!%
62332989!49401207!86632178!93130952!81887638!12471197!08587090!59837126!%
70419048!28742588!}%
\calx_lpairdef{calx_lLb47}{155826962!58597009!43882490!25711417!17173572!62538271!%
91122294!42886071!55970564!69589149!67573865!96781458!56065725!06919422!%
86302648!10105522!}%
\calx_lpairdef{calx_lLb48}{157890870!73992775!01350328!59183507!78535211!75661349!%
24980374!68872363!47055382!48885874!40256082!25792261!37393118!12145128!%
13413328!94946192!}%
\calx_lpairdef{calx_lLb49}{159948758!15809322!91752543!97568192!68978369!21744328!%
45892404!80383317!07312323!46001885!35974481!81530419!44963544!06978478!%
69078111!65325770!}%
\calx_lpairdef{calx_lLb50}{161948824!82876018!15784308!19078473!53456111!16361943!%
14759529!13473170!03761996!71190807!82798420!70642277!23849324!60870862!%
34330159!55147854!}%
\calx_lpairdef{calx_lLb51}{163886672!31249125!15395120!34995610!28351694!36881376!%
84878173!06444447!68407021!24579761!56578073!10065298!56743717!34091876!%
56680847!78926947!}%
\calx_lpairdef{calx_lLb52}{165810308!44153813!97028520!44087245!10913574!04918497!%
73896757!87041709!01431896!04656781!16943440!67848978!15965748!47571283!%
95604586!18254900!}%
\calx_lpairdef{calx_lLb53}{167718156!45206199!01253552!29770190!21767151!70279977!%
52189983!25295430!07262389!95927543!98356510!15007313!83456955!12466705!%
54040736!44539503!}%
\calx_lpairdef{calx_lLb54}{169554061!14875120!89851774!54635284!00253491!38165233!%
44591823!54048941!18757216!44593294!48272102!05452390!59361022!04538374!%
22943678!67049218!}%
\calx_lpairdef{calx_lLb55}{171368793!38078995!12260729!94037709!59563807!25054657!%
87425966!72643864!76599606!99529406!58913706!80838550!32846540!75726202!%
```

```
96388490!21877870!}%
\calx_lpairdef{calx_lLb56}{173160554!64083079!10042678!75341086!92940924!67118733!%
11494685!32526722!06023421!54802099!64750359!00652429!03722748!94435050!%
15917874!80931506!}%
\calx_lpairdef{calx_lLb57}{174927485!76218997!04882919!47793383!59576082!77865327!%
23170728!78905758!70482478!21324778!29342775!33251478!55162961!06099772!%
31816418!82376550!}%
\calx_lpairdef{calx_lLb58}{176667668!88605918!46335612!62850485!39773479!65765923!%
50139270!04728098!60031668!44262357!34770563!89169342!32080871!69685376!%
85580947!33002611!}%
\calx_lpairdef{calx_lLb59}{178319623!85675671!28338439!07266264!16536515!85015716!%
48863689!47857833!50333205!89809490!90193545!37614419!57129189!01573783!%
96010840!13683953!}%
\calx_lpairdef{calx_lLb60}{179999327!41551585!57484421!11756516!59830250!23221470!%
46955493!89866821!80722662!28875133!20628475!66911064!78613686!08891572!%
70157715!68984053!}%
\calx_lpairdef{calx_lLb61}{181646208!18684268!35162618!38715089!25187758!83332915!%
56353774!87228655!44058022!67148063!84804391!07869060!53315028!54269127!%
22681863!58526588!}%
\calx_lpairdef{calx_lLb62}{183258146!37483101!30367054!42353602!21429002!02439816!%
52493558!39357639!61570731!47592609!80485411!02193521!84651592!96440022!%
89754332!07696998!}%
\calx_lpairdef{calx_lLb63}{184832982!07164492!98974603!93499685!04124211!37400750!%
69636837!92370037!09364014!10480719!01488310!55956698!26719893!67777879!%
46996337!73033069!}%
\calx_lpairdef{calx_lLb64}{186433016!20628904!31524050!70628454!77728277!64181137!%
20923443!06177389!53557424!72684844!38639929!05839528!65365296!01122161!%
03338185!63402795!}%
\calx_lpairdef{calx_lLb65}{187928006!67575503!01841856!72575517!24772303!90473260!%
64605536!13734764!02229259!99252497!30438216!21595652!73997553!80784825!%
62982486!13034382!}%
\calx_lpairdef{calx_lLb66}{189445686!74663976!02164854!54167440!02811308!54780693!%
59160032!94032095!96504115!35513527!03039824!31941553!77470141!21571652!%
74055136!17774361!}%
\calx_lpairdef{calx_lLb67}{190986755!77083839!53570139!14989848!80908592!03746305!%
45925035!16229688!25033728!87519070!57151978!29172962!39722056!07312348!%
03350001!27835472!}%
\calx_lpairdef{calx_lLb68}{192414865!72738006!13544633!38515637!34838230!51486654!%
46600938!36048362!47534231!14176543!18257989!65437517!31322987!04467048!%
96338424!42294755!}%
\calx_lpairdef{calx_lLb69}{193863666!50888149!19138665!25293717!68563035!71259310!%
42351320!04676252!70686728!93565631!16294057!13064676!22575966!27795162!%
61031650!02530424!}%
\calx_lpairdef{calx_lLb70}{195333766!48841099!27489536!64048005!71586763!21191471!%
10021240!06300090!37533107!80405339!49256313!54822299!74977656!69366366!%
20779413!97346582!}%
\calx_lpairdef{calx_lLb71}{196682739!73107015!60679395!44961500!51858743!19415587!%
53896986!34352903!38255138!98028203!09281764!04724256!83671038!05868014!%
90705742!82836563!}%
\calx_lpairdef{calx_lLb72}{198122649!41828809!24186815!74613144!61257586!77162575!%
19100987!53170820!17209199!90849169!64205729!67114909!76718689!44828525!%
34817616!52938461!}%
```

```
\calx_lpairdef{calx_lLb73}{199436536!93247166!64273124!04020619!38515242!26452886!%
22775015!90205837!64110671!35539244!81509061!98092420!95482760!53442981!%
89016134!94412265!}%
\calx_lpairdef{calx_lLb74}{200842405!44445556!33841401!01588853!62161915!17223136!%
15841185!66802388!26883385!98152976!10776415!79259235!94138768!10370178!%
03739679!73175263!}%
\calx_lpairdef{calx_lLb75}{202117263!35558601!52725978!01325602!37809932!25493066!%
83885725!98863509!03927829!35060453!31490611!19686112!74192582!47096446!%
23005715!85388132!}%
\calx_lpairdef{calx_lLb76}{203485065!83519607!88463612!11624558!09768982!89146829!%
99290864!76672804!55958873!60474397!28134285!59356431!55318073!90010046!%
44074731!52084082!}%
\calx_lpairdef{calx_lLb77}{204794287!46204649!37185886!61378471!98825966!77723182!%
01406117!36041043!42364361!94046203!52967838!62371065!51934042!76426928!%
04088610!10102178!}%
\calx_lpairdef{calx_lLb78}{206042353!58440740!80390540!76882760!21982494!68416230!%
90715667!48057838!38272423!91116649!03632338!32745741!96307048!80746441!%
09305344!97292261!}%
\calx_lpairdef{calx_lLb79}{207306193!47157969!02037103!40837470!65628406!25220401!%
45569907!04811753!01614493!94842285!90270090!91652302!89752571!24408104!%
91584122!77340948!}%
\calx_lpairdef{calx_lLb80}{208586210!94827586!61662245!15470150!70299440!53831846!%
15255280!99132159!81420691!09909000!65188771!36767783!42510776!23543793!%
70574676!98984031!}%
\calx_lpairdef{calx_lLb81}{209801292!72652712!98963198!37268654!50384441!24734283!%
38400714!52011167!73596759!27643308!57819172!88192813!35235675!48848520!%
17783255!79692704!}%
\calx_lpairdef{calx_lLb82}{211031320!53469229!74958058!07198558!88327150!32540774!%
22719422!59918895!43997017!94906765!13539663!04809866!97164743!69718036!%
26775269!00925870!}%
\calx_lpairdef{calx_lLb83}{212193159!33007880!77844534!52439782!35754053!95770282!%
15732657!34853939!82517877!91134229!88474823!57638899!66714230!97089347!%
05080886!31643606!}%
\calx_lpairdef{calx_lLb84}{213453150!79977959!58859638!19216363!98911567!85335412!%
81648283!58304326!05781241!30852029!43571809!94889542!68883629!30577777!%
68410358!85004061!}%
\calx_lpairdef{calx_lLb85}{214558134!41843809!33209996!44864272!26087030!55405480!%
61290821!15495249!64264575!27042932!75611854!42200841!22868070!51019021!%
80292783!96041345!}%
\calx_lpairdef{calx_lLb86}{215761932!27438599!56738432!34007485!06230769!99660623!%
09182129!74761091!18918043!16815792!79080724!30290549!92324717!16500336!%
29508483!47187570!}%
\calx_lpairdef{calx_lLb87}{216892870!81813720!80234034!53717611!30262355!15714943!%
50723775!66577287!38116590!53561660!94508621!91128425!68166948!10310841!%
53301743!53855000!}%
\calx_lpairdef{calx_lLb88}{218036746!02697964!83471842!87043707!09507739!74077183!%
32513335!96356675!59143923!47300736!98718015!12768922!81706324!02315174!%
04046752!25788500!}%
\calx_lpairdef{calx_lLb89}{219193857!29069820!27493883!97499206!69723337!00801332!%
75469147!44443730!59954865!06846106!48153008!13362001!82107236!06563598!%
46380387!81724787!}%
\calx_lpairdef{calx_lLb90}{220273975!80243295!45168184!42607282!40504853!33642900!%
```

```
62227850!43804638!50354207!87641069!60021854!10787221!14628029!83790074!%
79652710!98434929!}%
\calx_lpairdef{calx_lLb91}{221365888!37996441!75405852!90816927!26949050!34492475!%
25900892!70834169!28050891!18852080!99649483!20311866!24284094!65056767!%
24499844!48748048!}%
\calx_lpairdef{calx_lLb92}{222469855!43369744!58444262!31715508!70030659!75137654!%
43212893!41678795!80478979!21881055!01348979!74770652!39805722!23486967!%
97346908!74661616!}%
\calx_lpairdef{calx_lLb93}{223492644!45202308!78749575!54702981!56212401!58189754!%
91137453!98669836!56000459!50206478!41506729!11767184!24511488!70781789!%
96725523!68614754!}%
\calx_lpairdef{calx_lLb94}{224620475!95579380!45041228!03365888!80254441!45190473!%
96516601!53608394!10910324!77011860!89413233!09055984!87711747!52280059!%
14035008!23512435!}%
\calx_lpairdef{calx_lLb95}{225665616!11056458!81477239!05931629!14166080!30059231!%
98006317!22286710!15813007!94600701!88692497!11340940!57964497!87770411!%
20889925!46701023!}%
\calx_lpairdef{calx_lLb96}{226721794!91567543!67764053!16134647!68007162!43369045!%
08404378!76487537!52933585!02759015!65455821!34105515!63050899!75276312!%
56814768!76172983!}%
\calx_lpairdef{calx_lLb97}{227691734!62454678!92459970!55462384!96111490!54885387!%
45947627!05883570!55121235!01969101!60031198!45105038!34366491!89347299!%
50575223!82661515!}%
\calx_lpairdef{calx_lLb98}{228769648!05002950!29182581!70970617!78557472!06956741!%
96891483!96032108!15750807!11183055!52328293!28859485!63292884!35542164!%
11455995!63204106!}%
\calx_lpairdef{calx_lLb99}{229759755!14830066!10405889!25222502!07356643!81139201!%
92939334!81852479!83348652!25505629!28013249!99630456!00924597!26128168!%
11551694!55888592!}%
\calx_lpairdef{calx_lLs-1100}{11653381!62559515!29718167!36869887!91957975!%
42590124!58658795!99582246!47191326!57524871!40249308!58229720!85366664!%
74018577!90550123!71810058!}%
\calx_lpairdef{calx_lLs-1000}{10536051!56578263!01227500!98083931!27983061!%
20372983!27407256!39392336!92584023!24013454!64887656!95462134!12076602!%
77259103!70517148!67351701!}%
\calx_lpairdef{calx_lLs-900}{9431067!94712413!26877142!72436023!00807598!50302915!%
47764718!82201413!34100689!27822551!32847612!48150835!58092161!56817859!%
58634723!56314418!}%
\calx_lpairdef{calx_lLs-800}{8338160!89390510!58394765!83464217!91606088!22393696!%
59970182!66604054!65364729!43051723!10618666!64033364!40724573!53924721!%
00988903!58928182!}%
\calx_lpairdef{calx_lLs-700}{7257069!28348354!30711573!34790384!55001047!73488954!%
20452857!52433958!22669509!53854476!13845138!24029377!87732298!86021295!%
74786709!58050348!}%
\calx_lpairdef{calx_lLs-600}{6187540!37180874!71797800!11813837!81382306!52340877!%
06666474!54674456!34933383!87920917!22554413!29748031!92818548!25131629!%
84300804!10077363!}%
\calx_lpairdef{calx_lLs-500}{5129329!43875505!33426196!14425468!72384392!22361689!%
89941161!60476740!51673529!58284990!71277021!48270277!09008692!86318421!%
06798018!07309191!}%
\calx_lpairdef{calx_lLs-400}{4082199!45202551!29554577!06515531!98701772!11747633!%
52022972!85614208!30682816!28762241!55690620!20383371!07018595!81339157!%
```

```
61202856!02344552!}%
\calx_lpairdef{calx_lLs-300}{3045920!74847085!45919261!28766476!67013932!88074598!%
76459609!04214331!74967927!57651783!51297116!65144922!04675698!48874361!%
92790639!28105957!}%
\calx_lpairdef{calx_lLs-200}{2020270!73175194!48408045!30102419!23878525!33383733!%
56832102!71954925!66591871!88087170!92908140!86007034!85515581!06986522!%
99529709!68602617!}%
\calx_lpairdef{calx_lLs-100}{1005033!58535014!41183548!85755854!77060855!15007674!%
62987337!86994255!29583009!00425031!36497081!92549097!62769327!97840645!%
18767259!82747331!}%
\calx_lpairdef{calx_lLs100}{-995034!91468319!17151784!64245573!92583113!20390059!%
94120213!53904402!33133335!75236108!89673143!03437239!83883547!43948261!%
53515211!92925247!}%
\calx_lpairdef{calx_lLs200}{-1980263!27038202!86973970!93311489!96068910!09272488!%
79645891!40437209!21964728!36546999!70804204!60331542!87283561!57959508!%
56888499!36793362!}%
\calx_lpairdef{calx_lLs300}{-2955881!77584555!97267380!59431528!75945739!41868886!%
74278945!54248429!04614014!90691069!08802484!35127087!69109980!19646489!%
80605270!21838109!}%
\calx_lpairdef{calx_lLs400}{-3922072!68467187!03730799!10342888!01061704!34629416!%
57307328!55702627!91499686!12121063!42548297!65749451!04528867!29329300!%
97718434!23033573!}%
\calx_lpairdef{calx_lLs500}{-4879017!58305679!96934625!59577683!53413920!26335584!%
41758995!99234268!85892075!67636896!11805862!78684413!37010293!75417910!%
47927713!00328921!}%
\calx_lpairdef{calx_lLs600}{-5826891!18760242!24474281!64888149!40767667!25089986!%
07309489!78463224!23746963!44152038!63314711!38696856!67929982!02665829!%
10948514!79732061!}%
\calx_lpairdef{calx_lLs700}{-6765865!15261851!94731584!09234545!14136390!56701126!%
13839850!65390826!88427849!82471230!39146757!06677585!94677291!92941367!%
10476890!27662207!}%
\calx_lpairdef{calx_lLs800}{-7696105!88638716!75015782!95568479!81651087!31035068!%
78708862!12127820!35875095!75948862!86394204!91745162!95523602!25188455!%
57528572!34399778!}%
\calx_lpairdef{calx_lLs900}{-8617770!37589476!67658666!45715952!67641418!54093754!%
31479899!17822472!54241935!15954197!87279881!38217743!82418594!27066738!%
40537569!46152001!}%
\calx_lpairdef{calx_lLs1000}{-9531018!01956751!39956047!87671923!49077793!94634691!%
35580081!47601918!36998985!76411576!71609424!97086963!50692725!20581541!%
48250111!15395630!}%
\calx_lpairdef{calx_lLs1100}{-10436002!46757572!32272642!00141475!55466546!%
34832267!16033668!70487871!04661179!14573492!81993571!21693295!84711415!%
65867277!18964138!66946740!}%
\calx_lpairdef{calx_lLt-56}{561573!87856356!95176194!38278177!61543268!68308793!%
91517181!23558309!15962043!32803899!02151540!28866630!26136448!78466098!%
40239662!35528060!}%
\calx_lpairdef{calx_lLt-55}{551518!06881101!62377693!95222777!26411903!52527255!%
12337503!49917965!75365621!43919276!22549539!01159756!96870435!83208755!%
63921226!24834084!}%
\calx_lpairdef{calx_lLt-54}{541463!27014988!81751532!38194002!50357803!37572395!%
52091881!01998703!37911481!49929545!14766865!32887073!89481154!85422854!%
80851513!27678733!}%
```

```
\calx_lpairdef{calx_lLt-53}{531409!48237687!93572635!55604600!60359986!38851846!%
76425983!34708024!14770069!31534978!42876462!09776922!18974268!64688264!%
24737071!90854177!}%
\calx_lpairdef{calx_lLt-52}{521356!70528874!51252817!59905599!60052579!08234191!%
27590980!24162993!37232208!78419849!99636509!64942399!42008419!21184552!%
60647660!43171199!}%
\calx_lpairdef{calx_lLt-51}{511304!93868230!21094256!51909687!40372273!86760004!%
46098118!12426340!02334168!83498217!88869136!04398480!28982120!11152292!%
21295741!95803169!}%
\calx_lpairdef{calx_lLt-50}{501254!18235442!82043093!73895836!77813865!97831048!%
32870728!08748234!84439635!94348377!59518734!62877397!23675183!12228934!%
50470719!03868032!}%
\calx_lpairdef{calx_lLt-49}{491204!43610206!25443157!44024929!69137438!30190080!%
34292708!17824785!11097359!36325706!52991895!30706963!82409590!78962662!%
85263599!85712556!}%
\calx_lpairdef{calx_lLt-48}{481155!69972220!54789809!64601389!17139069!80870215!%
92875332!66702224!49820063!50782272!77144038!62242941!81171344!35693157!%
16342384!20459548!}%
\calx_lpairdef{calx_lLt-47}{471107!97301191!85483916!96721077!25486332!80256184!%
41812629!07977902!95200740!13361070!13829415!82355364!28532639!78654804!%
78962218!68965380!}%
\calx_lpairdef{calx_lLt-46}{461061!25576832!44585944!93850963!82012232!98750504!%
86693267!16816596!04522306!42599572!56572951!19372559!19461856!50853149!%
72783811!34580548!}%
\calx_lpairdef{calx_lLt-45}{451015!54778860!70570175!86891313!09638167!65304806!%
67819341!93678992!55221989!83029736!60141289!98088856!15583928!75485092!%
95640477!14382245!}%
\calx_lpairdef{calx_lLt-44}{440970!84887001!13079050!13276372!42639081!16419854!%
87160539!69958933!66674152!56921375!70707905!21785840!75481818!23234704!%
56638634!17006749!}%
\calx_lpairdef{calx_lLt-43}{430927!15880984!32677630!82674781!43653050!62767771!%
58885159!77371495!43641380!50362678!65687635!43863370!46212753!27236656!%
76466277!16222121!}%
\calx_lpairdef{calx_lLt-42}{420884!47740547!00608191!71856149!14053422!92737182!%
85186753!41060052!13655284!43883044!21125485!54812625!62870188!76381407!%
28906642!34145098!}%
\calx_lpairdef{calx_lLt-41}{410842!80445431!98544928!41295471!67424350!14335973!%
66300420!37155716!41114732!97677835!06367836!01264693!38324131!66866080!%
29643175!48122949!}%
\calx_lpairdef{calx_lLt-40}{400802!13975388!18348792!66092284!33289755!00582679!%
82259793!59858873!39857301!98506843!67842219!09491651!99692597!76229563!%
08879610!78730834!}%
\calx_lpairdef{calx_lLt-39}{390762!48310170!61822449!73786658!36320641!67674955!%
15626246!15291290!37953584!17202886!41515383!19551790!17064542!07265281!%
15109350!14779210!}%
\calx_lpairdef{calx_lLt-38}{380723!83429540!40465358!81659365!55365109!28149677!%
38291516!81454275!63412613!18422060!91908758!09066602!86527107!31799415!%
17033879!68443441!}%
\calx_lpairdef{calx_lLt-37}{370686!19313264!75228976!26108742!07187905!42696002!%
04301184!05770254!22840356!21070758!06135929!04917745!93839003!05014439!%
81173975!04744956!}%
\calx_lpairdef{calx_lLt-36}{360649!55941116!96272081!76701987!22149973!77430863!%
```

```
13975964!33137680!58680347!32072955!85869124!68444888!60748530!27797458!%
95249318!89809424!}%
\calx_lpairdef{calx_lLt-35}{350613!93292876!42716227!27503832!83580923!46839881!%
43818445!44063582!72070751!21036324!08603957!00615547!22507922!47510632!%
69760565!80389381!}%
\calx_lpairdef{calx_lLt-34}{340579!31348328!62401308!58290714!89676019!72015814!%
88014777!66755111!36389036!79454316!83260599!35220043!43432535!95136250!%
56476010!96416986!}%
\calx_lpairdef{calx_lLt-33}{330545!70087265!11641259!58263770!66761126!34161786!%
88978206!21939114!68640439!48115807!63797865!02533387!25840458!55739861!%
47534168!77040521!}%
\calx_lpairdef{calx_lLt-32}{320513!09489483!54979869!04879172!16090606!60300359!%
25033508!45038386!71182195!29910554!42858134!61835371!17318951!53516542!%
29163544!07131831!}%
\calx_lpairdef{calx_lLt-31}{310481!49534787!64946719!90419490!13350714!70069972!%
70046123!68877017!14752570!18367869!93807272!67321581!51788429!67088460!%
15076705!86315976!}%
\calx_lpairdef{calx_lLt-30}{300450!90202987!21813250!88934962!01110315!32002627!%
33817267!62979648!89868993!66290844!38843128!50075526!84354940!01172005!%
83374650!12888709!}%
\calx_lpairdef{calx_lLt-29}{290421!31473898!13348940!56188712!19967305!23275664!%
22314178!31838854!84160989!79516208!47854995!33674780!44681891!08724277!%
34588987!52253576!}%
\calx_lpairdef{calx_lLt-28}{280392!73327342!34577613!55245144!14457954!04622376!%
81157408!91188750!12722664!30153329!89215094!11820793!07547623!17958158!%
48720328!66984024!}%
\calx_lpairdef{calx_lLt-27}{270365!15743147!87533869!00345890!25302230!46900626!%
01373266!92832241!18745096!99073445!76671269!35352216!93812142!33923980!%
27930793!83865735!}%
\calx_lpairdef{calx_lLt-26}{260338!58701148!81019631!11722868!40625359!43290820!%
00880376!06955235!70411073!77110391!80622357!61997714!49874623!19979881!%
02407651!19212023!}%
\calx_lpairdef{calx_lLt-25}{250313!02181185!30360821!74003152!25798312!48697274!%
31700157!59711009!37565605!25921886!83082884!26954690!46018986!61736331!%
54725731!07638112!}%
\calx_lpairdef{calx_lLt-24}{240288!46163103!57164154!90865516!84851242!56452630!%
02664765!79159588!24469617!78948700!22954397!26467283!42119836!14280141!%
69311058!26368745!}%
\calx_lpairdef{calx_lLt-23}{230264!90626755!89074053!28613671!26407233!57318515!%
64123216!01201320!41416062!71215865!45134884!55843429!26555945!80371652!%
09301888!30521394!}%
\calx_lpairdef{calx_lLt-22}{220242!35552000!59529686!41336336!14131970!28416893!%
12851928!46283775!00775954!81340044!33676481!50474888!73255161!73903593!%
38947703!65108822!}%
\calx_lpairdef{calx_lLt-21}{210220!80918702!07522130!70329465!76170496!08663622!%
10073816!78025748!90545313!36748334!07145012!07006197!54615794!91684630!%
32228317!32253884!}%
\calx_lpairdef{calx_lLt-20}{200200!26706730!77351651!10461052!30317193!49408528!%
27793340!24730508!63971963!17752605!12137547!44880370!22030667!72305768!%
01438630!45568933!}%
\calx_lpairdef{calx_lLt-19}{190180!72895963!18385104!36164083!52111200!20700248!%
95563545!57034090!54648246!05987247!40125567!92808426!60770587!05723622!%
```

```
22559384!14077957!}%
\calx_lpairdef{calx_lLt-18}{180162!19466281!84813463!79748354!72037995!01845251!%
94774779!78879944!81342291!40529640!71382010!71285021!53390563!38104823!%
53186177!49955751!}%
\calx_lpairdef{calx_lLt-17}{170144!66397575!35409465!54726961!35919813!88280100!%
22941920!59873765!88050098!89352496!80392071!36836153!41598347!19813803!%
81826100!08120769!}%
\calx_lpairdef{calx_lLt-16}{160128!13669738!33285376!16858419!99763476!46377050!%
09330301!41316836!22182502!40883304!98238917!86132822!11547463!10668458!%
58921290!25378125!}%
\calx_lpairdef{calx_lLt-15}{150112!61262671!45650881!55610482!57174335!68336188!%
52621857!47726216!73561757!05045043!71389875!78527844!47645493!95402384!%
27322774!29225713!}%
\calx_lpairdef{calx_lLt-14}{140098!09156281!43571097!08756822!14309254!69878679!%
19847067!60134167!50571685!78790676!13223829!47993661!20146366!04847537!%
70142728!67509440!}%
\calx_lpairdef{calx_lLt-13}{130084!57330481!01724698!92822877!34599244!48395095!%
59953540!98869842!14475937!72285150!62547601!46981437!09977414!36750829!%
77554949!21715236!}%
\calx_lpairdef{calx_lLt-12}{120072!05765188!98162176!42102247!73492767!98912938!%
19071253!33295176!83794905!43346957!46954920!59069383!18716765!99137776!%
45447730!31723807!}%
\calx_lpairdef{calx_lLt-11}{110060!54440330!14064205!48970133!12622471!36903992!%
28432293!72379366!09777110!46844572!85860766!71512217!40100007!99981865!%
44220718!33447946!}%
\calx_lpairdef{calx_lLt-10}{100050!03335835!33500142!98225406!83449607!55205250!%
43440925!09880207!97245202!38586947!46881228!17155429!96788018!43126380!%
89481287!34298441!}%
\calx_lpairdef{calx_lLt-9}{90040!52431641!43186641!88198005!42959672!29943485!%
50738554!58462760!79581942!48342852!60099601!00272009!73785678!68520851!%
73395909!49944067!}%
\calx_lpairdef{calx_lLt-8}{80032!01707691!32246387!31363406!28737409!38810305!%
25378310!28346523!44014370!23745105!93984628!62872708!36340931!53335882!%
41562714!35592633!}%
\calx_lpairdef{calx_lLt-7}{70024!51143933!91966953!27211048!78106623!25063859!%
67262590!53728599!07061816!82540631!97990052!42185665!05230457!64343152!%
02267551!10920163!}%
\calx_lpairdef{calx_lLt-6}{60018!00720324!15559780!00118635!46347060!42718791!%
14255264!68385248!26403392!91744966!35038991!47359499!79100643!54240016!%
51001127!42376027!}%
\calx_lpairdef{calx_lLt-5}{50012!50416822!97919271!94989326!02663514!53209785!%
57440037!73996927!49289687!18964788!51259409!49714909!34584454!81481845!%
57016951!30941720!}%
\calx_lpairdef{calx_lLt-4}{40008!00213397!35382016!23413909!19947435!42656222!%
36477994!12392134!88556600!24634170!63712345!46338577!72799142!64946031!%
56418209!05395980!}%
\calx_lpairdef{calx_lLt-3}{30004!50090020!25486121!53125106!05804477!93944169!%
14731057!00846962!14359465!89361037!58760482!46256368!14043085!76980917!%
60140462!78953450!}%
\calx_lpairdef{calx_lLt-2}{20002!00026670!66730677!33516222!48188038!92655810!%
37182939!84597792!85285350!95568687!46570819!02951022!81908997!36091767!%
12775420!80131040!}%
```

```
\calx_lpairdef{calx_lLt-1}{10000!50003333!58335333!50001428!69643968!35397734!%
57107551!40898657!62716344!75661140!26170224!95986337!46652875!41788906!%
72282471!75672579!}%
\calx_lpairdef{calx_lLt1}{-10000!49996666!91664666!83331904!88689365!17935598!%
92892448!73387056!78355083!92592828!09226600!52787111!39025335!33724468!%
86197936!79034935!}%
\calx_lpairdef{calx_lLt2}{-19999!99973337!33269343!99817174!85145499!20678205!%
62819400!72553256!29029490!44218709!82064449!60207540!32632732!51802363!%
45109956!45229722!}%
\calx_lpairdef{calx_lLt3}{-29996!49910020!24514121!46876534!19207331!86144404!%
35952224!33105306!98330115!73603145!32634782!28501332!85217791!59380648!%
57100042!13598957!}%
\calx_lpairdef{calx_lLt4}{-39993!99786730!64619349!09935808!13595613!13473314!%
01869933!53050282!04084771!51941425!25845158!18893465!23422240!47924899!%
96560621!55117039!}%
\calx_lpairdef{calx_lLt5}{-49988!49583489!52085936!38441663!57622944!20813660!%
64490956!56636496!32843130!66049255!79484709!80480630!25677973!80057849!%
74594269!12906045!}%
\calx_lpairdef{calx_lLt6}{-59983!99280323!84455772!00301268!65746178!40078376!%
80628846!84751742!45454019!29060322!28105253!52043912!23177756!56808678!%
90567499!01260551!}%
\calx_lpairdef{calx_lLt7}{-69976!48857266!58072263!07563485!36721782!95151642!%
71789357!64401950!29442757!12716047!18127425!03588078!72079745!05686339!%
06763254!11903431!}%
\calx_lpairdef{calx_lLt8}{-79969!98294356!67840994!06164233!19344403!27775433!%
67332262!78044032!71866120!84302271!93521852!68433462!65442111!50513758!%
00644738!66374208!}%
\calx_lpairdef{calx_lLt9}{-89960!47571639!06990505!22563681!79397678!61678609!%
14785573!89058515!04544752!32860889!30026124!56112318!12228250!66734017!%
23744085!96255333!}%
\calx_lpairdef{calx_lLt10}{-99951!96669164!66833190!60107946!49885392!44937606!%
83344800!29803331!71099675!04234128!04457037!45237799!08784886!77399401!%
06884834!71710048!}%
\calx_lpairdef{calx_lLt11}{-109940!45566990!36526315!37952972!33201012!88367961!%
55231107!66831213!98626425!16076951!54934399!87837279!99008250!31828564!%
48505676!77978257!}%
\calx_lpairdef{calx_lLt12}{-119929!94245179!02833152!65393300!09994936!99662609!%
35541346!67246993!16084659!30586238!17625399!08670664!48334765!16993109!%
27476628!31799982!}%
\calx_lpairdef{calx_lLt13}{-129916!42683799!49884239!44443411!95471288!84750149!%
62632654!13955571!03723744!24673011!91038707!37169063!71947950!53040121!%
67956078!34801201!}%
\calx_lpairdef{calx_lLt14}{-139903!90862926!58938751!93524353!70049563!84724859!%
49107635!76711920!35795780!73300475!57703933!60206736!64116859!07354448!%
34347635!54230720!}%
\calx_lpairdef{calx_lLt15}{-149888!38762641!08145999!85113326!98120033!63016511!%
11532240!36085017!42950813!78893940!74839087!02268117!35860333!84593890!%
06887819!87858673!}%
\calx_lpairdef{calx_lLt16}{-159873!86363029!72307039!90218936!36364538!06014376!%
54570538!15479114!41359387!81013177!25507806!63475913!87503673!48505304!%
73988746!70914897!}%
\calx_lpairdef{calx_lLt17}{-169856!33644185!22636408!22549764!22164648!71798553!%
```

```
76475742!17089830!02223062!35802389!40008231!12843404!17016878!55342843!%
29393376!47412684!}%
\calx_lpairdef{calx_lLt18}{-179839!80586206!26523971!75248930!23335387!43841630!%
02516420!82650467!34722411!92804023!41673063!24512562!81773799!05996591!%
54509168!53965160!}%
\calx_lpairdef{calx_lLt19}{-189820!27169197!47296898!43072272!23162562!93919172!%
62058119!84542262!98164765!17209867!52040846!50742726!98308994!76164240!%
30423982!94334152!}%
\calx_lpairdef{calx_lLt20}{-199801!73373269!43981746!22892761!09842829!91171092!%
32262238!49246129!77472319!36422281!00206583!70652821!86077681!79639063!%
62673820!82768388!}%
\calx_lpairdef{calx_lLt21}{-209780!19178538!71066670!85418734!37284902!97423365!%
75157183!10554312!57479236!48032926!42556105!93134992!07273055!45917480!%
05098666!56016255!}%
\calx_lpairdef{calx_lLt22}{-219759!64565127!78263752!11018501!25184741!11778389!%
94901647!99057663!32137531!16643451!48281921!48993308!20984406!94937801!%
04328859!80472034!}%
\calx_lpairdef{calx_lLt23}{-229736!09513165!10271438!82548835!70693369!22622945!%
19049793!64519333!52195293!54503449!81560618!62179534!83457165!27473198!%
22642553!41858359!}%
\calx_lpairdef{calx_lLt24}{-239713!54002785!06537112!28089834!52209378!87403671!%
31740078!54034261!26937308!70135399!56281129!03423515!78865719!92665932!%
10217602!74161500!}%
\calx_lpairdef{calx_lLt25}{-249689!98014128!01019768!06493573!18204752!65925038!%
27465521!87394076!82375555!71009552!44225694!29566093!18140644!77293337!%
74303770!97685651!}%
\calx_lpairdef{calx_lLt26}{-259663!41527340!21952816!28658944!70887831!77974707!%
54889012!75507527!19212743!90871221!39672853!34241857!61324827!36228999!%
59115353!70360246!}%
\calx_lpairdef{calx_lLt27}{-269637!84522573!91607000!07450014!76275987!96437930!%
93207995!91845274!81956190!86942325!55608966!56007042!64869844!79847083!%
55565953!85721105!}%
\calx_lpairdef{calx_lLt28}{-279609!26979987!26053432!29180170!59247506!76487450!%
91450081!84426527!19606664!59005920!97898548!02251108!23143167!80991459!%
41526473!05080183!}%
\calx_lpairdef{calx_lLt29}{-289581!68879744!34926750!39589281!74721624!50620150!%
43855866!32752421!88942232!95154733!17513125!29368812!67341631!40273085!%
01085879!14832669!}%
\calx_lpairdef{calx_lLt30}{-299551!10202015!21188389!37246026!84631517!54663970!%
11961129!07092679!61225138!96569868!24365567!87428658!17849711!12105237!%
86726325!77972644!}%
\calx_lpairdef{calx_lLt31}{-309521!50926975!80889972!67312473!15160898!72260074!%
34656099!25035119!01377923!83147303!95062810!29786787!35005877!61263413!%
42990583!66249130!}%
\calx_lpairdef{calx_lLt32}{-319490!91034808!02936821!08612924!20163958!28744014!%
77544215!93692867!19355331!63458808!98577066!65740611!66720013!88239120!%
16250985!25049374!}%
\calx_lpairdef{calx_lLt33}{-329457!30505699!68851579!46953975!35133569!55606166!%
39651581!38752784!53299115!84840226!50521533!34981810!25260711!82717202!%
34380636!12754302!}%
\calx_lpairdef{calx_lLt34}{-339424!69319844!52537961!27647637!71876485!39998518!%
01828467!40264966!08249805!50253992!00201937!72543394!01850463!80505907!%
```

```
36966577!64279468!}%
\calx_lpairdef{calx_lLt35}{-349389!07457442!20044610!80194307!77548851!95210130!%
08845564!09026825!22986587!12854729!14342175!31437003!90542724!96472676!%
34316637!13724139!}%
\calx_lpairdef{calx_lLt36}{-359354!44898698!29329083!08087272!63252580!70248455!%
38870530!34855349!03361117!92741830!21268999!39716791!69638411!54524904!%
42513819!43883153!}%
\calx_lpairdef{calx_lLt37}{-369318!81623824!30021941!36705349!97344419!38143979!%
94947068!42503080!03784511!26707026!93788376!39105776!04421329!46348061!%
62906318!68548974!}%
\calx_lpairdef{calx_lLt38}{-379280!17613037!63190972!12265166!57316065!26176947!%
25633904!95642949!93152987!42815451!16688577!41700638!96012381!28375522!%
06458145!24615761!}%
\calx_lpairdef{calx_lLt39}{-389242!52846561!61105517!44809480!41916141!62442395!%
07511819!75782869!22551963!10567998!31707588!77585140!03825540!21223288!%
75681474!59559001!}%
\calx_lpairdef{calx_lLt40}{-399203!87304625!47000924!88212848!62453729!58581582!%
07048422!29302064!81489177!81446227!44068022!53961987!05532439!58809660!%
98603333!92113591!}%
\calx_lpairdef{calx_lLt41}{-409162!20967464!34843114!50190836!59298484!73973286!%
01872581!68651518!24002944!55920514!24365338!88936029!99246003!21604407!%
68473566!07924725!}%
\calx_lpairdef{calx_lLt42}{-419121!53815319!29093263!25303853!26823893!95589652!%
21851341!81002329!35262250!98017779!16171579!72444973!87207269!98306304!%
05564628!79252793!}%
\calx_lpairdef{calx_lLt43}{-429079!85828437!24472606!43951582!77777316!79189977!%
02118350!82327226!70564200!99394588!89554521!07382550!17631048!69463294!%
18368409!42474018!}%
\calx_lpairdef{calx_lLt44}{-439035!16987071!05727356!30358864!36652135!04524022!%
99161719!64561778!58544605!29803339!00239343!15774540!83255901!11209751!%
32315281!92450127!}%
\calx_lpairdef{calx_lLt45}{-448991!47271479!47393737!62558750!21432282!24671683!%
55017168!26205022!60973109!09195611!47762208!03717483!71754379!09553403!%
52657963!71596095!}%
\calx_lpairdef{calx_lLt46}{-458946!76661927!13563140!27383344!54425974!52490782!%
31232480!93982417!03252976!19268258!90505856!15489698!98579324!82352236!%
31179598!61420896!}%
\calx_lpairdef{calx_lLt47}{-468899!05138684!57647388!63477897!36151600!68321748!%
55795930!27751604!35002193!64102071!64225208!42851546!25787363!78859452!%
41035494!48753980!}%
\calx_lpairdef{calx_lLt48}{-478852!32682028!22144127!85358492!21732091!61517489!%
91002348!10455491!78203158!49112000!94872990!95852652!77210463!80838905!%
64167844!55863977!}%
\calx_lpairdef{calx_lLt49}{-488804!59272240!38402326!81538529!57341970!46821567!%
13507223!36151916!74614307!37763687!86270870!62475558!86104662!32994524!%
90295001!04872996!}%
\calx_lpairdef{calx_lLt50}{-498755!84889609!26387897!79754065!65280632!79650573!%
15641731!48573470!15776042!57770381!25653277!94540857!71090400!48287745!%
25303061!14936045!}%
\calx_lpairdef{calx_lLt51}{-508704!09514428!94449432!72322921!10562822!66063535!%
09160159!80806124!95011244!67561603!58330282!52236663!66886243!60758057!%
34584392!89869079!}%
```

```
\calx_lpairdef{calx_lLt52}{-518653!33126999!39084055!94677324!39869013!75089511!%
11857136!36988901!33870994!62760910!69151294!06741456!64758648!59601462!%
14679418!50776616!}%
\calx_lpairdef{calx_lLt53}{-528601!55707626!44703393!50114701!55630387!19685687!%
25578800!50557057!84966278!63124313!10680693!03734537!64349642!05982279!%
30648903!08123941!}%
\calx_lpairdef{calx_lLt54}{-538548!77236621!83399658!73816066!74280888!77235342!%
34574062!21163218!02468950!41180663!31099444!29434884!04927355!94369713!%
88291708!01700580!}%
\calx_lpairdef{calx_lLt55}{-548494!97694303!14711854!29186307!18637673!00905796!%
05120648!85591135!65689110!33838396!13883670!11187767!42048708!27778611!%
96916115!97918011!}%
\calx_lpairdef{calx_lLt56}{-558438!17060993!85392090!29575491!10315982!27121117!%
75627123!32229002!57345994!67337600!37438820!97353639!89732423!10768246!%
94203830!48452284!}%
\calx_lpairdef{calx_lLu-5100}{5100!13005442!18691369!25798304!28407466!59720605!%
10398581!56934406!17114698!24490037!55735647!44712033!71107581!74606828!%
42495774!03161169!}%
\calx_lpairdef{calx_lLu-5000}{5000!12500416!68229229!16927094!49453592!19999845!%
43399175!65250398!69664800!24170272!64816892!16689712!88251109!15433720!%
20118585!39054382!}%
\calx_lpairdef{calx_lLu-4900}{4900!12005392!17774590!08069011!14266647!04264023!%
91935611!52798808!92577480!46329302!64567924!69821812!55322584!78819228!%
31016520!54460521!}%
\calx_lpairdef{calx_lLu-4800}{4800!11520368!65327154!96283211!56144285!99206763!%
26830753!97377255!68364003!60275346!21940715!38801320!23555248!27910085!%
32245936!15615617!}%
\calx_lpairdef{calx_lLu-4700}{4700!11045346!08886632!78746467!49325413!63233429!%
65534594!93823434!55221984!30613136!78957246!00330863!11352129!50210663!%
76238381!81495733!}%
\calx_lpairdef{calx_lLu-4600}{4600!10580324!46452738!52750764!50781713!70570236!%
45347529!38165156!43056153!52260319!40132188!77839342!29081784!10940776!%
52291025!83258240!}%
\calx_lpairdef{calx_lLu-4500}{4500!10125303!76025193!15700901!43209524!21181431!%
70889180!37999502!19370650!32023574!07515510!78085486!84107389!08505702!%
73192509!58049490!}%
\calx_lpairdef{calx_lLu-4400}{4400!09680283!95603723!65112089!79222050!96255123!%
85388018!15268998!32620598!83327874!63012880!40830224!86750471!65976122!%
32725744!92362713!}%
\calx_lpairdef{calx_lLu-4300}{4300!09245265!03188062!98607553!26741912!39023342!%
04911116!15374049!42521427!11789287!39007805!34524165!79011600!41564381!%
17333285!15867599!}%
\calx_lpairdef{calx_lLu-4200}{4200!08820246!96777950!13916127!15594008!40686969!%
40163798!05880025!55877122!36450432!45324797!86788830!57268502!01695196!%
54326298!93127479!}%
\calx_lpairdef{calx_lLu-4100}{4100!08405229!74373130!08869857!85298706!11221229!%
11965688!69414066!19090823!91316685!40189646!14934408!45527875!43505135!%
61020102!77290190!}%
\calx_lpairdef{calx_lLu-4000}{4000!08000213!33973353!81401602!34065335!14842445!%
43952825!99134192!14339716!63045344!62697413!12872087!76876188!80017098!%
02844059!09060423!}%
\calx_lpairdef{calx_lLu-3900}{3900!07605197!73578378!29542627!68985985!49921842!%
```

```
89465056!21788771!48335804!62130552!76032495!29963406!65673176!29466153!%
80492490!81642636!}%
\calx_lpairdef{calx_lLu-3800}{3800!07220182!91187966!51420210!57429600!53137189!%
18953957!33228517!56725220!66002179!38588023!12925201!77999426!78250648!%
03013617!50873644!}%
\calx_lpairdef{calx_lLu-3700}{3700!06845168!84801887!45255236!79636358!07658128!%
59589051!64610456!02674764!15847952!54226203!05042739!72238271!06509561!%
23268773!76469883!}%
\calx_lpairdef{calx_lLu-3600}{3600!06480155!52419916!09359800!82512332!35166094!%
40049115!34731825!97268889!87190475!23249344!12337821!91750931!15860093!%
10956595!32393798!}%
\calx_lpairdef{calx_lLu-3500}{3500!06125142!92041833!42134805!34624429!51514730!%
50761005!15203518!67183390!99957571!02195407!01826201!26781641!33714984!%
49038022!22031007!}%
\calx_lpairdef{calx_lLu-3400}{3400!05780131!01667426!42067560!82395589!65841793!%
83090637!70732940!19815969!24732638!30904611!07455141!15106431!50232686!%
02420995!60873763!}%
\calx_lpairdef{calx_lLu-3300}{3300!05445119!79296488!07729385!07500248!02948550!%
60199611!66814345!48589117!21124756!49758220!52067555!22667238!01374295!%
45785201!63497418!}%
\calx_lpairdef{calx_lLu-3200}{3200!05120109!22928817!37773202!85460048!28767721!%
17456491!91763659!25230539!16188168!91838508!46007499!27138146!97770645!%
21360743!53179630!}%
\calx_lpairdef{calx_lLu-3100}{3100!04805099!30564219!30931145!45439800!58746069!%
31434016!49391201!27931443!93892739!80596078!35782321!32357363!67113310!%
96898417!73214943!}%
\calx_lpairdef{calx_lLu-3000}{3000!04500090!00202504!86012150!31243677!28972773!%
33632475!72749894!23485700!22709856!66881480!12899213!13488406!13335896!%
91799216!14250705!}%
\calx_lpairdef{calx_lLu-2900}{2900!04205081!29843491!01899560!63511639!09889757!%
78145286!18362474!49510363!90283205!57523571!43019398!71795160!44134752!%
73545288!06043005!}%
\calx_lpairdef{calx_lLu-2800}{2800!03920073!17487000!77548725!03116084!42425205!%
61525375!54116671!83845944!91424852!02369622!04013198!26845952!71220029!%
25976371!56437759!}%
\calx_lpairdef{calx_lLu-2700}{2700!03645065!61132863!11984597!15758714!76396512!%
18120444!82583692!98889224!79215866!60891219!52522232!48956286!67471771!%
79771311!88959262!}%
\calx_lpairdef{calx_lLu-2600}{2600!03380058!58780913!04299335!37767608!91033983!%
35121517!74787756!77961616!15379452!88195574!13609938!41849842!78872751!%
08269315!84187312!}%
\calx_lpairdef{calx_lLu-2500}{2500!03125052!08430991!53649902!43094498!77481621!%
48512459!57321705!00215678!52118758!85172012!57418254!94765583!60516679!%
75773780!18804078!}%
\calx_lpairdef{calx_lLu-2400}{2400!02880046!08082945!59255665!11512239!63136383!%
94018386!28018372!83630904!65627014!14849142!96608763!01514875!55046599!%
78136175!09018992!}%
\calx_lpairdef{calx_lLu-2300}{2300!02645040!55736628!20395993!98012467!57692339!%
96028125!01965666!78123124!46243341!57084582!78696571!99926184!99281518!%
30772861!42331849!}%
\calx_lpairdef{calx_lLu-2200}{2200!02420035!49391898!36407863!03403437!00761192!%
82310165!40275087!90572331!67828420!56186460!55227354!51782951!50055804!%
```

```
18433831!49814072!}%
\calx_lpairdef{calx_lLu-2100}{2100!02205030!87048621!06683449!46108030!90945675!%
13152894!18422377!10580286!08415852!15074848!90583691!59983652!76791134!%
55793632!34295911!}%
\calx_lpairdef{calx_lLu-2000}{2000!02000026!66706667!30667733!35161936!76247366!%
10338369!78882371!81886372!52534463!48747004!32211487!87821185!13646769!%
87618367!33073605!}%
\calx_lpairdef{calx_lLu-1900}{1900!01805022!86365914!07856097!44411980!85695520!%
64104504!13849077!43376804!26630485!96428492!67696578!99041631!96610589!%
52699877!87750779!}%
\calx_lpairdef{calx_lLu-1800}{1800!01620019!44026244!37791926!87914613!82088540!%
84963313!67701101!21119941!16952594!36785112!49550642!33422745!03169251!%
30066109!95253189!}%
\calx_lpairdef{calx_lLu-1700}{1700!01445016!37687547!20064208!96534540!15744761!%
51922915!65140404!66201373!37386546!78616220!52863463!76675113!16995365!%
76367149!19791869!}%
\calx_lpairdef{calx_lLu-1600}{1600!01280013!65349717!54305132!95743484!59164261!%
89308215!89160938!21346479!79953071!11910241!63453639!05225037!35039224!%
07482944!70844425!}%
\calx_lpairdef{calx_lLu-1500}{1500!01125011!25012656!40187689!84619088!02508456!%
80989796!82718969!63040110!44540981!05124403!61825214!88813416!44745393!%
82260468!94352242!}%
\calx_lpairdef{calx_lLu-1400}{1400!00980009!14676270!77423272!16043925!89809261!%
03412405!58668349!27248667!11143850!18443032!66271178!81479808!96572838!%
98580625!23435671!}%
\calx_lpairdef{calx_lLu-1300}{1300!00845007!32340473!65759273!78104641!75824661!%
27363698!03644819!69541978!60046932!49542078!12713098!06232700!12115854!%
75292272!98446152!}%
\calx_lpairdef{calx_lLu-1200}{1200!00720005!76005184!04976689!76691188!83462571!%
02940551!82550728!19406221!08230462!80890497!66168543!51401551!03668884!%
35252219!96257744!}%
\calx_lpairdef{calx_lLu-1100}{1100!00605004!43670326!94887716!19296172!41699884!%
02654356!20485797!90147331!24409157!23097982!20809623!04323624!43844477!%
16648444!23272745!}%
\calx_lpairdef{calx_lLu-1000}{1000!00500003!33335833!35333350!00014285!83928682!%
54068254!87735321!07551344!69426136!14863667!43769358!61920829!41735142!%
47258641!56167620!}%
\calx_lpairdef{calx_lLu-900}{900!00405002!43001640!26180988!85741832!86666597!%
45778396!70072328!51094132!62860712!63208696!37504185!02204651!13277243!%
70100975!90861354!}%
\calx_lpairdef{calx_lLu-800}{800!00320001!70667690!67322031!03576329!28573357!%
38937874!00426739!50757702!64859456!60770107!39250061!32305467!74089803!%
79928646!30476955!}%
\calx_lpairdef{calx_lLu-700}{700!00245001!14333933!58669475!29416176!49720604!%
60876235!88157692!41195527!88005344!09792970!04330774!69805874!96785490!%
31043273!65311981!}%
\calx_lpairdef{calx_lLu-600}{600!00180000!72000324!00155520!77760399!91067095!%
97689290!37934388!74802130!07696533!57742775!23200471!92750667!51656847!%
11965418!00327297!}%
\calx_lpairdef{calx_lLu-500}{500!00125000!41666822!91729166!92708444!94096447!%
38963391!30748754!48736199!37506307!64384094!61516092!79383865!19079005!%
38658963!55571249!}%
```

```
\calx_lpairdef{calx_lLu-400}{400!00080000!21333397!33353813!40160023!40579620!%
60055578!73976638!41444282!54174886!12344131!15533192!08108349!51944831!%
64112451!28337321!}%
\calx_lpairdef{calx_lLu-300}{300!00045000!09000020!25004860!01215003!12429391!%
55575843!44763446!75745585!57393927!37095315!21565352!73392677!48173349!%
90866655!62230970!}%
\calx_lpairdef{calx_lLu-200}{200!00020000!02666670!66667306!66773333!51619079!%
61910450!80389081!22690096!02371817!81044648!33000077!80369192!81924379!%
55038641!17400254!}%
\calx_lpairdef{calx_lLu-100}{100!00005000!00333333!58333353!33335000!00142857!%
26785725!39683539!68344877!42821075!51338265!62393229!05572905!20525781!%
96366122!06008490!}%
\calx_lpairdef{calx_lLu100}{-100!00004999!99666666!91666646!66668333!33190476!%
31547607!93651793!64988456!07178924!48663162!94749628!21212809!08046757!%
71887846!29388334!}%
\calx_lpairdef{calx_lLu200}{-200!00019999!97333337!33332693!33439999!81714317!%
71422882!54992252!10643919!97628182!42361065!95579542!76773667!23632445!%
84748756!10853752!}%
\calx_lpairdef{calx_lLu300}{-300!00044999!91000020!24995140!01214996!87572248!%
69424157!73334553!24342987!92606140!95717541!98101201!67683055!76179510!%
72097498!76625740!}%
\calx_lpairdef{calx_lLu400}{-400!00079999!78666730!66646186!73493309!92770096!%
73277775!56508694!94685253!45828948!66878731!35623434!21226628!79350236!%
12447153!31274181!}%
\calx_lpairdef{calx_lLu500}{-500!00124999!58333489!58270833!59374888!39334542!%
19370137!33834579!25287238!12580885!43651810!40541664!32644517!12663187!%
44187855!37099000!}%
\calx_lpairdef{calx_lLu600}{-600!00179999!28000323!99844480!77759600!09352808!%
02311918!94417614!87994925!93422954!48112179!72680828!49979009!78810739!%
89195702!34902476!}%
\calx_lpairdef{calx_lLu700}{-700!00244998!85667266!91330528!62747156!85053928!%
97462746!95673663!99019005!71683556!94318571!11990606!36178547!05727887!%
10184475!83515116!}%
\calx_lpairdef{calx_lLu800}{-800!00319998!29334357!32677977!70237004!08954279!%
94416934!16554708!35821753!97969779!26873164!89983629!41963696!76274240!%
55717926!48548661!}%
\calx_lpairdef{calx_lLu900}{-900!00404997!57001640!23819028!85728167!24095082!%
79291338!98730142!20495284!13950608!17623293!14690400!85608294!24363930!%
53687960!24069961!}%
\calx_lpairdef{calx_lLu1000}{-1000!00499996!66669166!64666683!33319047!74404650!%
79465078!45599678!92448669!59145292!54779189!70627466!77762710!26518834!%
87229093!00939162!}%
\calx_lpairdef{calx_lLu1100}{-1100!00604995!56336993!55112342!85907161!45223169!%
56197725!62054766!14582176!50588465!96112037!27040379!95793334!20494497!%
64999246!34306910!}%
\calx_lpairdef{calx_lLu1200}{-1200!00719994!24005183!95023409!76588812!24032852!%
98297795!45912131!97335338!33461890!20020540!27240585!20011701!97169779!%
28234970!32780817!}%
\calx_lpairdef{calx_lLu1300}{-1300!00844992!67673806!84240887!11258693!61441352!%
33726805!13425343!77978997!38187157!24070112!04822284!42431479!28154485!%
58258161!95336837!}%
\calx_lpairdef{calx_lLu1400}{-1400!00979990!85342937!22576978!82409411!12471336!%
```

```
35706020!84711473!91482136!31810827!86878640!07331196!99475059!61818465!%
08874389!12864819!}%
\calx_lpairdef{calx_lLu1500}{-1500!01124988!75012656!09812689!84130918!38214199!%
55543210!98747274!26612497!47259463!77707471!23639179!30579781!13152016!%
45166198!28245213!}%
\calx_lpairdef{calx_lLu1600}{-1600!01279986!34683050!45695426!28309859!47910895!%
66015350!00193520!08115590!14247701!29285570!03519128!62558691!47693580!%
50569664!47267542!}%
\calx_lpairdef{calx_lLu1700}{-1700!01444983!62354213!29936595!62028810!61527932!%
46730295!70061965!73360974!10136338!63486496!82345750!16902240!48983959!%
44618228!09355174!}%
\calx_lpairdef{calx_lLu1800}{-1800!01619980!56026243!62209206!86165413!72901603!%
86446030!87474951!09171299!59728667!64091659!10839664!70370352!13484619!%
04875984!35168384!}%
\calx_lpairdef{calx_lLu1900}{-1900!01804977!13699246!42145470!75191394!93528574!%
16850154!38775114!09915033!04598589!32654296!83226402!12181421!35158539!%
90196824!05012387!}%
\calx_lpairdef{calx_lLu2000}{-2000!01999973!33373332!69334399!98171460!57085969!%
27794963!61277628!18347684!61759442!79824715!52042490!86151640!64165806!%
18397338!62071812!}%
\calx_lpairdef{calx_lLu2100}{-2100!02204969!13048619!43319409!40962063!64769168!%
45444725!50239335!41780904!54923765!79618654!55448198!20428751!58990711!%
74808263!64931610!}%
\calx_lpairdef{calx_lLu2200}{-2200!02419964!50725229!63595916!29610033!51540529!%
82221623!97331249!82158035!93258190!32177011!29387843!84659190!03379759!%
26587616!69415317!}%
\calx_lpairdef{calx_lLu2300}{-2300!02644959!44403292!29608940!54951061!53387321!%
90835432!95190040!62203048!51326726!77094337!35918358!89589274!86426591!%
11865202!23363042!}%
\calx_lpairdef{calx_lLu2400}{-2400!02879953!92082942!40750704!98408035!55692172!%
74049233!63600872!92050294!21998280!00470150!16783845!75625729!92352206!%
25302480!93069732!}%
\calx_lpairdef{calx_lLu2500}{-2500!03124947!91764320!96358235!58989216!02824387!%
17169507!56602703!01004602!26523981!46193747!06699515!49685036!60853203!%
40794443!56468997!}%
\calx_lpairdef{calx_lLu2600}{-2600!03379941!41447574!95710961!81486246!49065522!%
21553730!24345837!74867790!52355615!70427181!25354289!84769471!23326555!%
29919859!70734334!}%
\calx_lpairdef{calx_lLu2700}{-2700!03644934!39132857!38028316!85871991!30987649!%
24702199!56872032!66241912!87945676!20460910!50247010!56742353!93489901!%
32600800!99066830!}%
\calx_lpairdef{calx_lLu2800}{-2800!03919926!82820327!22467337!97898193!41407770!%
95743295!57095103!57031236!79358506!41108059!12159264!00159200!97650196!%
19490506!02949955!}%
\calx_lpairdef{calx_lLu2900}{-2900!04204918!70510149!48120266!80892944!85046899!%
94332719!05064164!60576128!58213439!11212132!75446839!35423754!86989764!%
58475290!77178382!}%
\calx_lpairdef{calx_lLu3000}{-3000!04499910!00202495!14012149!68757962!96027344!%
76167533!12985106!44842428!39813840!60334701!36707083!28058303!83310134!%
54385120!54991741!}%
\calx_lpairdef{calx_lLu3100}{-3100!04804900!69897541!19098438!00165664!97346788!%
19465069!58315439!44961697!54068568!08537486!60993470!47769671!06547933!%
```

```
29093061!06386046!}%
\calx_lpairdef{calx_lLu3200}{-3200!05119890!77595470!62262588!53956033!82472781!%
33874997!48353516!77899902!80600708!17518124!08497183!21612910!38713776!%
25802475!87611100!}%
\calx_lpairdef{calx_lLu3300}{-3300!05444880!21296472!42313663!85733267!99206315!%
96380121!41899515!36384331!07654353!15219054!45857176!51302881!07057038!%
14649415!01349552!}%
\calx_lpairdef{calx_lLu3400}{-3400!05779868!99000741!57983932!65662208!15968176!%
37797816!59520203!23134537!20119536!12697617!60110549!28783560!68419162!%
14778284!28135322!}%
\calx_lpairdef{calx_lLu3500}{-3500!06124857!08708479!07926470!17464533!50666810!%
78519454!81414000!66953636!04420424!67084829!96643924!98751382!51122185!%
47507912!03822940!}%
\calx_lpairdef{calx_lLu3600}{-3600!06479844!48419891!90712758!58614720!42311500!%
83119764!88522413!33611386!33218724!43095557!56523058!40815957!73658194!%
95839217!81101975!}%
\calx_lpairdef{calx_lLu3700}{-3700!06844831!16135193!04830287!41735756!45539646!%
70431839!76007566!58101778!88299636!56867695!66743361!87136227!88405207!%
79249927!19437550!}%
\calx_lpairdef{calx_lLu3800}{-3800!07219817!09854601!48680153!97194602!28232023!%
78616486!86116015!29239577!68385790!31863649!12777708!30064790!19970247!%
08389847!16809843!}%
\calx_lpairdef{calx_lLu3900}{-3900!07604802!27578342!20574663!76897394!52394905!%
43656853!97342681!38070651!62087238!33491702!82495219!19223915!72542626!%
67955122!52670513!}%
\calx_lpairdef{calx_lLu4000}{-4000!07999786!67306646!18734930!99284382!18492985!%
04580786!96225846!20452517!30968425!27943275!41805568!20753069!09654435!%
30386890!45392240!}%
\calx_lpairdef{calx_lLu4100}{-4100!08404770!27039750!41288478!95524589!53422068!%
99554223!74538488!60401225!40179185!11015801!61032838!73090375!77930823!%
69404791!40010380!}%
\calx_lpairdef{calx_lLu4200}{-4200!08819753!04777897!86266840!56910198!22315550!%
63799137!65550528!10029455!98791415!13775989!89995073!35513666!92369499!%
02766319!91765737!}%
\calx_lpairdef{calx_lLu4300}{-4300!09244734!98521337!51603158!83450641!44383713!%
83069148!85842051!56292372!62150376!53939666!71391270!64641336!01882048!%
36127549!82539067!}%
\calx_lpairdef{calx_lLu4400}{-4400!09679716!06270324!35129787!33666402!92989952!%
95164961!11234441!92927988!09885350!17176393!90993126!72171505!68976160!%
71089589!03653678!}%
\calx_lpairdef{calx_lLu4500}{-4500!10124696!26025119!34575890!75582513!60173034!%
66690289!14123288!82882850!91425845!57613290!34930439!47922907!23687734!%
31313934!93716835!}%
\calx_lpairdef{calx_lLu4600}{-4600!10579675!55785989!47565045!38921738!65829565!%
02936955!52156570!83344510!03615240!72605519!62417973!42989480!09245986!%
65186531!94307151!}%
\calx_lpairdef{calx_lLu4700}{-4700!11044653!93553207!71612839!68497447!91775863!%
75445391!52080113!69586772!58995595!15309567!10384865!29564302!29392566!%
17285682!79245575!}%
\calx_lpairdef{calx_lLu4800}{-4800!11519631!37327053!04124474!78806162!20913485!%
74413910!04909733!48530811!36967630!30797771!18840195!65301077!66411119!%
24144835!06751482!}%
```

```
\calx_lpairdef{calx_lLu4900}{-4900!12004607!85107810!42392365!09819768!61727668!%
11726867!06589482!47520754!62444668!82130496!06093773!02782405!36002709!%
27427772!04172801!}%
\calx_lpairdef{calx_lLu5000}{-5000!12499583!34895770!83593738!83977397!38353019!%
14938226!69125473!23415747!63601812!14807063!30238573!94514879!55879584!%
18153093!93366937!}%
\calx_lpairdef{calx_lLu5100}{-5100!13004557!84691231!24788238!64376953!26445803!%
63083131!17976037!02539314!37766571!58799779!64229096!20416974!57931196!%
01965236!41122723!}%
\calx_lpairdef{calx_lLv-51}{51!00001300!50044217!01691300!94005079!52715915!%
51582617!24236246!74584275!16289244!82079549!59005935!53687449!52795747!%
43134993!57035449!}%
\calx_lpairdef{calx_lLv-50}{50!00001250!00041666!68229167!29166692!70834449!%
40525018!62289187!48449945!18287568!41668124!67070556!26326351!57654522!%
85062621!35452203!}%
\calx_lpairdef{calx_lLv-49}{49!00001200!50039216!34774534!14828406!20215502!%
39051948!18130549!36504573!95256679!16117477!45773743!77310202!70029491!%
09030805!22823594!}%
\calx_lpairdef{calx_lLv-48}{48!00001152!00036864!01327104!50960813!98432582!%
66941262!97270813!31975293!86806511!10085234!75874526!56337476!10633064!%
88536624!88754870!}%
\calx_lpairdef{calx_lLv-47}{47!00001104!50034607!67886587!37535686!03203278!%
58094592!16838103!71634382!78937949!35748025!75571656!99618109!51095304!%
65336283!41559047!}%
\calx_lpairdef{calx_lLv-46}{46!00001058!00032445!34452697!74525944!32383438!%
59690376!09696505!30336374!61919476!39996745!79058995!13816407!88155796!%
75797285!43505587!}%
\calx_lpairdef{calx_lLv-45}{45!00001012!50030375!01025156!61905638!83961471!%
31371465!34371727!59218053!56010982!37017590!91562318!30152274!40690050!%
25505980!66759942!}%
\calx_lpairdef{calx_lLv-44}{44!00000968!00028394!67603690!99649923!56052765!%
44466559!90544716!22063507!18244630!42893156!83349332!01728687!97942495!%
31578629!11645079!}%
\calx_lpairdef{calx_lLv-43}{43!00000924!50026502!34188033!87735032!46894229!%
81244768!39878850!86231457!44395216!57940801!77968146!73983086!66475989!%
28265000!52894989!}%
\calx_lpairdef{calx_lLv-42}{42!00000882!00024696!00777924!26138255!54838953!%
34202567!31951409!26396812!64321052!67034482!82467509!64147094!47613915!%
19565105!95521612!}%
\calx_lpairdef{calx_lLv-41}{41!00000840!50022973!67373107!14837914!78350985!%
05382440!35065017!01172786!96393912!95056611!48268841!16541085!77784625!%
04883859!52522283!}%
\calx_lpairdef{calx_lLv-40}{40!00000800!00021333!33973333!53813340!16000234!%
05722477!71719842!64454017!78125900!27194541!73160459!85173155!52589625!%
21216035!50991086!}%
\calx_lpairdef{calx_lLv-39}{39!00000760!50019773!00578360!43044845!66457489!%
54436215!58532337!75054874!25705089!41699688!13730779!70284467!08518610!%
87845602!93435417!}%
\calx_lpairdef{calx_lLv-38}{38!00000722!00018290!67187950!82513705!28489560!%
78421995!50391361!69910593!95333373!62004742!93226891!22226279!18480906!%
16330795!89746155!}%
\calx_lpairdef{calx_lLv-37}{37!00000684!50016884!33801873!72202129!00954537!%
```

```
11701123!88647570!67762007!24381957!68155970!85626621!26749490!26522224!%
96809909!57018641!}%
\calx_lpairdef{calx_lLv-36}{36!00000648!00015552!00419904!12093238!82797167!%
94884111!53136990!71857412!20805288!63144244!99403412!02278400!11048695!%
41568626!43636578!}%
\calx_lpairdef{calx_lLv-35}{35!00000612!50014291!67041823!02171044!73044362!%
74664273!17844734!41777645!45345774!13062665!24167339!12449500!16904852!%
22932790!59173849!}%
\calx_lpairdef{calx_lLv-34}{34!00000578!00013101!33667417!42420420!70800811!%
03337967!10019861!06022556!93182291!12603777!13488137!65240044!51051033!%
02586098!11390282!}%
\calx_lpairdef{calx_lLv-33}{33!00000544!50011979!00296480!32827080!75244722!%
38350754!72557419!88488939!30188114!07130270!94352704!52309944!06109418!%
02289572!03589190!}%
\calx_lpairdef{calx_lLv-32}{32!00000512!00010922!66928810!73377554!85623686!%
41868760!29468756!23421484!74231374!63675462!57602308!79563420!98414116!%
32689714!18425984!}%
\calx_lpairdef{calx_lLv-31}{31!00000480!50009930!33564213!64059165!01250652!%
80374510!54266200!34829544!34336384!56390897!77089386!58244198!62716575!%
18551577!83972458!}%
\calx_lpairdef{calx_lLv-30}{30!00000450!00009000!00202500!04860001!21500031!%
24286534!41093298!57733347!30389991!44166323!93896623!20059195!87044067!%
17579458!65270786!}%
\calx_lpairdef{calx_lLv-29}{29!00000420!50008129!66843486!95768897!45803911!%
47602002!78436786!79933899!13786474!92781470!74345877!59033394!93435816!%
30300575!65061901!}%
\calx_lpairdef{calx_lLv-28}{28!00000392!00007317!33486997!36775407!73648403!%
27561688!25261545!84291019!95320209!50760188!58049738!56479075!48065392!%
55741038!02675081!}%
\calx_lpairdef{calx_lLv-27}{27!00000364!50006561!00132860!27869782!04570096!%
44336524!89414818!72743905!61120300!43207318!08983942!90442674!03475418!%
55095156!02404114!}%
\calx_lpairdef{calx_lLv-26}{26!00000338!00005858!66780910!69042942!38152640!%
80735048!08151009!54518196!20838523!13810195!65162776!79562370!86311673!%
49391172!20857707!}%
\calx_lpairdef{calx_lLv-25}{25!00000312!50005208!33430989!60286458!74023446!%
21930994!30633423!82132817!85014044!30744119!30195539!60262428!61539695!%
31611220!43847766!}%
\calx_lpairdef{calx_lLv-24}{24!00000288!00004608!00082944!01592525!11850502!%
55210341!02274350!19948825!10908455!82669774!64456100!97078730!86436886!%
10415637!66421357!}%
\calx_lpairdef{calx_lLv-23}{23!00000264!50004055!66736626!92953935!51339319!%
69737066!50779923!31091112!88495492!27957107!78844709!75798052!94056499!%
67757656!72122497!}%
\calx_lpairdef{calx_lLv-22}{22!00000242!00003549!33391897!34364059!92229987!%
56336909!73771247!69622187!61064309!21692177!76482883!82906478!26202750!%
48697368!70441522!}%
\calx_lpairdef{calx_lLv-21}{21!00000220!50003087!00048620!25816820!34294356!%
07298410!27858302!65855188!88416254!25270059!35682906!78312092!76014740!%
30475516!73991849!}%
\calx_lpairdef{calx_lLv-20}{20!00000200!00002666!66706666!67306666!77333335!%
16190508!19048187!93661033!65265546!93168023!71817705!12055274!50843034!%
```

```
83970985!08491804!}%
\calx_lpairdef{calx_lLv-19}{19!00000180!50002286!33365913!58828553!21174314!%
77695983!94374310!09551916!03557077!17920313!68966161!30317246!82856156!%
54887174!59033698!}%
\calx_lpairdef{calx_lLv-18}{18!00000162!00001944!00026244!00377913!65668704!%
87460018!34638149!54210436!75352516!80841359!09058593!88060222!55722002!%
36578660!83852421!}%
\calx_lpairdef{calx_lLv-17}{17!00000144!50001637!66687546!91950638!10689595!%
41953152!48160288!07980356!93071994!82501941!23203349!94715318!03395502!%
84261594!96918200!}%
\calx_lpairdef{calx_lLv-16}{16!00000128!00001365!33349717!33543048!56129536!%
38347927!65442416!92641371!76575595!89557503!65232042!96221383!12870931!%
52978830!60513019!}%
\calx_lpairdef{calx_lLv-15}{15!00000112!50001125!00012656!25151875!01898437!%
74408485!34647085!12556380!22189783!05835597!61256357!02309307!59776883!%
41185887!42554243!}%
\calx_lpairdef{calx_lLv-14}{14!00000098!00000914!66676270!66774231!47921589!%
48392407!17806988!29005487!47691401!24557618!42803440!73323442!67486811!%
83800882!48412918!}%
\calx_lpairdef{calx_lLv-13}{13!00000084!50000732!33340473!58407591!94137801!%
58964074!87680637!62206417!39819068!10516997!08048021!68418298!52871573!%
24689173!28865103!}%
\calx_lpairdef{calx_lLv-12}{12!00000072!00000576!00005184!00049766!40497664!%
05118830!25176289!16166132!77451383!57651518!60401042!42289900!44852011!%
98480477!34876409!}%
\calx_lpairdef{calx_lLv-11}{11!00000060!50000443!66670326!91698876!86961926!%
86117215!17271053!22113261!44536370!26792295!72232552!52640074!88211718!%
35678406!53672972!}%
\calx_lpairdef{calx_lLv-10}{10!00000050!00000333!33335833!33353333!33500000!%
01428571!55357143!96825406!82539773!44878178!21075513!37622766!12943279!%
00577894!91612967!}%
\calx_lpairdef{calx_lLv-9}{9!00000040!50000243!00001640!25011809!80088573!50683281!%
33952269!12689581!62964182!91395264!74757700!99260877!36361566!17240504!%
92552408!}%
\calx_lpairdef{calx_lLv-8}{8!00000032!00000170!66667690!66673220!26710357!33632926!%
49716199!76817843!86739269!57411138!50144407!74337587!50309824!92825111!%
85029121!}%
\calx_lpairdef{calx_lLv-7}{7!00000024!50000114!33333933!58336694!73352941!50117649!%
00720600!16983734!39358637!80868099!28188149!36082093!06435330!46817626!%
14583377!}%
\calx_lpairdef{calx_lLv-6}{6!00000018!00000072!00000324!00001555!20007776!00039990!%
85924237!72548315!48903760!78695838!07372564!02071423!83854192!44168900!%
09529023!}%
\calx_lpairdef{calx_lLv-5}{5!00000012!50000041!66666822!91667291!66669270!83344494!%
04810732!88907490!08913070!48089714!30878963!55624631!46865633!03097822!%
29328805!}%
\calx_lpairdef{calx_lLv-4}{4!00000008!00000021!33333397!33333538!13334016!00002340!%
57151049!14314841!39787397!28635268!63158563!14541729!87894130!57885377!%
25360763!}%
\calx_lpairdef{calx_lLv-3}{3!00000004!50000009!00000020!25000048!60000121!50000312!%
42857962!98216472!71434476!32873247!13031299!73823938!93548433!85921628!%
79193916!}%
```

```
\calx_lpairdef{calx_lLv-2}{2!00000002!00000002!66666670!66666673!06666677!33333351!%
61904793!90476247!36508038!90793836!97547238!88023718!17694587!97915382!%
44657220!}%
\calx_lpairdef{calx_lLv-1}{1!00000000!50000000!33333333!58333333!53333333!50000000!%
14285714!41071428!68253968!35396825!48773448!85678210!75513375!58480408!%
54707514!}%
\calx_lpairdef{calx_lLv1}{-1!00000000!49999999!66666666!91666666!46666666!83333333!%
19047619!17261904!65079365!17936507!84559884!64321789!24486624!55805305!%
73863913!}%
\calx_lpairdef{calx_lLv2}{-2!00000001!99999997!33333337!33333326!93333343!99999981!%
71428603!42857085!96825499!22539496!35786777!11976281!82307752!59227474!%
69636685!}%
\calx_lpairdef{calx_lLv3}{-3!00000004!49999991!00000020!24999951!40000121!49999687!%
57143677!26783527!28577333!47126752!87057273!76176061!07134847!42649799!%
83329780!}%
\calx_lpairdef{calx_lLv4}{-4!00000007!99999978!66666730!66666461!86667349!33330992!%
76198668!19018491!93755651!24698064!72970972!85458270!50453791!70686056!%
68653006!}%
\calx_lpairdef{calx_lLv5}{-5!00000012!49999958!33333489!58332708!33335937!49988839!%
28620256!69425843!26373387!85243619!43144473!94375377!25065170!54045225!%
58443236!}%
\calx_lpairdef{calx_lLv6}{-6!00000017!99999928!00000323!99998444!80007775!99960009!%
14495666!27451684!63189474!41304165!55424491!97928688!11026392!70120340!%
57780837!}%
\calx_lpairdef{calx_lLv7}{-7!00000024!49999885!66667266!91663305!26686274!83215684!%
34053933!41349599!50469745!32465257!12026384!13918875!82575076!53223915!%
01737822!}%
\calx_lpairdef{calx_lLv8}{-8!00000031!99999829!33334357!33326779!73377023!99700406!%
87811437!56515491!61342428!55922309!36435048!25668695!42048904!21812446!%
15631138!}%
\calx_lpairdef{calx_lLv9}{-9!00000040!49999757!00001640!24988190!20088573!49316718!%
76809411!12310425!34392697!28605205!96831712!50771803!76846285!42218627!%
45339787!}%
\calx_lpairdef{calx_lLv10}{-10!00000049!99999666!66669166!66646666!66833333!%
31904762!02976189!36507946!50793559!88456821!78924486!62520091!01342435!%
40493533!65529900!}%
\calx_lpairdef{calx_lLv11}{-11!00000060!49999556!33336993!58301123!13628593!%
47216118!69652000!36220123!76281883!26546268!32496900!97902424!88014459!%
50330184!36416466!}%
\calx_lpairdef{calx_lLv12}{-12!00000071!99999424!00005183!99950233!60497663!%
94881170!82319134!83833991!06021461!22363341!56340333!59544268!78654970!%
89767703!41289152!}%
\calx_lpairdef{calx_lLv13}{-13!00000084!49999267!66673806!91592408!07471134!%
74369260!49585375!96127191!65212635!02855166!39472391!87206524!87942619!%
21367302!47605054!}%
\calx_lpairdef{calx_lLv14}{-14!00000097!99999085!33342937!33225768!54588255!%
84940929!84473609!04328424!36572927!28870204!75925775!42550852!79021503!%
38640789!77763216!}%
\calx_lpairdef{calx_lLv15}{-15!00000112!49998875!00012656!24848125!01898437!%
25591521!06075571!12444773!07888341!94380646!28392080!89394872!43850430!%
83610271!50144934!}%
\calx_lpairdef{calx_lLv16}{-16!00000127!99998634!66683050!66456951!49462868!%
```

```
94985416!41632740!40694160!59083292!64244954!64219734!06718036!92547575!%
62594118!67519117!}%
\calx_lpairdef{calx_lLv17}{-17!00000144!49998362!33354213!58049361!97356260!%
91380198!29112405!50357008!39041428!31802429!16334945!95824434!19376666!%
70799678!18739473!}%
\calx_lpairdef{calx_lLv18}{-18!00000161!99998056!00026243!99622086!45668703!%
12540009!20351994!45796704!18092808!01086693!21179107!47387445!41719661!%
91406638!48009090!}%
\calx_lpairdef{calx_lLv19}{-19!00000180!49997713!66699246!91171446!94507645!%
55637391!84849783!48793679!43027816!39101877!84211543!61120875!71845180!%
54022642!09054946!}%
\calx_lpairdef{calx_lLv20}{-20!00000199!99997333!33373333!32693333!43999998!%
17142889!14285145!39692779!68067786!46991976!28182318!28516154!06308014!%
30314729!22992335!}%
\calx_lpairdef{calx_lLv21}{-21!00000220!49996913!00048620!24183179!94294350!%
92701684!27856537!59178170!87779386!06989653!16214913!05859239!41361524!%
80387285!81183977!}%
\calx_lpairdef{calx_lLv22}{-22!00000241!99996450!66725230!65635940!45563313!%
76996560!78530469!63764265!56827309!13066159!95358741!05409626!63523540!%
67458769!57467250!}%
\calx_lpairdef{calx_lLv23}{-23!00000264!49995944!33403293!57046064!98005976!%
63596462!60299730!27325073!47080571!21900599!59825509!35396696!40127273!%
52404802!03040458!}%
\calx_lpairdef{calx_lLv24}{-24!00000287!99995392!00082943!98407475!51850489!%
44789934!16554193!80177981!65284619!98197471!08220295!53950016!52084253!%
36644473!06587450!}%
\calx_lpairdef{calx_lLv25}{-25!00000312!49994791!66764322!89713542!07356762!%
11402720!49672565!76391250!34647415!01930288!65703430!08170406!55771432!%
27416943!45344365!}%
\calx_lpairdef{calx_lLv26}{-26!00000337!99994141!33447577!30957058!64819284!%
52598807!31948467!79097471!31625486!98571398!78451840!77544348!13826432!%
36375974!30235592!}%
\calx_lpairdef{calx_lLv27}{-27!00000364!49993439!00132860!22130219!24570066!%
55664181!17969301!52667876!65298631!86950407!40517473!72398942!42144406!%
13036655!36008423!}%
\calx_lpairdef{calx_lLv28}{-28!00000391!99992682!66820330!63224593!86981698!%
05772589!58571371!49634706!91352212!49603586!56821598!27977824!45509378!%
81692318!09045812!}%
\calx_lpairdef{calx_lLv29}{-29!00000420!49991870!33510153!54231104!52470528!%
85732581!16499786!79240848!66206904!50243167!93236546!74156141!95329904!%
76085596!20522238!}%
\calx_lpairdef{calx_lLv30}{-30!00000449!99991000!00202499!95140001!21499968!%
75715105!83906701!43447632!69610017!41568676!06110209!61226518!46908201!%
52063441!53687027!}%
\calx_lpairdef{calx_lLv31}{-31!00000480!49990069!66897546!85940837!94583907!%
52960955!01826493!25143045!56393122!09713741!87679720!40234688!31564831!%
12203909!98411402!}%
\calx_lpairdef{calx_lLv32}{-32!00000511!99989077!33595477!26622448!72290254!%
91467321!81771521!12163648!40470502!71193711!76889052!79894746!89740802!%
70400138!40444469!}%
\calx_lpairdef{calx_lLv33}{-33!00000544!49988021!00296480!17172923!55244600!%
61652761!29597140!39574218!66864803!52685587!31170736!57235461!61561472!%
```

```
56681723!04059221!}%
\calx_lpairdef{calx_lLv34}{-34!00000577!99986898!67000750!57579584!44133994!%
29999830!71789696!31439531!90963037!18097362!35401448!38060454!78072320!%
50154655!65479837!}%
\calx_lpairdef{calx_lLv35}{-35!00000612!49985708!33708489!47828961!39710845!%
58674689!84336255!22072782!58690740!52304848!18665234!73439599!58591307!%
45696020!04228025!}%
\calx_lpairdef{calx_lLv36}{-36!00000647!99984448!00419903!87906768!42796944!%
05122941!24339873!35454904!67207385!54157985!54287131!72313693!47791313!%
60151235!00203824!}%
\calx_lpairdef{calx_lLv37}{-37!00000684!49983115!67135206!77797879!54287599!%
21640990!64549243!00188494!83435055!16764038!87841335!96530504!14596629!%
53106282!19815915!}%
\calx_lpairdef{calx_lLv38}{-38!00000721!99981709!33854617!17486304!75155900!%
54922207!30976595!75979163!07597755!60891771!22792697!99973532!85444366!%
11703191!65929637!}%
\calx_lpairdef{calx_lLv39}{-39!00000760!49980227!00578360!06955166!06457097!%
45577164!43782782!66225937!91333437!14229737!82609107!35707153!27047005!%
46073642!10904160!}%
\calx_lpairdef{calx_lLv40}{-40!00000799!99978666!67306666!46186673!49333099!%
27627239!61613490!89850835!55207712!68165458!27223019!37112564!12852715!%
35934457!13036138!}%
\calx_lpairdef{calx_lLv41}{-41!00000840!49977026!34039773!35162101!05017095!%
27970855!29575596!84005864!98970276!64093889!13304018!43737372!99032141!%
52730081!46363975!}%
\calx_lpairdef{calx_lLv42}{-42!00000881!99975304!00777923!73861762!74838294!%
65821639!31047695!07763583!60015014!28747994!91572008!98959249!24055027!%
85076707!87577820!}%
\calx_lpairdef{calx_lLv43}{-43!00000924!49973497!67521366!62264988!60226786!%
52117785!43523883!15324840!51595632!54777193!66850091!86508556!84594396!%
47688897!17727767!}%
\calx_lpairdef{calx_lLv44}{-44!00000967!99971605!34270357!00350100!62718519!%
88901893!98694921!65664548!02773215!67258137!81038385!69333967!27792399!%
05702897!29949007!}%
\calx_lpairdef{calx_lLv45}{-45!00001012!49969625!01025155!88094388!83960403!%
68670572!46975929!33883204!27193291!83855000!65705817!20131670!32560562!%
81181014!85062048!}%
\calx_lpairdef{calx_lLv46}{-46!00001057!99967554!67786030!25474087!25715526!%
73693076!26694812!87838453!67970668!17172710!16193527!00956846!46660178!%
05456045!20327156!}%
\calx_lpairdef{calx_lLv47}{-47!00001104!49965392!34553253!12464349!89868497!%
75298269!38160670!91897215!01563330!30057155!86191387!73507200!51741047!%
02642369!19799611!}%
\calx_lpairdef{calx_lLv48}{-48!00001151!99963136!01327103!49039226!78430905!%
33129185!22836451!97874830!34904986!82153262!10338249!98659757!64139573!%
00175501!10192738!}%
\calx_lpairdef{calx_lLv49}{-49!00001200!49960783!68107866!35171639!93546897!%
94364467!47845185!81413291!97602247!39268156!88533225!96325278!15455979!%
96164639!41403919!}%
\calx_lpairdef{calx_lLv50}{-50!00001249!99958333!34895832!70833359!37498883!%
92905970!96044147!80195888!15049833!92706875!41648751!77245267!72974662!%
42146948!18292846!}%
```

```
\calx_lpairdef{calx_lLv51}{-51!00001300!49955783!01691299!55994979!12713351!%
48531802!61906355!88500572!49238515!86827743!22734939!74485657!02724801!%
63736203!80193639!}%
\fi
\ifnum\calx_ldigits=0
\def\calx_lLzl{2994!0456!8401!7991!4546!8436!4207!6011!0148!8628!7729!7603!%
3327!9009!6757!2609!6773!5248!0235!9972!0508!9598!2983!4196!7784!0422!8624!%
8633!4095!2546!}%
\fi
\ifnum\calx_ldigits>0
\@namedef{calx_lcdl0}{0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!}%
\@namedef{calx_lcdl1}{0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!5000!}%
\@namedef{calx_lcdl2}{3333!3333!3333!3333!3333!3333!3333!3333!3333!3333!3333!%
3333!3333!3333!3333!3333!3333!3333!3333!3333!3333!3333!3333!3333!3333!%
3333!3333!3333!3333!3333!3333!3333!3333!3333!}%
\@namedef{calx_lcdl3}{0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!2500!}%
\@namedef{calx_lcdl4}{0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!2000!}%
\@namedef{calx_lcdl5}{6666!6666!6666!6666!6666!6666!6666!6666!6666!6666!6666!%
6666!6666!6666!6666!6666!6666!6666!6666!6666!6666!6666!6666!6666!6666!%
6666!6666!6666!6666!6666!6666!6666!6666!1666!}%
\@namedef{calx_lcdl6}{5714!1428!2857!5714!1428!2857!5714!1428!2857!5714!1428!%
2857!5714!1428!2857!5714!1428!2857!5714!1428!2857!5714!1428!2857!5714!1428!%
2857!5714!1428!2857!5714!1428!2857!5714!1428!}%
\@namedef{calx_lcdl7}{0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!1250!}%
\@namedef{calx_lcdl8}{1111!1111!1111!1111!1111!1111!1111!1111!1111!1111!1111!%
1111!1111!1111!1111!1111!1111!1111!1111!1111!1111!1111!1111!1111!1111!%
1111!1111!1111!1111!1111!1111!1111!1111!1111!}%
\@namedef{calx_lcdl9}{0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!1000!}%
\@namedef{calx_lcdl10}{0909!0909!0909!0909!0909!0909!0909!0909!0909!0909!0909!%
0909!0909!0909!0909!0909!0909!0909!0909!0909!0909!0909!0909!0909!0909!%
0909!0909!0909!0909!0909!0909!0909!0909!0909!}%
\@namedef{calx_lcdl11}{3333!3333!3333!3333!3333!3333!3333!3333!3333!3333!3333!%
3333!3333!3333!3333!3333!3333!3333!3333!3333!3333!3333!3333!3333!3333!%
3333!3333!3333!3333!3333!3333!3333!3333!0833!}%
\@namedef{calx_lcdl12}{2307!0769!6923!2307!0769!6923!2307!0769!6923!2307!0769!%
6923!2307!0769!6923!2307!0769!6923!2307!0769!6923!2307!0769!6923!2307!0769!%
6923!2307!0769!6923!2307!0769!6923!2307!0769!}%
\@namedef{calx_lcdl13}{2857!5714!1428!2857!5714!1428!2857!5714!1428!2857!5714!%
1428!2857!5714!1428!2857!5714!1428!2857!5714!1428!2857!5714!1428!2857!5714!%
1428!2857!5714!1428!2857!5714!1428!2857!0714!}%
```

```
\@namedef{calx_lcdl14}{6666!6666!6666!6666!6666!6666!6666!6666!6666!6666!6666!%
6666!6666!6666!6666!6666!6666!6666!6666!6666!6666!6666!6666!6666!6666!6666!%
6666!6666!6666!6666!6666!6666!6666!6666!0666!}%
\@namedef{calx_lcdl15}{0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0625!}%
\@namedef{calx_lcdl16}{9411!2352!0588!7647!9411!2352!0588!7647!9411!2352!0588!%
7647!9411!2352!0588!7647!9411!2352!0588!7647!9411!2352!0588!7647!9411!2352!%
0588!7647!9411!2352!0588!7647!9411!2352!0588!}%
\@namedef{calx_lcdl17}{5555!5555!5555!5555!5555!5555!5555!5555!5555!5555!5555!%
5555!5555!5555!5555!5555!5555!5555!5555!5555!5555!5555!5555!5555!5555!5555!%
5555!5555!5555!5555!5555!5555!5555!5555!0555!}%
\@namedef{calx_lcdl18}{4736!5789!2631!2105!3684!8947!3157!0526!8421!4736!5789!%
2631!2105!3684!8947!3157!0526!8421!4736!5789!2631!2105!3684!8947!3157!0526!%
8421!4736!5789!2631!2105!3684!8947!3157!0526!}%
\@namedef{calx_lcdl19}{0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0500!}%
\@namedef{calx_lcdl20}{1904!0476!7619!1904!0476!7619!1904!0476!7619!1904!0476!%
7619!1904!0476!7619!1904!0476!7619!1904!0476!7619!1904!0476!7619!1904!0476!%
7619!1904!0476!7619!1904!0476!7619!1904!0476!}%
\@namedef{calx_lcdl21}{5454!5454!5454!5454!5454!5454!5454!5454!5454!5454!5454!%
5454!5454!5454!5454!5454!5454!5454!5454!5454!5454!5454!5454!5454!5454!5454!%
5454!5454!5454!5454!5454!5454!5454!5454!0454!}%
\@namedef{calx_lcdl22}{7826!0434!3913!5217!6956!2608!3478!1304!1739!5652!0869!%
7826!0434!3913!5217!6956!2608!3478!1304!1739!5652!0869!7826!0434!3913!5217!%
6956!2608!3478!1304!1739!5652!0869!7826!0434!}%
\@namedef{calx_lcdl23}{6666!6666!6666!6666!6666!6666!6666!6666!6666!6666!6666!%
6666!6666!6666!6666!6666!6666!6666!6666!6666!6666!6666!6666!6666!6666!6666!%
6666!6666!6666!6666!6666!6666!6666!6666!0416!}%
\@namedef{calx_lcdl24}{0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0400!}%
\@namedef{calx_lcdl25}{6153!5384!8461!6153!5384!8461!6153!5384!8461!6153!5384!%
8461!6153!5384!8461!6153!5384!8461!6153!5384!8461!6153!5384!8461!6153!5384!%
8461!6153!5384!8461!6153!5384!8461!6153!0384!}%
\@namedef{calx_lcdl26}{3703!0370!7037!3703!0370!7037!3703!0370!7037!3703!0370!%
7037!3703!0370!7037!3703!0370!7037!3703!0370!7037!3703!0370!7037!3703!0370!%
7037!3703!0370!7037!3703!0370!7037!3703!0370!}%
\@namedef{calx_lcdl27}{1428!2857!5714!1428!2857!5714!1428!2857!5714!1428!2857!%
5714!1428!2857!5714!1428!2857!5714!1428!2857!5714!1428!2857!5714!1428!2857!%
5714!1428!2857!5714!1428!2857!5714!1428!0357!}%
\@namedef{calx_lcdl28}{7931!2413!5517!6896!8620!8275!0344!7931!2413!5517!6896!%
8620!8275!0344!7931!2413!5517!6896!8620!8275!0344!7931!2413!5517!6896!8620!%
8275!0344!7931!2413!5517!6896!8620!8275!0344!}%
\@namedef{calx_lcdl29}{3333!3333!3333!3333!3333!3333!3333!3333!3333!3333!3333!%
3333!3333!3333!3333!3333!3333!3333!3333!3333!3333!3333!3333!3333!3333!3333!%
3333!3333!3333!3333!3333!3333!3333!3333!0333!}%
\@namedef{calx_lcdl30}{3225!1290!4516!5806!0322!6129!6451!2580!9032!1612!0645!%
2258!2903!5161!8064!3225!1290!4516!5806!0322!6129!6451!2580!9032!1612!0645!%
2258!2903!5161!8064!3225!1290!4516!5806!0322!}%
\fi
```

---

35. Library calxtrig implementation

Added at calx 2.0a1.

35.1 Catcodes, ε-TEX and reload detection . . . 958
35.2 Package identification . . . 959
35.3 Loading data tables . . . 959
35.4 Package implementation . . . 960
35.5 Function table entries, \calxFpSin, \calxFpCos . . . 960
35.6 Dispatch and kernels for P at most 8 . . . 965
35.7 Series kernel with column short products . . . 978
35.8 Series kernels with three-limb passes (P above 40) . . . 981
35.9 Series kernels with fixed-arity short products (P up to 40) . . . 986
35.10 Inverse functions; tangent, secant, sinc, degrees . . . 992
35.11 Generated helpers and products . . . 1012
35.12 Schedules of the fixed path . . . 1020
35.13 Argument reduction . . . 1020
35.14 Trigonometric functions of \xinteval, constants . . . 1035
35.15 Reconstruction at the scientific precision . . . 1037
35.16 \xintreloadcalxtrig, \xintreloadxinttrig . . . 1039

calxtrig.tex: native sin, cos for calx.sty (wip). Input by calx.sty (catcodes of calx.sty in force -- except for ! which will have catcode 12).

35.1. Catcodes, ε-TEX and reload detection

The code of this subsection is copied over from the xint packages, with the package names changed.

```
\begingroup\catcode61\catcode48\catcode32=10\relax%
  \catcode13=5    % ^^M
  \endlinechar=13 %
  \catcode123=1   % {
  \catcode125=2   % }
  \catcode64=11   % @
  \catcode44=12   % ,
  \catcode46=12   % .
  \catcode58=12   % :
  \catcode94=7    % ^
  \catcode35=6    % #
  \catcode33=12   % !
  \def\empty{}\def\space{ }\newlinechar10
  \expandafter\let\expandafter\t\csname ver@calxtrig.tex\endcsname
  \expandafter\let\expandafter\c\csname ver@calx.sty\endcsname
  \expandafter
    \ifx\csname PackageWarningNoLine\endcsname\relax
      \def\y#1#2{\immediate\write128{^^JPackage #1 Warning:^^J%
                  \space\space\space\space#2.^^J}}%
    \else
      \def\y#1#2{\PackageWarningNoLine{#1}{#2}}%
    \fi
  % attention that in this file \z by default does \endinput.
  \def\z{\endgroup\endinput}%
  \expandafter\ifx\csname numexpr\endcsname\relax
    \y{calx}{\numexpr not available, aborting input}%
  \else
```

BEWARE: on 2026/10/09 I hit a problem I knew about from years ago but forgot. It is known that TeX executes all macros on the line where \endinput is found. But imagine \endinput\input foo.tex\relax, then what will happen is that only the first line of foo.tex (if its EOL is not hidden by a comment char) is executed, so basically foo.tex is not read beyond that. I will put \endinput last in both situations, as is logical anyhow.

```
      \ifx\c\relax % calx.sty not yet loaded.
        \ifx\PackageWarningNoLine\relax
          \y{calx}{%
             Loading of calxtrig.tex should happen via
             \string\input\space calx.sty\relax^^J\space\space\space\space
             Will try that now%
          }%
          \def\z{\endgroup
                 \input calx.sty\relax\endinput}%
        \else
          \y{calx}{%
             Bad move! Inputting calxtrig.tex directly in a LaTeX\MessageBreak
             document is forbidden! Simply use \string\usepackage{calx}\MessageBreak
             in the preamble, as every obedient LaTeX user does!\MessageBreak
             Trying to recover now, hoping to be in the preamble%
          }%
          \def\z{\endgroup
                 \RequirePackage{calx}\endinput}%
        \fi
      \else % either calx.sty is trying to load us, or a second loading is
            % attempted on us by \input. Abort if the latter.
        \ifx\t\relax
            \def\z{\endgroup}%
        \else
            \y{calx}{calxtrig.tex already loaded! Aborting this input}%
        \fi
      \fi
  \fi
\z%
```

We arrive here only after a \XINTsetcatcodes which is done (near its tail) in calx.sty. In particular ! has its normal catcode 12. But attention: the \XINTsetcatcode does nothing with the letters F, I, N so we have to be careful whenever we set them to be of catcode "other" (=12).

There appears to be a problem with miniltx which provides \ProvidesFile but does not let it define \ver@calxtrig.tex. So we define it always.

35.2. Package identification

```
\ifdefined\ProvidesFile
  \ProvidesFile{calxtrig.tex}%
  [2026/11/26 v2.0a2 A calx component (JFB)]%
\fi
\expandafter\def\csname ver@calxtrig.tex\endcsname
  {2026/11/26 v2.0a2 A calx component (JFB)}%
```

35.3. Loading data tables

35.4. Package implementation

Copied unchanged from xinttrig.sty: \calx_tSinAux_b, \calx_tCosAux_b (Horner in \numexpr, 9-digit fixed point).

```
\def\calx_tSinAux_b#1.%
{%
  (((((((%
  -#1/110+\xint_c_e_ix)/%
  -72)*#1/\xint_c_e_ix+\xint_c_e_ix)/%
  -42)*#1/\xint_c_e_ix+\xint_c_e_ix)/%
  -20)*#1/\xint_c_e_ix+\xint_c_e_ix)/%
  -6)*#1/\xint_c_e_ix+\xint_c_e_ix
}%
\def\calx_tCosAux_b#1.%
{%
  ((((((((((%
  -#1/132+\xint_c_e_ix)/%
  -90)*#1/\xint_c_e_ix+\xint_c_e_ix)/%
  -56)*#1/\xint_c_e_ix+\xint_c_e_ix)/%
  -30)*#1/\xint_c_e_ix+\xint_c_e_ix)/%
  -12)*#1/\xint_c_e_ix+\xint_c_e_ix)/%
  -2)*#1/\xint_c_e_ix+\xint_c_e_ix
}%
```

35.5. Function table entries, \calxFpSin, \calxFpCos

calx: function table entries (pattern of \XINT_flexpr_func_sqr).

```
\def\XINT_flexpr_func_sin #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\calxFpSin#3}}%
}%
\def\XINT_flexpr_func_cos #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\calxFpCos#3}}%
}%
\def\XINT_flexpr_func_asin #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\calxFpAsin#3}}%
}%
\def\XINT_flexpr_func_acos #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\calxFpAcos#3}}%
}%
\def\XINT_flexpr_func_asind #1#2#3%
{%
```

```
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\calxFpAsind#3}}%
}%
\def\XINT_flexpr_func_acosd #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\calxFpAcosd#3}}%
}%
\def\XINT_flexpr_func_acsc #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\calxFpAcsc#3}}%
}%
\def\XINT_flexpr_func_asec #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\calxFpAsec#3}}%
}%
\def\XINT_flexpr_func_atan #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\calxFpAtan#3}}%
}%
\def\XINT_flexpr_func_atand #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\calxFpAtand#3}}%
}%
\def\XINT_flexpr_func_tan #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\calxFpTan#3}}%
}%
\def\XINT_flexpr_func_cot #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\calxFpCot#3}}%
}%
\def\XINT_flexpr_func_sec #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\calxFpSec#3}}%
}%
```

```
\def\XINT_flexpr_func_csc #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\calxFpCsc#3}}%
}%
\def\XINT_flexpr_func_sinc #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\calxFpSinc#3}}%
}%
\def\XINT_flexpr_func_sind #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\calxFpSind#3}}%
}%
\def\XINT_flexpr_func_cosd #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\calxFpCosd#3}}%
}%
\def\XINT_flexpr_func_tand #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\calxFpTand#3}}%
}%
\def\XINT_flexpr_func_cotd #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\calxFpCotd#3}}%
}%
\def\XINT_flexpr_func_secd #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\calxFpSecd#3}}%
}%
\def\XINT_flexpr_func_cscd #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:one
    {\romannumeral`&&@\calxFpCscd#3}}%
}%
\expandafter\def\csname XINT_flexpr_func_Arg\endcsname #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:two
```

```
    {\romannumeral`&&@\calxFpArg#3}}%
}%
\expandafter\def\csname XINT_flexpr_func_Argd\endcsname #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:two
    {\romannumeral`&&@\calxFpArgd#3}}%
}%
\expandafter\def\csname XINT_flexpr_func_pArg\endcsname #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:two
    {\romannumeral`&&@\calxFppArg#3}}%
}%
\expandafter\def\csname XINT_flexpr_func_pArgd\endcsname #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:two
    {\romannumeral`&&@\calxFppArgd#3}}%
}%
\expandafter\def\csname XINT_flexpr_func_atan2\endcsname #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:two
    {\romannumeral`&&@\calxFpAtantwo#3}}%
}%
\expandafter\def\csname XINT_flexpr_func_atan2d\endcsname #1#2#3%
{%
    \expandafter #1\expandafter #2\expandafter{%
    \romannumeral`&&@\XINT:NEhook:f:one:from:two
    {\romannumeral`&&@\calxFpAtantwod#3}}%
}%
\let\XINT_flexpr_func_tg\XINT_flexpr_func_tan
\let\XINT_flexpr_func_cotg\XINT_flexpr_func_cot
\let\XINT_flexpr_func_arcsin\XINT_flexpr_func_asin
\let\XINT_flexpr_func_arccos\XINT_flexpr_func_acos
\let\XINT_flexpr_func_arctg\XINT_flexpr_func_atan
\let\XINT_flexpr_func_arcsec\XINT_flexpr_func_asec
\let\XINT_flexpr_func_arccsc\XINT_flexpr_func_acsc
```

Syntax (f-expandable; X, Y f-expanded F, I or raw values, rounded to P = \calx_ttrigdigits digits; results F or I values, rounded to P digits):

```
  \calxFp<F>{X} -> F(X)
        F = Sin, Cos, Tan, Cot, Sec, Csc, Sinc (radians),
            Sind, Cosd, Tand, Cotd, Secd, Cscd (degrees),
            Asin, Acos, Asec, Acsc, Atan (radians),
            Asind, Acosd, Atand (degrees)
    \calxFpArg{X}{Y}     -> argument of (X, Y), in (-pi, pi]
    \calxFppArg{X}{Y}    -> the same, in [0, 2pi)
    \calxFpArgd{X}{Y}, \calxFppArgd{X}{Y}       -> the same in degrees
    \calxFpAtantwo{Y}{X}, \calxFpAtantwod{Y}{X}  -> atan2(Y, X) = Arg(X, Y)
```

\calxFpSin{X}, \calxFpCos{X}: P = \calx_ttrigdigits. X is first rounded to P digits, then \calx_tsin_g, \calx_tcos_g: |X| < 0.8 goes to the series kernels \calx_tsin_kernel, \calx_tcos_kernel,

|X| >= 0.8 to the argument reduction (\calx_tsin_r, \calx_tcos_r) then to the same kernels. P <= 8 ("ultra fast, less precise", as xinttrig): \numexpr kernels \calx_tsin_q, \calx_tcos_q (called by \calx_tsin_kernel, \calx_tcos_kernel).

```
\def\calx_tSin{\romannumeral0\expandafter\calx_tsin_a
    \romannumeral0\CALX_fround{\calx_ttrigdigits}}%
\def\calx_tCos{\romannumeral0\expandafter\calx_tcos_a
    \romannumeral0\CALX_fround{\calx_ttrigdigits}}%
\def\calx_tAsin #1{\romannumeral0\expandafter\calx_tad_a
    \romannumeral0\CALX_fround{\calx_ttrigdigits}{#1}a0}%
\def\calx_tAcos #1{\romannumeral0\expandafter\calx_tad_a
    \romannumeral0\CALX_fround{\calx_ttrigdigits}{#1}c0}%
\def\calx_tAsind #1{\romannumeral0\expandafter\calx_tad_a
    \romannumeral0\CALX_fround{\calx_ttrigdigits}{#1}a1}%
\def\calx_tAcosd #1{\romannumeral0\expandafter\calx_tad_a
    \romannumeral0\CALX_fround{\calx_ttrigdigits}{#1}c1}%
\def\calx_tAcsc #1{\romannumeral0\expandafter\calx_tae_a
    \romannumeral0\CALX_fround{\calx_ttrigdigits}{#1}a}%
\def\calx_tAsec #1{\romannumeral0\expandafter\calx_tae_a
    \romannumeral0\CALX_fround{\calx_ttrigdigits}{#1}c}%
\def\calx_tAtan #1{\romannumeral0\expandafter\calx_ttq_a
    \romannumeral0\CALX_fround{\calx_ttrigdigits}{#1}{0}{1}{0}}%
\def\calx_tAtand #1{\romannumeral0\expandafter\calx_ttq_a
    \romannumeral0\CALX_fround{\calx_ttrigdigits}{#1}{0}{1}{1}}%
\def\calx_tTan #1{\romannumeral0\expandafter\calx_tsc_a
    \romannumeral0\CALX_fround{\calx_ttrigdigits}{#1}\calx_tfntan r}%
\def\calx_tCot #1{\romannumeral0\expandafter\calx_tsc_a
    \romannumeral0\CALX_fround{\calx_ttrigdigits}{#1}\calx_tfncot r}%
\def\calx_tSec #1{\romannumeral0\expandafter\calx_tsc_a
    \romannumeral0\CALX_fround{\calx_ttrigdigits}{#1}\calx_tfnsec r}%
\def\calx_tCsc #1{\romannumeral0\expandafter\calx_tsc_a
    \romannumeral0\CALX_fround{\calx_ttrigdigits}{#1}\calx_tfncsc r}%
\def\calx_tSinc #1{\romannumeral0\expandafter\calx_tsc_a
    \romannumeral0\CALX_fround{\calx_ttrigdigits}{#1}\calx_tfnsinc r}%
\def\calx_tSind #1{\romannumeral0\expandafter\calx_tdg_a
    \romannumeral0\CALX_fround{\calx_ttrigdigits}{#1}\calx_tfnsin s}%
\def\calx_tCosd #1{\romannumeral0\expandafter\calx_tdg_a
    \romannumeral0\CALX_fround{\calx_ttrigdigits}{#1}\calx_tfncos c}%
\def\calx_tTand #1{\romannumeral0\expandafter\calx_tdg_a
    \romannumeral0\CALX_fround{\calx_ttrigdigits}{#1}\calx_tfntan t}%
\def\calx_tCotd #1{\romannumeral0\expandafter\calx_tdg_a
    \romannumeral0\CALX_fround{\calx_ttrigdigits}{#1}\calx_tfncot u}%
\def\calx_tSecd #1{\romannumeral0\expandafter\calx_tdg_a
    \romannumeral0\CALX_fround{\calx_ttrigdigits}{#1}\calx_tfnsec v}%
\def\calx_tCscd #1{\romannumeral0\expandafter\calx_tdg_a
    \romannumeral0\CALX_fround{\calx_ttrigdigits}{#1}\calx_tfncsc w}%
\def\calx_tArg {\romannumeral0\calx_tar_a a0}%
\def\calx_tArgd {\romannumeral0\calx_tar_a a1}%
\def\calx_tpArg {\romannumeral0\calx_tar_a p0}%
\def\calx_tpArgd {\romannumeral0\calx_tar_a p1}%
\def\calx_tAtantwo #1#2{\romannumeral0\calx_tar_a a0{#2}{#1}}%
\def\calx_tAtantwod #1#2{\romannumeral0\calx_tar_a a1{#2}{#1}}%
```

Reduction constants (130 digits, mpmath; the 80 digits of xinttrig.sty are a prefix): \calx_tt

rigR = 180/pi, \calx_ttrigDg = pi/180; \calx_tpdg<P> = pi/180 rounded to P+4 digits, P = 1, ..., 100.

```
\edef\calx_ttrigR{\romannumeral0\CALX_ffromraw
  {57295779513082320876798154814105170332405472466564321549160243861202847148%
  32155263244096899585111094418622338163286489328144826460[-128]]}}%
\edef\calx_ttrigDg{\romannumeral0\CALX_ffromraw
  {17453292519943295769236907684886127134428718885417254560971914401710091146%
  03449443682241569634509482212304492507379059248385469228[-131]]}}%
```

35.6. Dispatch and kernels for P at most 8

Names with uppercase F, I, N are not usable below (catcode 12).

```
%
\let\calx_tsinaux\calx_tSinAux_b \let\calx_tcosaux\calx_tCosAux_b
```

ATTENTION setting from here on F, I, N to "other" catcode. The following definitions are read with I of catcode 12.

```
\let\calx_tglobal\XINT_global
\catcode`F 12 \catcode`I 12 \catcode`N 12
\def\calx_tsin_a #1%
{%
    \if I#1\expandafter\calx_tsin_i\else\expandafter\calx_tsin_g\fi #1
}%
\def\calx_tcos_a #1%
{%
    \if I#1\expandafter\calx_tsin_i\else\expandafter\calx_tcos_g\fi #1
}%
```

#1 is I and we perhaps do not want to rely on its catcode being 12.

```
\def\calx_tsin_i #1#2\xint:{\calx_ttr_sp#2}%
```

Dispatch. |X| < 0.8: \calx_tsin_kernel, \calx_tcos_kernel below; otherwise reduction (\calx_tsin_r, \calx_tcos_r).

```
\def\calx_tsin_g F#1#2.%
{%
    \ifnum#2=\xint_c_ \xint_dothis\calx_tsin_g_zero\fi
    \xint_orthat\calx_tsin_g_nonzero #1#2.%
}%
\def\calx_tsin_g_zero #1#2.#3.#4\xint:{ F+0.0.}%
\def\calx_tsin_g_nonzero #1#2.#3.#4\xint:%
{%
    \ifnum\numexpr#2+8*#3>\xint_c_ \xint_dothis\calx_tsin_r\fi
    \ifnum\ifnum\numexpr#2+8*#3=\xint_c_
      \calx_ttrig_top#4\xint:\else0\fi>\xint_c_vii
      \xint_dothis\calx_tsin_r\fi
    \xint_orthat\calx_tsin_g_small #1#2.#3.#4\xint:%
}%
\def\calx_tsin_g_small #1#2.#3.#4\xint:%
{\calx_tsin_kernel#1#2.#3.#4\xint:}%
\def\calx_tcos_g F#1#2.%
{%
    \ifnum#2=\xint_c_ \xint_dothis\calx_tcos_g_zero\fi
    \xint_orthat\calx_tcos_g_nonzero #1#2.%
}%
```

```
\def\calx_tcos_g_zero #1#2.#3.#4\xint:{ F+1.0.100000001!}%
\def\calx_tcos_g_nonzero #1#2.#3.#4\xint:%
{%
    \ifnum\numexpr#2+8*#3>\xint_c_ \xint_dothis\calx_tcos_g_reduce\fi
    \ifnum\ifnum\numexpr#2+8*#3=\xint_c_
      \calx_ttrig_top#4\xint:\else0\fi>\xint_c_vii
      \xint_dothis\calx_tcos_g_reduce\fi
    \xint_orthat\calx_tcos_g_small #1#2.#3.#4\xint:%
}%
\def\calx_tcos_g_reduce #1#2.#3.#4\xint:{\calx_tcos_r+#2.#3.#4\xint:}%
\def\calx_tcos_g_small #1#2.#3.#4\xint:%
{\calx_tcos_kernel#1#2.#3.#4\xint:}%
\def\calx_ttrig_top 1#1#2!#3\xint:%
{%
    \if\relax\detokenize{#3}\relax\expandafter\xint_firstoftwo
    \else\expandafter\xint_secondoftwo\fi
    {#1}{\calx_ttrig_top#3\xint:}%
}%
```

P <= 8, |X| < 0.8 (\calx_tsin_q, \calx_tcos_q, called by \calx_tsin_kernel, \calx_tcos_kernel): X has E = D + 8e <= 0 (|X| < 10^E); X may have up to 8m digits (reduced argument). E < -4: sin X = round_P(X), cos X = 1 (u/6 < 2 10^-11). Else A = the top 9 digits of M rounded (\calx_trta), X = A 10^(E-9) (1 + d), |d| < 6 10^-9; U9 = round(round(A^2/10^9) / 10^(-2E)) (= round(10^9 u) within 1); S9 = 10^9 sin(X)/X, C9 = 10^9 cos(X) by the Horner kernels of xinttrig; sin X = round_P(round(A S9/10^9) 10^(E-9)), cos X = round_P(C9 10^-9), both rounded and written as natives by \calx_tdgo (below).

```
\def\calx_tsin_q #1#2.#3.%
{%
    \ifnum\numexpr#2+8*#3<-\xint_c_iv
      \xint_dothis\calx_tsin_q_tiny\fi
    \xint_orthat\calx_tsin_q_compute #1#2.#3.%
}%
\def\calx_tsin_q_tiny #1#2.#3.#4\xint:%
  {\CALX_fnround{\calx_ttrigdigits}{F#1#2.#3.#4}}%
\def\calx_tsin_q_compute #1#2.#3.#4\xint:%
{%
    \expandafter\calx_tsqa\the\numexpr#2+8*#3\expandafter.%
      \the\numexpr(2*#2+7)/16\expandafter.%
      \the\numexpr#2+8-8*((2*#2+7)/16).#1#4\xint:%
}%
\def\calx_tsqa #1.#2.#3.#4#5\xint:%
  {\expandafter\calx_tsqb\the\numexpr\calx_trta#2.#3.#5\xint:.#4#1.}%
\def\calx_tsqb #1.#2#3.%
{%
    \expandafter\calx_tdgo\the\numexpr#2#1*(%
      \expandafter\calx_tsinaux\the\numexpr#1*#1/1000000000/%
      \CALX_pten{-2*#3}.)/1000000000.#3-9.%
}%
\def\calx_tcos_q #1#2.#3.%
{%
    \ifnum\numexpr#2+8*#3<-\xint_c_iv \xint_dothis\calx_tcos_q_tiny\fi
    \xint_orthat\calx_tcos_q_compute #1#2.#3.%
}%
```

```
\def\calx_tcos_q_tiny #1#2.#3.#4\xint:{ F+1.0.100000001!}%
\def\calx_tcos_q_compute #1#2.#3.#4\xint:%
{%
    \expandafter\calx_tcqa\the\numexpr#2+8*#3\expandafter.%
      \the\numexpr(2*#2+7)/16\expandafter.%
      \the\numexpr#2+8-8*((2*#2+7)/16).#4\xint:%
}%
\def\calx_tcqa #1.#2.#3.#4\xint:%
  {\expandafter\calx_tcqb\the\numexpr\calx_trta#2.#3.#4\xint:.#1.}%
\def\calx_tcqb #1.#2.%
{%
    \expandafter\calx_tdgo\the\numexpr
      \expandafter\calx_tcosaux\the\numexpr#1*#1/1000000000/%
      \CALX_pten{-2*#2}..-9.%
}%
```

sind, cosd, tand, cotd, secd, cscd for P <= 8, all in \numexpr, 9-digit fixed point as \calx_tsinq (kinds s, c, t, u, v, w). P > 8, NaN and infinities go to \calx_tsc_a (jumper \calx_tdg_j), zero gives 0 or 1.

Input (after \CALX_fround to P): |x| = A 10^(E-9) EXACTLY, E = D + 8e, A integer, 10^8 <= A < 10^9 (\calx_tdg_one: one block, A = b 10^(9-D); \calx_tdg_two: two blocks, A = b_1 10^(17-D) + b_0 / 10^(D-9), the division being exact: x has at most P + 1 <= 9 significant digits).

E < 2 (|x| < 10): no reduction, y = |x| (jumper \calx_tdg_l). Otherwise the exact reduction of xinttrig's \XINT_sind_i (mod 360, quadrant by rounding), in integers:

I = floor(|x|), n = I mod 360, q = round((2n+1)/180) in 0..4,
10^7 y = (n - 90 q) 10^7 + 10^7 frac(|x|), -45 <= y <= 45,

then y = A' 10^(E'-9) (\calx_tdgw, \calx_tdgd), y = 0 giving A' = 0. Quadrant (\calx_tdgq<kind><q>): sind = sign(x) times S(y), C(y), -S(y), -C(y), S(y) for q = 0..4, cosd = C(y), -S(y), -C(y), S(y), C(y), with S = sind, C = cosd on [-45, 45] (\calx_tdgks, \calx_tdgkc); tand, cotd (odd, factor sign(x)): T, -1/T, T, -1/T, T and 1/T, -T, 1/T, -T, 1/T (T = tand y); secd: 1/C, -1/S, -1/C, 1/S, 1/C; cscd (odd): 1/S, 1/C, -1/S, -1/C, 1/S. Zero: tand 0, secd 1, cotd and cscd DivisionByZero. For P > 8 the kinds t, u, v, w are those of \calx_tsc_a for tand, cotd (\calx_tsc_d), secd (\calx_tsc_c), cscd (\calx_tsc_s).

```
\def\calx_tdg_small #1%
{%
    \if I#1\calx_tdg_j\fi
    \calx_tdg_b #1%
}%
\def\calx_tdg_j\fi #1\calx_tdg_b #2\xint:{\fi\calx_tsc_a #2\xint:}%
\def\calx_tdg_b F#1#2.#3.#4\xint:#5%
{%
    \ifnum#2=\xint_c_ \calx_tdg_z\fi
    \calx_tdg_c #2.#3.#1#4\xint:%
}%
\def\calx_tdg_z\fi\calx_tdg_c #1\xint:#2{\fi\csname calx_tdgz#2\endcsname}%
\expandafter\def\csname calx_tdgzs\endcsname{ F+0.0.}%
\expandafter\def\csname calx_tdgzc\endcsname{ F+1.0.100000001!}%
\expandafter\def\csname calx_tdgzt\endcsname{ F+0.0.}%
\expandafter\def\csname calx_tdgzu\endcsname
  {\CALX_nsig{DivisionByZero}{x / 0}{+}}%
\expandafter\let\csname calx_tdgzv\expandafter\endcsname
  \csname calx_tdgzc\endcsname
\expandafter\let\csname calx_tdgzw\expandafter\endcsname
```

```
  \csname calx_tdgzu\endcsname
\def\calx_tdg_c #1.#2.#31#4!#5%
  {\xint_gob_til_xint: #5\calx_tdg_one\xint:\calx_tdg_two #1.#2.#3#4.}%
\def\calx_tdg_one\xint:\calx_tdg_two #1.#2.#3#4.%
{%
    \expandafter\calx_tdg_d\the\numexpr#1+8*#2\expandafter.%
    \the\numexpr#4*\CALX_pten{9-#1}.#3%
}%
\def\calx_tdg_two #1.#2.#3#4.#5!\xint:%
{%
    \expandafter\calx_tdg_d\the\numexpr#1+8*#2\expandafter.%
    \the\numexpr#5*\CALX_pten{17-#1}+#4/\CALX_pten{#1-9}.#3%
}%
\def\calx_tdg_d #1.%
{%
    \ifnum#1<\xint_c_ii \calx_tdg_l\fi
    \csname calx_tdge\ifnum#1>11 c\else#1\fi\endcsname#1.%
}%
\def\calx_tdg_l\fi #1\endcsname#2.#3.#4#5%
  {\fi\csname calx_tdgd#5\endcsname #4#3.#2.}%
\expandafter\def\csname calx_tdgds\endcsname #1#2.#3.{\calx_tdgks #11.#2.#3.}%
\expandafter\def\csname calx_tdgdc\endcsname #1#2.#3.{\calx_tdgkc 1.#2.#3.}%
\expandafter\def\csname calx_tdgdt\endcsname #1#2.#3.{\calx_tdgkt #11.#2.#3.}%
\expandafter\def\csname calx_tdgdu\endcsname #1#2.#3.%
  {\csname calx_tdgkb+\endcsname #11.#2.#3.}%
\expandafter\def\csname calx_tdgdv\endcsname #1#2.#3.{\calx_tdgke 1.#2.#3.}%
\expandafter\def\csname calx_tdgdw\endcsname #1#2.#3.{\calx_tdgkr #11.#2.#3.}%
```

\calx_tdge<E> <E>.<A>. (one \csname from \calx_tdg_d, E = 2..11, c for E > 11) gives \calx_tdg_s <n>.<10^7 frac(|x|)>. For E <= 9, with k = 9 - E: I = floor(A/10^k) = round((2A - 10^k)/(2 10^k)) and 10^7 frac(|x|) = (A - I 10^k) 10^(E-2) (exact: frac(|x|) has at most k <= 7 decimal digits); \calx_tdgdef builds these eight macros with literal powers of ten. For E >= 10, x is an integer: n = ((A mod 360) (10^(E-9) mod 360)) mod 360, 10^(E-9) = 10, 100 or 280 (mod 360).

```
\def\calx_tdgdef #1#2#3#4%
{%
    \expandafter\def\csname calx_tdge#1\endcsname##1.##2.%
    {%
      \expandafter\calx_tdg_s\the\numexpr\calx_tmod{((2*##2-#2)/#3)}%
      \expandafter.\the\numexpr(##2-#2*((2*##2-#2)/#3))#4.%
    }%
}%
\calx_tdgdef 2{10000000}{20000000}{}%
\calx_tdgdef 3{1000000}{2000000}{*10}%
\calx_tdgdef 4{100000}{200000}{*100}%
\calx_tdgdef 5{10000}{20000}{*1000}%
\calx_tdgdef 6{1000}{2000}{*10000}%
\calx_tdgdef 7{100}{200}{*100000}%
\calx_tdgdef 8{10}{20}{*1000000}%
\calx_tdgdef 9{1}{2}{*10000000}%
\expandafter\def\csname calx_tdge10\endcsname #1.#2.%
  {\expandafter\calx_tdg_s\the\numexpr\calx_tmod{\calx_tmod{#2}*10}.0.}%
\expandafter\def\csname calx_tdge11\endcsname #1.#2.%
  {\expandafter\calx_tdg_s\the\numexpr\calx_tmod{\calx_tmod{#2}*100}.0.}%
```

```
\expandafter\def\csname calx_tdgec\endcsname #1.#2.%
  {\expandafter\calx_tdg_s\the\numexpr\calx_tmod{\calx_tmod{#2}*280}.0.}%
```

\calx_tdg_s <n>.<r>., \calx_tdg_t: q and 10^7 y; \calx_tdg_u selects the quadrant macro with the sign token of x.

```
\def\calx_tdg_s #1.{\expandafter\calx_tdg_t\the\numexpr(2*#1+1)/180.#1.}%
\def\calx_tdg_t #1.#2.#3.%
  {\expandafter\calx_tdg_u\the\numexpr(#2-90*#1)*10000000+#3.#1.}%
\def\calx_tdg_u #1.#2.#3#4{\csname calx_tdgq#4#2\endcsname#3#1.}%
\expandafter\def\csname calx_tdgqs0\endcsname#1{\calx_tdgw\calx_tdgks#11.}%
\expandafter\def\csname calx_tdgqs1\endcsname#1{\calx_tdgw\calx_tdgkc#11.}%
\expandafter\def\csname calx_tdgqs2\endcsname#1{\calx_tdgw\calx_tdgks#1-1.}%
\expandafter\def\csname calx_tdgqs3\endcsname#1{\calx_tdgw\calx_tdgkc#1-1.}%
\expandafter\def\csname calx_tdgqs4\endcsname#1{\calx_tdgw\calx_tdgks#11.}%
\expandafter\def\csname calx_tdgqc0\endcsname#1{\calx_tdgw\calx_tdgkc1.}%
\expandafter\def\csname calx_tdgqc1\endcsname#1{\calx_tdgw\calx_tdgks-1.}%
\expandafter\def\csname calx_tdgqc2\endcsname#1{\calx_tdgw\calx_tdgkc-1.}%
\expandafter\def\csname calx_tdgqc3\endcsname#1{\calx_tdgw\calx_tdgks1.}%
\expandafter\def\csname calx_tdgqc4\endcsname#1{\calx_tdgw\calx_tdgkc1.}%
\expandafter\def\csname calx_tdgqt0\endcsname#1{\calx_tdgw\calx_tdgkt#11.}%
\expandafter\def\csname calx_tdgqt1\endcsname#1%
  {\expandafter\calx_tdgw\csname calx_tdgkb#1\endcsname#1-1.}%
\expandafter\def\csname calx_tdgqt2\endcsname#1{\calx_tdgw\calx_tdgkt#11.}%
\expandafter\def\csname calx_tdgqt3\endcsname#1%
  {\expandafter\calx_tdgw\csname calx_tdgkb\if-#1+\else-\fi\endcsname#1-1.}%
\expandafter\def\csname calx_tdgqt4\endcsname#1{\calx_tdgw\calx_tdgkt#11.}%
\expandafter\def\csname calx_tdgqu0\endcsname#1%
  {\expandafter\calx_tdgw\csname calx_tdgkb+\endcsname#11.}%
\expandafter\def\csname calx_tdgqu1\endcsname#1{\calx_tdgw\calx_tdgkt#1-1.}%
\expandafter\def\csname calx_tdgqu2\endcsname#1%
  {\expandafter\calx_tdgw\csname calx_tdgkb-\endcsname#11.}%
\expandafter\def\csname calx_tdgqu3\endcsname#1{\calx_tdgw\calx_tdgkt#1-1.}%
\expandafter\def\csname calx_tdgqu4\endcsname#1%
  {\expandafter\calx_tdgw\csname calx_tdgkb+\endcsname#11.}%
\expandafter\def\csname calx_tdgqv0\endcsname#1{\calx_tdgw\calx_tdgke1.}%
\expandafter\def\csname calx_tdgqv1\endcsname#1{\calx_tdgw\calx_tdgkr-1.}%
\expandafter\def\csname calx_tdgqv2\endcsname#1{\calx_tdgw\calx_tdgke-1.}%
\expandafter\def\csname calx_tdgqv3\endcsname#1{\calx_tdgw\calx_tdgkr1.}%
\expandafter\def\csname calx_tdgqv4\endcsname#1{\calx_tdgw\calx_tdgke1.}%
\expandafter\def\csname calx_tdgqw0\endcsname#1{\calx_tdgw\calx_tdgkr#11.}%
\expandafter\def\csname calx_tdgqw1\endcsname#1{\calx_tdgw\calx_tdgke#11.}%
\expandafter\def\csname calx_tdgqw2\endcsname#1{\calx_tdgw\calx_tdgkr#1-1.}%
\expandafter\def\csname calx_tdgqw3\endcsname#1{\calx_tdgw\calx_tdgke#1-1.}%
\expandafter\def\csname calx_tdgqw4\endcsname#1{\calx_tdgw\calx_tdgkr#11.}%
```

\calx_tdgw <kernel><factor>.<v>. (v = 10^7 y): \calx_tdgd gives, from the digit count d of |v|, 10^(9-d) and E' = d - 7; then \calx_tdgx calls the kernel with <factor (+1 or -1)>.<A' = v 10^(9-d)>.<E'>. (A' carries the sign of y).

```
\def\calx_tdgw #1#2.#3.%
{%
    \expandafter\calx_tdgx\expanded{\expandafter\calx_tdgd
      \the\numexpr\ifnum#3<\xint_c_-\fi#3.}#3.#1#2.%
}%
\def\calx_tdgd #1.%
```

```
{%
    \CALX_nd_c #1{1.2}{1.2}{10.1}{100.0}{1000.-1}{10000.-2}{100000.-3}%
    {1000000.-4}{10000000.-5}{100000000.-6}\xint:.%
}%
\def\calx_tdgx #1.#2.#3.#4#5.%
  {\expandafter#4\the\numexpr#5\expandafter.\the\numexpr#3*#1.#2.}%
```

Kernels \calx_tdgks, \calx_tdgkc <f>.<A>.<E>. (f = +-1, |y| = |A| 10^(E-9) degrees, t = pi y/180 radians). \calx_tdgu (inside \numexpr): U9 = round(10^9 t^2) = round(round(A^2/10^9) r_E) with r_E = p_E/q_E, the best approximation of 10^(2E) (pi/180)^2 by p_E, q_E < 2^31 (continued fraction, relative error < 10^-11), \calx_tdgr<E> for E = -2..2; E < -2: U9 = 0 (t^2 < 3.1 10^-10). S9 = 10^9 sin(t)/t, C9 = 10^9 cos(t) by \calx_tsinaux, \calx_tcosaux (xinttrig's Horner kernels);

```
     sind y = round(f A Z / 10^9) 10^(E-11),   Z = round(S9 p/q),
     cosd y = f C9 10^-9,
```

p/q = 1489429756/853380389 (100 pi/180, relative error 10^-19). Error before the final rounding: a few units of 10^-9 relative (Horner, U9, Z), i.e. about 1 ulp at P = 8.

```
\def\calx_tdgu #1.#2.%
  {\ifnum#2<-\xint_c_ii 0\else
   #1*#1/1000000000\csname calx_tdgr#2\endcsname\fi}%
\expandafter\def\csname calx_tdgr2\endcsname{*920239054/302096661}%
\expandafter\def\csname calx_tdgr1\endcsname{*53922105/1770158287}%
\expandafter\def\csname calx_tdgr0\endcsname{*532125/1746863329}%
\expandafter\def\csname calx_tdgr-1\endcsname{*4337/1423753114}%
\expandafter\def\csname calx_tdgr-2\endcsname{*2/65656127}%
\def\calx_tdgks #1.#2.#3.%
{%
    \expandafter\calx_tdgo\the\numexpr#1*#2*(%
      (\expandafter\calx_tsinaux\the\numexpr\calx_tdgu#2.#3..)%
      *1489429756/853380389)/1000000000.#3-11.%
}%
\def\calx_tdgkc #1.#2.#3.%
{%
    \expandafter\calx_tdgo\the\numexpr#1*(%
      \expandafter\calx_tcosaux\the\numexpr\calx_tdgu#2.#3..).-9.%
}%
```

Kernels of tand, cotd, secd, cscd (same arguments <f>.<A>.<E>.; Z, C9 as above):

```
     tan y = round(f A Z / C9) 10^(E-11)                  (\calx_tdgkt)
     cot y = round(f 10^17 / round(A Z / C9)) 10^(-E-6)   (\calx_tdgkb<s>)
     sec y = round(f 10^18 / C9) 10^-9                    (\calx_tdgke)
     csc y = round(f round(10^17 / A) 10^9 / Z) 10^(-E-6) (\calx_tdgkr)
```

(products then divisions as one scaled \numexpr operation, all values below 2^31). y = 0 (A = 0) in cot, csc: pole, DivisionByZero with the sign s of the numerator of the general path (\calx_tdgkb+, \calx_tdgkb-; csc: +), as \CALX_fdiv gives it.

```
\def\calx_tdgkt #1.#2.#3.%
  {\expandafter\calx_tdgkta\the\numexpr\calx_tdgu#2.#3..#1.#2.#3.}%
\def\calx_tdgkta #1.#2.#3.#4.%
{%
    \expandafter\calx_tdgo\the\numexpr#2*#3*%
      ((\calx_tsinaux#1.)*1489429756/853380389)/(\calx_tcosaux#1.).#4-11.%
}%
\def\calx_tdgkbdef #1%
{%
    \expandafter\def\csname calx_tdgkb#1\endcsname##1.##2##3.##4.%
```

```
      {\xint_gob_til_zero##2\calx_tdgpz#10\calx_tdgkba##1.##2##3.##4.}%
}%
\calx_tdgkbdef+\calx_tdgkbdef-\let\calx_tdgkbdef\undefined
\def\calx_tdgpz #10#2#3.#4.#5.%
  {\CALX_nsig{DivisionByZero}{x / 0}{#1}}%
\def\calx_tdgkba #1.#2.#3.%
  {\expandafter\calx_tdgkbb\the\numexpr\calx_tdgu#2.#3..#1.#2.#3.}%
\def\calx_tdgkbb #1.#2.#3.#4.%
{%
    \expandafter\calx_tdgo\the\numexpr#2*(1000000000*100000000/(#3*%
      ((\calx_tsinaux#1.)*1489429756/853380389)/(\calx_tcosaux#1.))).-#4-6.%
}%
\def\calx_tdgke #1.#2.#3.%
{%
    \expandafter\calx_tdgo\the\numexpr#1*(1000000000*1000000000/(%
      \expandafter\calx_tcosaux\the\numexpr\calx_tdgu#2.#3..)).-9.%
}%
\def\calx_tdgkr #1.#2#3.#4.%
  {\xint_gob_til_zero#2\calx_tdgpz+0\calx_tdgkra#1.#2#3.#4.}%
\def\calx_tdgkra #1.#2.#3.%
{%
    \expandafter\calx_tdgo\the\numexpr#1*((1000000000*100000000/#2)*1000000000%
      /((\expandafter\calx_tsinaux\the\numexpr\calx_tdgu#2.#3..)%
      *1489429756/853380389)).-#3-6.%
}%
```

Syntax:
  \calx_tdgo <N>.<x>. -> <space>F<s><D>.<b>.<blocks> round_P(N 10^x)
    \calx_tdgoc <M>.<b>.<g>.<s> -> <space>F<s><D>.<b>.<blocks> (M 10^g)
(P = \calx_ttrigdigits; for \calx_tdgoc, 10^(P-1) <= M <= 10^P, b the block exponent, g - 8b in 0..7.)
  \calx_tdgo <N>.<x>. (N a signed integer of at most 10 digits, x an integer expression; also used by \calx_tsin_q, \calx_tcos_q): the native round_P(N 10^x), without \CALX_ffromraw nor \CALX_fround. \calx_tdgy: sign apart, N = 0 gives F+0.0., else s = (number of digits of N) - P >= 0 (N >= 10^7 for all callers: 8, 9 or 10 digits); \calx_tdgob: M = round(N/10^s) (\numexpr division, half away from zero as \CALX_fround), g = x + s (|value| = M 10^g), block exponent b = round((2g - 7)/16), h = g - 8b in 0..7; \calx_tdgoc: D = h + P, plus 1 if M = 10^P (carry, as \CALX_fround); \calx_tdgod: F<sign><D>.<b>. and the block(s) of M 10^h, one if D <= 8, else two (\calx_tdgoe, \calx_tdgof: high part floor(M / 10^(8-h)), low part the rest times 10^h; D <= 16).

```
\def\calx_tdgo #1%
  {\if-#1\expandafter\calx_tdgy\else\expandafter\calx_tdgy\expandafter+\fi#1}%
\def\calx_tdgy #1#2.#3.%
{%
    \ifnum#2=\xint_c_ \calx_tdgyz\fi
    \expandafter\calx_tdgob
    \the\numexpr\ifnum#2<1000000000 \ifnum#2<100000000 8\else9\fi
      \else10\fi-\calx_ttrigdigits.#2.#3.#1%
}%
\def\calx_tdgyz\fi #1\calx_tdgob #2.#3.#4.#5{\fi\space F+0.0.}%
\expandafter\def\csname CALX_pt9\endcsname{1000000000}%
\def\calx_tdgob #1.#2.#3.%
{%
    \expandafter\calx_tdgoc\the\numexpr#2/\CALX_pten{#1}\expandafter.%
```

```
    \the\numexpr(2*(#3+#1)-7)/16\expandafter.\the\numexpr#3+#1.%
}%
\def\calx_tdgoc{\calx_tdgoP\calx_ttrigdigits.}%
\def\calx_tdgoP #1.#2.#3.#4.%
{%
    \expandafter\calx_tdgod\the\numexpr#4-8*#3\expandafter.%
    \the\numexpr#4-8*#3+#1%
      \ifnum#2<\CALX_pten{#1} \else+1\fi.#2.#3.%
}%
\def\calx_tdgod #1.#2.#3.#4.#5%
{%
    \expanded{ F#5#2.#4.\ifnum#2>\xint_c_viii \calx_tdgoe#1.#3.\else
      \the\numexpr100000000+#3*\CALX_pten{#1}!\fi}%
}%
\def\calx_tdgoe #1.#2.%
{%
    \expandafter\calx_tdgof\the\numexpr(2*#2-\CALX_pten{8-#1})%
      /(2*\CALX_pten{8-#1}).#1.#2.%
}%
\def\calx_tdgof #1.#2.#3.%
{%
    \the\numexpr100000000+(#3-#1*\CALX_pten{8-#2})*\CALX_pten{#2}!%
    \the\numexpr100000000+#1!%
}%
```

asind, acosd, atand and asin, acos, atan for P <= 8, all in \numexpr. P > 8, NaN, infinities, zero, |x| >= 1 (asind, acosd) go to the general code (\calx_tas_a, \calx_tat_a; jumpers \calx_tad_j, \calx_tad_k, \calx_ttq_j, \calx_ttq_k). |x| = A 10^(E-9) exactly (\calx_trta, as \calx_tsin_q).

Small kernel \calx_tad_s <A>.<E>.<den>.<cont> (a = A 10^(E-9) <= sqrt(2)/2): U9 = round(10^9 a^2) (0 if E < -4), Zd = round(R9 p'/q'), R9 = 10^9 asin(a)/a by \calx_tasb (xinttrig's \XINT_Arcsin_Ib), p'/q' = 853380389/1489429756 (180/(100 pi)); gives <cont><N>.<E-7>. with N = round(A Zd/den): asind a = N 10^(E-7) for den = 10^9 (half of it for den = 2 10^9). A need not be normalized (E = 0, absolute accuracy).

Large kernel \calx_tad_l <A>.<cont> (E = 0, A > 707106781): u = sqrt(1 - a^2) = sqrt(d (2 10^9 - d)) 10^-9, d = 10^9 - A exact; with k = 4, 3, 3, 2, 2, 1, 1, 0, 0 for d of 1, ..., 9 digits, s = round(sqrt(d 10^(2k) (2 10^9 - d))) (u = s 10^(-9-k), s < 1.42 10^9) by five Newton steps s + (P M/s - s)/2 (one scaled \numexpr division) from a linear guess (\calx_tad_lc, relative error < 25%); then as the small kernel with A = s: <cont><N>.<-k-7>., asind u = N 10^(-k-7).

Finishers: \calx_tad_ys <N>.<e>.<s> gives round_P(s N 10^e) by \calx_tdgo; \calx_tad_yc <N>.<e>.{B}<o><s> gives round_P(s (B o N 10^(e+7)) 10^-7) (units of 10^-7 degree, B = 45, 90 or 180 degrees). asind: s V (small), s (90 - V(u)) (large); acosd: 90 -+ V (small), V(u) or 180 - V(u) (large).

atand, t = |x|: (i) t <= tan(22.5): x' = 2t/(1+t^2) (A' = 2 A 10^9 / (10^9 + U9), divided by 10 if it has ten digits), atand = s V(x')/2; (ii) E = 0, tan(22.5) < t < 1: y' = (1-t^2)/(1+t^2), atand = s (45 - V(y')/2); (iv) 1 <= t < tan(67.5): y' = (t^2-1)/(t^2+1) from (A - 10^8)(A + 10^8)/10^8 and A^2/10^8 + 10^8, atand = s (45 + V(y')/2); (v) t >= tan(67.5): r = 1/t (round(10^17/A), E' = 1 - E), x' = 2r/(1+r^2), atand = s (90 - V(x')/2). No square root.

Radians: the same kernels (tables \calx_tad<a|c><1|0><s|l>, \calx_ttq<kind> <1|0>, 1: degrees, 0: radians) with the finishers \calx_tad_zs, \calx_tad_zc (result times pi/180 = 1489429756/853380389 / 100, in units of 10^-9 radian, angles up to 90 degrees) and \calx_tad_zm (times pi/18 = 105115715/602268683, units of 10^-8 radian: acos of x < 0, results >= pi/2 up to pi).

acsc, asec for P <= 8 (\calx_tae_a; |x| < 1, zero, x = +-1 and P > 8 go to \calx_tac_a): y = 1/|x|. 1 <= |x| < sqrt(2) (E = 1, A < 141421357): the large kernel on u = sqrt(1 - y^2) with d

= 10^9 (|x| - 1)/|x| from the exact A (\calx_tae_l: d 10^(2k) = (A - 10^8) 10^(2k+1) 10^8/A, no cancellation); otherwise the small kernel on y, A' = round(10^17/A), E' = 1 - E (\calx_tae_t). Tables \calx_tae<s|l|o><a|c><p|m> (sign of x).

```
\def\calx_tad_small #1%
{%
    \if I#1\calx_tad_j\fi
    \calx_tad_b #1%
}%
\def\calx_tad_j\fi #1\calx_tad_b #2\xint:{\fi\calx_tas_a #2\xint:}%
\def\calx_tad_b F#1#2.#3.#4\xint:%
{%
    \ifnum\ifnum#2=\xint_c_ \xint_c_i\else\numexpr#2+8*#3\relax\fi
      >\xint_c_ \calx_tad_k\fi
    \calx_tad_ba #1#2.#3.#4\xint:%
}%
\def\calx_tad_k\fi\calx_tad_ba #1\xint:{\fi\calx_tas_a F#1\xint:}%
\def\calx_tad_ba #1#2.#3.#4\xint:%
{%
    \expandafter\calx_tad_bb\the\numexpr#2+8*#3\expandafter.%
    \the\numexpr(2*#2+7)/16\expandafter.%
    \the\numexpr#2+8-8*((2*#2+7)/16).#4\xint:#1%
}%
\def\calx_tad_bb #1.#2.#3.#4\xint:#5%
  {\expandafter\calx_tad_c\the\numexpr\calx_trta#2.#3.#4\xint:.#1.#5}%
\def\calx_tad_c #1.#2.#3#4#5%
{%
    \csname calx_tad#4#5\ifnum#2<\xint_c_ s\else
      \ifnum#1>707106781 l\else s\fi\fi\endcsname #1.#2.#3%
}%
\expandafter\def\csname calx_tada1s\endcsname #1.#2.#3%
  {\calx_tad_s#1.#2.1000000000.\calx_tad_ys#3}%
\expandafter\def\csname calx_tada1l\endcsname #1.#2.#3%
  {\calx_tad_l#1.\calx_tad_yc{900000000}-#3}%
\expandafter\def\csname calx_tadc1s\endcsname #1.#2.#3%
  {\calx_tad_s#1.#2.1000000000.\calx_tad_yc{900000000}{\if-#3+\else-\fi}+}%
\expandafter\def\csname calx_tadc1l\endcsname #1.#2.#3%
  {\if-#3\expandafter\calx_tad_cm\else\expandafter\calx_tad_cp\fi #1.}%
\def\calx_tad_cp #1.{\calx_tad_l#1.\calx_tad_ys+}%
\def\calx_tad_cm #1.{\calx_tad_l#1.\calx_tad_yc{1800000000}-+}%
\expandafter\def\csname calx_tada0s\endcsname #1.#2.#3%
  {\calx_tad_s#1.#2.1000000000.\calx_tad_zs#3}%
\expandafter\def\csname calx_tada0l\endcsname #1.#2.#3%
  {\calx_tad_l#1.\calx_tad_zc{900000000}-#3}%
\expandafter\def\csname calx_tadc0s\endcsname #1.#2.#3%
  {\if-#3\expandafter\calx_tad_es\else\expandafter\calx_tad_fs\fi #1.#2.}%
\def\calx_tad_es #1.#2.%
  {\calx_tad_s#1.#2.1000000000.\calx_tad_zm{900000000}++}%
\def\calx_tad_fs #1.#2.%
  {\calx_tad_s#1.#2.1000000000.\calx_tad_zc{900000000}-+}%
\expandafter\def\csname calx_tadc0l\endcsname #1.#2.#3%
  {\if-#3\expandafter\calx_tad_dm\else\expandafter\calx_tad_dp\fi #1.}%
\def\calx_tad_dp #1.{\calx_tad_l#1.\calx_tad_zs+}%
```

```
\def\calx_tad_dm #1.{\calx_tad_l#1.\calx_tad_zm{1800000000}-+}%
\def\calx_tad_s #1.#2.%
{%
    \expandafter\calx_tad_sa\the\numexpr\ifnum#2<-\xint_c_iv 0\else
      #1*#1/1000000000/\CALX_pten{-2*#2}\fi.#1.#2.%
}%
\def\calx_tad_sa #1.#2.#3.#4.#5%
{%
    \expandafter#5\the\numexpr#2*((\calx_tasb#1.)*853380389/1489429756)/#4%
    \expandafter.\the\numexpr#3-7.%
}%
\def\calx_tad_l #1.{\expandafter\calx_tad_la\the\numexpr1000000000-#1.}%
\def\calx_tad_la #1.%
  {\expandafter\calx_tad_lb\the\numexpr\CALX_nd_c #10001122334\xint:.#1.}%
\def\calx_tad_lb #1.#2.%
{%
    \expandafter\calx_tad_lc\the\numexpr#2*\CALX_pten{2*#1}\expandafter.%
    \the\numexpr2000000000-#2.#1.%
}%
\def\calx_tad_lc #1.%
{%
    \expandafter\calx_tad_na\the\numexpr\ifnum#1<100000000
      (#1+31622777)*38233/10000\else(#1+316227766)*120903/100000\fi.#1.%
}%
\def\calx_tad_na #1.#2.#3.%
  {\expandafter\calx_tad_nb\the\numexpr#1+(#2*#3/#1-#1)/2.#2.#3.}%
\def\calx_tad_nb #1.#2.#3.%
  {\expandafter\calx_tad_nc\the\numexpr#1+(#2*#3/#1-#1)/2.#2.#3.}%
\def\calx_tad_nc #1.#2.#3.%
  {\expandafter\calx_tad_nd\the\numexpr#1+(#2*#3/#1-#1)/2.#2.#3.}%
\def\calx_tad_nd #1.#2.#3.%
  {\expandafter\calx_tad_ne\the\numexpr#1+(#2*#3/#1-#1)/2.#2.#3.}%
\def\calx_tad_ne #1.#2.#3.%
  {\expandafter\calx_tad_lf\the\numexpr#1+(#2*#3/#1-#1)/2.}%
\def\calx_tad_lf #1.#2.%
{%
    \expandafter\calx_tad_sa\the\numexpr#1*#1/1000000000/\CALX_pten{2*#2}.%
    #1.-#2.1000000000.%
}%
\def\calx_tad_ys #1.#2.#3{\expandafter\calx_tdgo\the\numexpr#3#1.#2.}%
\def\calx_tad_yc #1.#2.#3#4#5%
{%
    \expandafter\calx_tdgo\the\numexpr#51*(#3#4(\ifnum#2<-16 0\else
      #1/\CALX_pten{-#2-7}\fi)).-7.%
}%
\def\calx_tad_zs #1.#2.#3%
{%
    \expandafter\calx_tdgo\the\numexpr#3#1*1489429756/853380389%
    \expandafter.\the\numexpr#2-2.%
}%
\def\calx_tad_zc #1.#2.#3#4#5%
{%
```

```
    \expandafter\calx_tdgo\the\numexpr#51*(#3#4(\ifnum#2<-16 0\else
      #1/\CALX_pten{-#2-7}\fi))*1489429756/853380389.-9.%
}%
\def\calx_tad_zm #1.#2.#3#4#5%
{%
    \expandafter\calx_tdgo\the\numexpr#51*(#3#4(\ifnum#2<-16 0\else
      #1/\CALX_pten{-#2-7}\fi))*105115715/602268683.-8.%
}%
\def\calx_tae_small #1%
{%
    \if I#1\calx_tae_j\fi
    \calx_tae_b #1%
}%
\def\calx_tae_j\fi #1\calx_tae_b #2\xint:{\fi\calx_tac_a #2\xint:}%
\def\calx_tae_b F#1#2.#3.#4\xint:%
{%
    \ifnum\ifnum#2=\xint_c_ \xint_c_\else\numexpr#2+8*#3\relax\fi
      <\xint_c_i \calx_tae_k\fi
    \calx_tae_ba #1#2.#3.#4\xint:%
}%
\def\calx_tae_k\fi\calx_tae_ba #1\xint:{\fi\calx_tac_a F#1\xint:}%
\def\calx_tae_ba #1#2.#3.#4\xint:%
{%
    \expandafter\calx_tae_bb\the\numexpr#2+8*#3\expandafter.%
    \the\numexpr(2*#2+7)/16\expandafter.%
    \the\numexpr#2+8-8*((2*#2+7)/16).#4\xint:#1%
}%
\def\calx_tae_bb #1.#2.#3.#4\xint:#5%
  {\expandafter\calx_tae_c\the\numexpr\calx_trta#2.#3.#4\xint:.#1.#5}%
\def\calx_tae_c #1.#2.#3#4%
{%
    \csname calx_tae\ifnum#2=\xint_c_i\ifnum#1<141421357 \ifnum#1=100000000
      o\else l\fi\else s\fi\else s\fi#4\if-#3m\else p\fi\endcsname #1.#2.%
}%
\expandafter\def\csname calx_taesap\endcsname #1.#2.%
  {\calx_tae_t#1.#2.\calx_tad_zs+}%
\expandafter\def\csname calx_taesam\endcsname #1.#2.%
  {\calx_tae_t#1.#2.\calx_tad_zs-}%
\expandafter\def\csname calx_taescp\endcsname #1.#2.%
  {\calx_tae_t#1.#2.\calx_tad_zc{900000000}-+}%
\expandafter\def\csname calx_taescm\endcsname #1.#2.%
  {\calx_tae_t#1.#2.\calx_tad_zm{900000000}++}%
\expandafter\def\csname calx_taelap\endcsname #1.#2.%
  {\calx_tae_l#1.\calx_tad_zc{900000000}-+}%
\expandafter\def\csname calx_taelam\endcsname #1.#2.%
  {\calx_tae_l#1.\calx_tad_zc{900000000}--}%
\expandafter\def\csname calx_taelcp\endcsname #1.#2.%
  {\calx_tae_l#1.\calx_tad_zs+}%
\expandafter\def\csname calx_taelcm\endcsname #1.#2.%
  {\calx_tae_l#1.\calx_tad_zm{1800000000}-+}%
\expandafter\def\csname calx_taeoap\endcsname #1.#2.%
  {\calx_tac_a F+1.0.100000001!\xint: a}%
```

```
\expandafter\def\csname calx_taeoam\endcsname #1.#2.%
  {\calx_tac_a F-1.0.100000001!\xint: a}%
\expandafter\def\csname calx_taeocp\endcsname #1.#2.%
  {\calx_tac_a F+1.0.100000001!\xint: c}%
\expandafter\def\csname calx_taeocm\endcsname #1.#2.%
  {\calx_tac_a F-1.0.100000001!\xint: c}%
\def\calx_tae_t #1.#2.%
  {\expandafter\calx_tae_u\the\numexpr100000000*1000000000/#1.#2.}%
\def\calx_tae_u #1.#2.%
{%
    \expandafter\calx_tad_s\the\numexpr\ifnum#1<1000000000 #1\else
      100000000\fi\expandafter.\the\numexpr\ifnum#1<1000000000 1\else
      2\fi-#2.1000000000.%
}%
\def\calx_tae_l #1.%
  {\expandafter\calx_tae_la\the\numexpr(#1-100000000)*1000000000/#1.#1.}%
\def\calx_tae_la #1.%
  {\expandafter\calx_tae_lb\the\numexpr\CALX_nd_c #10001122334\xint:.#1.}%
\def\calx_tae_lb #1.#2.#3.%
{%
    \expandafter\calx_tad_lc\the\numexpr(#3-100000000)*\CALX_pten{2*#1+1}%
      *100000000/#3\expandafter.\the\numexpr2000000000-#2.#1.%
}%
\def\calx_ttq_small #1%
{%
    \if I#1\calx_ttq_j\fi
    \calx_ttq_b #1%
}%
\def\calx_ttq_j\fi #1\calx_ttq_b #2\xint:{\fi\calx_tat_a #2\xint:}%
\def\calx_ttq_b F#1#2.#3.#4\xint:%
{%
    \ifnum#2=\xint_c_ \calx_ttq_k\fi
    \calx_ttq_ba #1#2.#3.#4\xint:%
}%
\def\calx_ttq_k\fi\calx_ttq_ba #1\xint:{\fi\calx_tat_a F#1\xint:}%
\def\calx_ttq_ba #1#2.#3.#4\xint:%
{%
    \expandafter\calx_ttq_bb\the\numexpr#2+8*#3\expandafter.%
    \the\numexpr(2*#2+7)/16\expandafter.%
    \the\numexpr#2+8-8*((2*#2+7)/16).#4\xint:#1%
}%
\def\calx_ttq_bb #1.#2.#3.#4\xint:#5%
  {\expandafter\calx_ttq_c\the\numexpr\calx_trta#2.#3.#4\xint:.#1.#5}%
\def\calx_ttq_c #1.#2.#3#4#5#6%
{%
    \csname calx_ttq\ifnum#2<\xint_c_ i\else\ifnum#2=\xint_c_
      \ifnum#1>414213562 ii\else i\fi\else\ifnum#2=\xint_c_i
      \ifnum#1<241421357 iv\else v\fi\else v\fi\fi\fi#6\endcsname #1.#2.#3%
}%
\expandafter\def\csname calx_ttqi1\endcsname{\calx_ttq_i\calx_tad_ys}%
\expandafter\def\csname calx_ttqii1\endcsname{\calx_ttq_ii\calx_tad_yc}%
\expandafter\def\csname calx_ttqiv1\endcsname{\calx_ttq_iv\calx_tad_yc}%
```

```
\expandafter\def\csname calx_ttqv1\endcsname{\calx_ttq_v\calx_tad_yc}%
\expandafter\def\csname calx_ttqi0\endcsname{\calx_ttq_i\calx_tad_zs}%
\expandafter\def\csname calx_ttqii0\endcsname{\calx_ttq_ii\calx_tad_zc}%
\expandafter\def\csname calx_ttqiv0\endcsname{\calx_ttq_iv\calx_tad_zc}%
\expandafter\def\csname calx_ttqv0\endcsname{\calx_ttq_v\calx_tad_zc}%
\def\calx_ttq_i #1#2.#3.#4%
{%
    \expandafter\calx_ttq_x\the\numexpr2*#2*1000000000/(1000000000+
      \ifnum#3<-\xint_c_iv 0\else#2*#2/1000000000/\CALX_pten{-2*#3}\fi).%
    #3.#1#4%
}%
\def\calx_ttq_x #1.#2.%
{%
    \ifnum#1<1000000000 \expandafter\calx_ttq_y\else
      \expandafter\calx_ttq_z\fi #1.#2.%
}%
\def\calx_ttq_y #1.#2.{\calx_tad_s#1.#2.2000000000.}%
\def\calx_ttq_z #1.#2.%
{%
    \expandafter\calx_tad_s\the\numexpr#1/10\expandafter.%
    \the\numexpr#2+1.2000000000.%
}%
\def\calx_ttq_ii #1#2.#3.#4%
  {\expandafter\calx_ttq_w\the\numexpr#2*#2/1000000000.#1#4}%
\def\calx_ttq_w #1.#2#3%
{%
    \expandafter\calx_tad_s\the\numexpr(1000000000-#1)*1000000000/%
      (1000000000+#1).0.2000000000.#2{450000000}-#3%
}%
\def\calx_ttq_iv #1#2.#3.#4%
{%
    \expandafter\calx_tad_s\the\numexpr((#2-100000000)*(#2+100000000)%
      /100000000)*1000000000/(#2*#2/100000000+100000000).%
    0.2000000000.#1{450000000}+#4%
}%
\def\calx_ttq_v #1#2.#3.%
  {\expandafter\calx_ttq_r\the\numexpr100000000*1000000000/#2.#3.#1}%
\def\calx_ttq_r #1.#2.%
{%
    \expandafter\calx_ttq_u\the\numexpr\ifnum#1<1000000000 #1\else
      100000000\fi\expandafter.\the\numexpr\ifnum#1<1000000000 1\else
      2\fi-#2.%
}%
\def\calx_ttq_u #1.#2.#3#4%
{%
    \expandafter\calx_ttq_x\the\numexpr2*#1*1000000000/(1000000000+
      \ifnum#2<-\xint_c_iv 0\else#1*#1/1000000000/\CALX_pten{-2*#2}\fi).%
    #2.#3{900000000}-#4%
}%
```

35.7. Series kernel with column short products

Horner levels T_k = C_k - U T_(k+1) / 10^(4b) by column short products. Level (b, m, n): U has b limbs (more limbs of U_0 enter as extra precision), T_(k+1) m limbs, T_k n limbs. Columns c (weight 10^(4c) of U T) from b+n-1 down to b-2; columns b-1, b-2 are guards (carry only). Column text: terms `u*t+` with u = -(limb of U) balanced, t limb of T; the text of column c-1 comes from the text of column c by \calx_tZr (`u*t+` -> `t+u*`: each u takes the t of the next term), with the next u (f) opened in front and the next t (g) closing the last term. Stream of a level, inside the \numexpr of the outermost column:

    <first>\calx_tZs <U list>\calx_tZt <T items>\calx_tZw <static items>

U list `u!..u!0!0!0!0!` (MS first, negated, balanced, top limb up to 10^4 in modulus), T items `\calx_tZn x_top.\calx_tZl x. .. \calx_tZl0.` (\calx_tZl evaluates a balanced limb where the item enters a text), static item of column c: `C_c!<macro of c-1><carry macro of c-1>`. Column value x: the parent adds round(x/10^4) (\calx_tZu, \calx_tZg) and the limb x - 10^4 round(x/10^4) in [-5000,5000] is left after the parent's terminator; final levels: floor carry, limbs 0000..9999 (\calx_tZv: \calx_tZuf). Output after the outermost \numexpr:

    x_top.\calx_tZl x. ... (n limbs, MS first)

|u| <= 5000 (top <= 10^4), |t| <= 10^4, <= 28 terms: |x| < 1.5e9.

```
\let\calx_tZs\relax \let\calx_tZt\relax \let\calx_tZw\relax \let\calx_tZq\relax
\def\calx_tZa #1\calx_tZs#2!#3\calx_tZt#4.#5\calx_tZw#6!#7#8%
{%
    #6+#1\expandafter#8\the\numexpr\expandafter#7\expanded
    {#2*\calx_tZr#1\calx_tZx*#4.+}\calx_tZs#3\calx_tZt#5\calx_tZw
}%
\def\calx_tZb #1\calx_tZs#2!#3\calx_tZt#4.#5\calx_tZw#6!#7#8%
{%
    #6+\expandafter#8\the\numexpr\expandafter#7\expanded{#2*#4.+}%
    \calx_tZs#3\calx_tZt#5\calx_tZw
}%
\def\calx_tZc #1\calx_tZs#2\calx_tZt#3\calx_tZw#4!#5#6%
   {#4+\expandafter#6\the\numexpr#5\calx_tZs#2\calx_tZt#3\calx_tZw}%
\def\calx_tZz #1\calx_tZs#2\calx_tZw#3!{#3+#10.}%
```

\calx_tZad: as \calx_tZa when U is used up (f = 0): the first term's t is dropped instead of opening a zero term in front (\calx_tZrf)

```
\def\calx_tZad #1\calx_tZs#2!#3\calx_tZt#4.#5\calx_tZw#6!#7#8%
{%
    #6+#1\expandafter#8\the\numexpr\expandafter#7\expanded
    {\calx_tZrf#1\calx_tZx*#4.+}\calx_tZs#3\calx_tZt#5\calx_tZw
}%
\def\calx_tZrf #1*#2+{#1*\calx_tZr}%
\def\calx_tZr #1*#2+{#2+#1*\calx_tZr}%
\def\calx_tZx*\calx_tZr{}%
\def\calx_tZu #1.{#1/10000.\calx_tZl#1.}%
\def\calx_tZg #1.{#1/10000.}%
\def\calx_tZl #1.{\the\numexpr#1-#1/10000*10000\relax}%
\def\calx_tZn #1.{#1}%
```

negating variants: a level whose output feeds a level computing C + U T (instead of C - U T) uses the carry \calx_tZun (limbs negated by \calx_tZln); the next level is \calx_tZvn (top limb negated by \calx_tZnn)

```
\def\calx_tZun #1.{#1/10000.\calx_tZln#1.}%
\def\calx_tZln #1.{\the\numexpr#1/10000*10000-#1\relax}%
\def\calx_tZnn #1.{\the\numexpr-#1\relax}%
```

final variant: floor carry, limb as 4 digits followed by !

```
\def\calx_tZuf #1.{\expandafter\calx_tZuh\the\numexpr#1-#1/10000*10000.#1.}%
\def\calx_tZuh #1.%
   {\ifnum#1<\xint_c_ \expandafter\calx_tZum\else\expandafter\calx_tZup\fi#1.}%
\def\calx_tZum #1.#2.{(#2/10000-1).\calx_tZlm#1.}%
\def\calx_tZup #1.#2.{#2/10000.\calx_tZlp#1.}%
\def\calx_tZlm #1.{\the\numexpr10000+#1!}%
\def\calx_tZlp #1.{\expandafter\xint_gobble_i\the\numexpr10000+#1!}%
```

driver: <output>\calx_tZq{<U list>}<next descriptor>
\calx_tZv{<first>}{<static items>} or a client's final macro, which receives {<U list>}<output>\calx_tZq.

```
\def\calx_tZd #1\calx_tZq#2#3{#3{#2}#1\calx_tZq}%
\def\calx_tZv #1#2\calx_tZq#3#4%
{%
    \expandafter\calx_tZd\the\numexpr#3\calx_tZs#1\calx_tZt\calx_tZn#2%
    \calx_tZl0.\calx_tZw#4\calx_tZq{#1}%
}%
\def\calx_tZvn #1#2\calx_tZq#3#4%
{%
    \expandafter\calx_tZd\the\numexpr#3\calx_tZs#1\calx_tZt\calx_tZnn#2%
    \calx_tZl0.\calx_tZw#4\calx_tZq{#1}%
}%
```

level whose T items are given whole (no \calx_tZn prefix), U pad #5 (square level of sin/cos); carry \calx_tZuu: limbs evaluated as `v!' (U list)

```
\def\calx_tZvq #1#2\calx_tZq#3#4#5%
{%
    \expandafter\calx_tZd\the\numexpr#3\calx_tZs#1#5\calx_tZt#2%
    \calx_tZl0.\calx_tZw#4\calx_tZq{#1}%
}%
\def\calx_tZuu #1.{#1/10000.\calx_tZlu#1.}%
\def\calx_tZlu #1.{\the\numexpr#1-#1/10000*10000!}%
```

exact product level (all columns kept, C = 0): T negated as in \calx_tZvn, pads from the descriptor: \calx_tZvp{<first>}{<static>}{<U pad>}{<T pad>}

```
\def\calx_tZvp #1#2\calx_tZq#3#4#5#6%
{%
    \expandafter\calx_tZd\the\numexpr#3\calx_tZs#1#5\calx_tZt\calx_tZnn#2%
    \calx_tZl0.#6\calx_tZw#4\calx_tZq{#1}%
}%
```

load time: \calx_tZlevp{b}{m}<carry macro>: product of U (b limbs) by T (m limbs), columns b+m-1 down to 0, all limbs output (carry macro); columns x < m (U used up for the text of x-1) use \calx_tZad

```
\def\calx_tZlevp #1#2#3%
{%
    \noexpand\calx_tZvp{\noexpand\calx_tZb}%
    {\expandafter\calx_tZlpv\the\numexpr#1+#2-1\expandafter.\the\numexpr#2.#3}%
    {\expandafter\calx_tZzr\the\numexpr#2-1.}%
    {\expandafter\calx_tZlpz\the\numexpr#1-1.}%
}%
\def\calx_tZlpv #1.#2.#3% column #1 (m = #2): C = 0, macro of #1-1, carry
{%
    0!\ifnum#1=\xint_c_i\noexpand\calx_tZz\else
```

```
      \ifnum#1>#2 \noexpand\calx_tZa\else\noexpand\calx_tZad\fi\fi\noexpand#3%
    \ifnum#1=\xint_c_i 0!\else\expandafter\calx_tZlpv\the\numexpr#1-1.#2.#3\fi
}%
\def\calx_tZlpz #1.{\ifnum#1>\xint_c_ \noexpand\calx_tZl0.\expandafter\calx_tZlpz
   \the\numexpr#1-1\expandafter.\fi}%
```

list helpers: \calx_tZrev <l!..l!>\calx_tZq! reverses (inside \expanded); \calx_tZzr <k>. k zero limbs 0!; \calx_tZrep <k>. k limbs 9999!; \calx_tZtn c1!c2!..\relax -> c1.\calx_tZn c2 .. (T items, load time)

```
\def\calx_tZrev #1\calx_tZq!{\calx_tZrw#1\calx_tZq!\relax}%
\def\calx_tZrw #1!#2\relax{\calx_tZgob#1\calx_tZre\calx_tZq\calx_tZrw#2\relax#1!}%
\def\calx_tZre\calx_tZq\calx_tZrw#1\relax\calx_tZq!{}%
\def\calx_tZzr #1.{\ifnum#1>\xint_c_ 0!\expandafter\calx_tZzr
   \the\numexpr#1-1\expandafter.\fi}%
\def\calx_tZrep #1.{\ifnum#1>\xint_c_ 9999!\expandafter\calx_tZrep
   \the\numexpr#1-1\expandafter.\fi}%
\def\calx_tZtn #1!#2\relax{#1\calx_tZtm#2\relax}%
\def\calx_tZtm #1\relax{\if\relax\detokenize{#1}\relax\else\calx_tZtk#1\relax\fi}%
\def\calx_tZtk #1!#2\relax{.\noexpand\calx_tZn#1\calx_tZtm#2\relax}%
```

start: \calx_tZst{<T_N limb>}{<descriptors, final macro last>}{<U list>}

```
\def\calx_tZst #1#2#3{\calx_tZd#1.\calx_tZq{#3}#2}%
```

U list from limbs u_1!..u_b!\calx_tZq! (MS first, u_1 signed, others 0..9999): -(u_i - 10^4 [u_i >= 5000] + [u_(i+1) >= 5000]), top limb -(u_1 + [u_2 >= 5000]) kept whole; four zero limbs appended.

```
\def\calx_tZgob #1\calx_tZq{}%
\def\calx_tZbt #1!#2!{\the\numexpr0-(#1)-#2/10000!\calx_tZbn#2!}%
\def\calx_tZbn #1!#2!%
   {\calx_tZgob#2\calx_tZbe\calx_tZq\the\numexpr-#2/10000-#1+10000*(#1/10000)!%
    \calx_tZbn#2!}%
\def\calx_tZbe\calx_tZq\the\numexpr-#1/10000#2!\calx_tZbn\calx_tZq!%
   {\the\numexpr#2!0!0!0!0!}%
```

load time: descriptor of a level (inside \edef) \calx_tZlev{b}{m}{n}{<C limbs MS first, n items l!>}<carry macro> \calx_tZlevx<level macro>{b}{m}{n}{C}<carry macro> (\calx_tZvn)

```
\def\calx_tZlev{\calx_tZlevx\calx_tZv}%
\def\calx_tZlevx #1#2#3#4#5#6%
{%
    \noexpand#1{\expandafter\calx_tZmac\the\numexpr#2+#4-1.#2.#3.}%
    {\expandafter\calx_tZlv\the\numexpr#2+#4-1\expandafter.\expanded{#2.#3.#5}\relax#6}%
}%
\def\calx_tZmac #1.#2.#3.% macro of column #1 (b = #2, m = #3)
{%
    \ifnum#1>\numexpr#2+#3-1\relax\noexpand\calx_tZc\else
    \ifnum#1=\numexpr#2+#3-1\relax\noexpand\calx_tZb\else
    \ifnum#1<\numexpr#2-1\relax\noexpand\calx_tZz\else\noexpand\calx_tZa
    \fi\fi\fi
}%
\def\calx_tZlv #1.#2.#3.#4\relax#5% column #1 >= b-1: C!<macro c-1><carry c-1>
{%
    \ifnum#1<#2 0\else\calx_tZhd#4\relax\fi!%
    \expandafter\calx_tZmac\the\numexpr#1-1.#2.#3.%
    \ifnum#1>#2 \noexpand#5\else\noexpand\calx_tZg\fi
    \ifnum#1<#2 0!\else
      \expandafter\calx_tZlv\the\numexpr#1-1\expandafter.\the\numexpr#2%
```

```
      \expandafter.\the\numexpr#3\expandafter.\calx_tZtl#4\relax\relax#5%
    \fi
}%
\def\calx_tZhd #1!#2\relax{#1}%
\def\calx_tZtl #1!#2\relax{#2}%
\let\calx_tqW\relax
\def\calx_tgobW #1\calx_tqW{}%
```

35.8. Series kernels with three-limb passes (P above 40)

Generic path (P >= 9, |X| < 0.8): fixed point Horner on 4-digit limbs with the precision schedule of xinttrig (decreasing widths). u = X^2, S(u) = sum (-u)^k/(2k+1)!, C(u) = sum (-u)^k/(2k)!, c_k the coefficients. Scale 10^(4A) (A limbs) at all levels. Level k = N-1..0:
T_k = C_k - ceil(U_k T_{k+1} / 10^(4 b_k)), T_N = C_N,
C_k = floor(c_k 10^(4A)), U_k = floor(u 10^(4 b_k)) (b_k limbs, the top b_k limbs of U_0). T_k is kept on its n_k significant limbs (c_k 10^(4A) + 4 < 10^(4 n_k): the other limbs are zero), n_0 = A.

Error on S (or C), units 10^(-4A), u <= 0.64: level k contributes u^k (1 (C_k) + 1 (ceil) + 10^(4A-4 b_k) T_{k+1} 10^(-4A)), T_N 1, tail u^(N+1)/(2N+3)! (resp. (2N+2)!). The table below (generated by tools/trig_sched2.py with exact rationals, all P = 9..100) chooses N, A, b_k, n_k so that the total is <= 0.005 10^-P S_min (S_min = sin(0.8)/0.8, C_min = cos(0.8)): relative error <= 0.005 ulp before the single final rounding (sin X = round_P(X S), cos X = round_P(C)).

The c_k are computed at load time by the recurrence c_k = c_{k-1} / ((2k)(2k+1)) (resp. ((2k-1)(2k))) on floor(10^140 c_k) (no factorial; error < 2 10^-140, i.e. < 10^-30 unit), then truncated to 10^(4A).

Product: the level list has L = b_k + n_k items <u>,<a>! (LS first), item i stands for column i: items i < b_k are u_i,0!, items b_k + j are 0,c_j! (the limbs of C_k), then the sentinel \calx_tqW,0!. One pass handles three limbs t0, t1, t2 of T_{k+1} (missing ones are 0): with a window of the three previous u's, column i becomes x = a + carry - t0 u_i - t1 u_(i-1) - t2 u_(i-2); the pass emits u_(i-3),low(x)! (the u's move three items up, the columns stay: the next pass multiplies by t3, t4, t5). After ceil(m/3) passes the items hold the limbs of D = C_k 10^(4 b_k) - U_k T_{k+1} modulo 10^(4L) (carries out of item L-1 and the u's leaving the list only concern columns >= L), exact since 0 <= D < 10^(4L) (D 10^(-4 b_k) = T_k, n_k limbs; for k = 0, C_0 = 10^(4A) is 0 modulo 10^(4L)). T_k = items b_k..L-1. |x| < 3.0003 10^8: 1400000000 + x has 10 digits, the first six minus 140000 are the carry, the last four the low limb (carry by offset).

\calx_ttp <count>.<native blocks>/!: the limbs, LS first, as pairs <limb>,0!, exactly <count> of them (zero limbs added on top; a top limb beyond <count> is dropped: zero by construction). \calx_ttq: same, plain <limb>!.

```
\def\calx_ttp #1.%
  {\ifnum#1>\xint_c_ \expandafter\calx_ttpa\else\expandafter\calx_ttpg\fi
   #1.}%
\def\calx_ttpg #1.#2/!{}%
\def\calx_ttpa #1.#2!{\calx_tgsl#2\calx_ttpe/\calx_ttpb#1.#2!}%
\def\calx_ttpe/\calx_ttpb#1./!{\calx_ttpz#1.}%
\def\calx_ttpb #1.1#2#3#4#5#6#7#8#9!%
{%
    #6#7#8#9,0!\ifnum#1>\xint_c_i #2#3#4#5,0!\fi
    \expandafter\calx_ttp\the\numexpr#1-2.%
}%
\def\calx_ttpz #1.%
  {\ifnum#1>\xint_c_ 0000,0!\expandafter\calx_ttpz\the\numexpr#1-1.\fi}%
\def\calx_ttq #1.%
```

```
998   {\ifnum#1>\xint_c_ \expandafter\calx_ttqa\else\expandafter\calx_ttpg\fi
999    #1.}%
1000 \def\calx_ttqa #1.#2!{\calx_tgsl#2\calx_ttqe/\calx_ttqb#1.#2!}%
1001 \def\calx_ttqe/\calx_ttqb#1./!{\calx_ttqz#1.}%
1002 \def\calx_ttqb #1.1#2#3#4#5#6#7#8#9!%
1003 {%
1004     #6#7#8#9!\ifnum#1>\xint_c_i #2#3#4#5!\fi
1005     \expandafter\calx_ttq\the\numexpr#1-2.%
1006 }%
1007 \def\calx_ttqz #1.%
1008   {\ifnum#1>\xint_c_ 0000!\expandafter\calx_ttqz\the\numexpr#1-1.\fi}%
1009 \def\calx_ttr #1.%
1010   {\ifnum#1>\xint_c_ \expandafter\calx_ttra\else\expandafter\calx_ttpg\fi
1011    #1.}%
1012 \def\calx_ttra #1.#2!{\calx_tgsl#2\calx_ttre/\calx_ttrb#1.#2!}%
1013 \def\calx_ttre/\calx_ttrb#1./!{\calx_ttrz#1.}%
1014 \def\calx_ttrb #1.1#2#3#4#5#6#7#8#9!%
1015 {%
1016     0,#6#7#8#9!\ifnum#1>\xint_c_i 0,#2#3#4#5!\fi
1017     \expandafter\calx_ttr\the\numexpr#1-2.%
1018 }%
1019 \def\calx_ttrz #1.%
1020   {\ifnum#1>\xint_c_ 0,0000!\expandafter\calx_ttrz\the\numexpr#1-1.\fi}%
1021 \def\calx_tgsl #1/{}%
```

Kernels. \calx_tsin_kernel <s><D>.<e>.<blocks>\xint:, same for \calx_tcos_kernel. Schedule macro \calx_tts<s|c><P>: <b_0>.<A>.{d}{skip}{n}{C}{T_N} for level N-1, then \CALX_lv{d}{skip}{n}{C} for the next levels, \CALX_lend{}{}{}{}.

```
1022 \def\calx_tsin_fixed #1\xint:{\csname calx_ttfs0\endcsname\xint:#1\xint: s}%
1023 \def\calx_tcos_fixed #1\xint:{\csname calx_ttfc0\endcsname\xint:#1\xint: c}%
1024 \def\calx_tsin_scheduled #1\xint:%
1025   {\expandafter\calx_tsha\romannumeral`&&@\csname calx_ttss0\endcsname
1026    \xint:#1\xint: s}%
1027 \def\calx_tcos_scheduled #1\xint:%
1028   {\expandafter\calx_tsha\romannumeral`&&@\csname calx_ttsc0\endcsname
1029    \xint:#1\xint: c}%
```

u 10^(4 b_0) < 1 iff D + 8e + 2 b_0 <= 0: then sin X = round_P(X), cos X = 1 (b_0 >= A, 4A >= P + 3: u/6 < 0.005 10^-P).

Generic path, all on the kernel calx_tZ: X = N 10^(8e) (nb blocks), z = -e - nb >= 0 zero blocks, X' = N 10^(-8nb) in [10^-8, 1): its limbs are those of N (\calx_tshr), X = X' 10^(-8z). Square level (\calx_tZvq): U = X', T = X' on b_0+1 limbs (\calx_ttku, \calx_ttkn), output -X'^2 10^(4(b_0+1)) (|err| < 2.5 units); \calx_tshu: U_0 list = 4z zero limbs, these limbs, 0!0!0!0!. Series levels; sin and B: level 0 negated (\calx_tZun), \calx_tshw swaps in the X' list (carried in the stream after the descriptors), product level (\calx_tZvp, T = T_0 + 3 zero limbs): Y = X' T_0 10^12 on A+4 limbs, |err| < 1 unit (relative < 10^(-4A-4)), \calx_tshx; cos and A: level 0 floor carries, \calx_tshy. \calx_tshf: limbs dddd! (one zero limb in front, one appended if needed: <pairs>.<m0>.{pre}{post} static per P), calxtrig's \calx_trsa (m = m0 + z for sin), \calx_tsho<s|c|A|B>: one rounding to P (s, c) or P+4 with the continuation (A, B). Schedule \calx_tts<s|c><P> = <b_0>.<A>.{T_N}<S><Q><F>, three macros with one parameter (the dynamic header) whose bodies are the level descriptors: inserted by expansion, never grabbed. Stream: \calx_tsha <b_0>.<A>.{T_N}<S><Q><F>\xint: <s><D>.<e>.<blocks>\xint:<kind>.

```
1030 \def\calx_tsha #1.#2.#3\xint:#4#5.#6.%
1031 {%
```

```
    \ifnum\numexpr#5+8*#6+2*#1<\xint_c_i
      \xint_dothis\calx_tsha_tiny\fi
    \xint_orthat\calx_tsha_regular #1.#2.#3\xint:#4#5.#6.%
}%
\def\calx_tsha_tiny #1.#2.#3\xint:#4#5.#6.#7\xint:#8%
  {\csname calx_tshz#8\endcsname#4#5.#6.#7\xint:}%
\def\calx_tsha_regular #1.#2.#3\xint:#4#5.#6.#7\xint:#8%
{%
    \expandafter\calx_tshb\expanded{{\calx_tshr#7\calx_tZq!\relax}}%
      #3#1.\the\numexpr-#6-(#5+3)/8.#4#8%
}%
\def\calx_tshr #1#2!{\calx_tZgob#1\calx_tshre\calx_tZq\calx_tshrb#2!}%
\def\calx_tshrb #1#2#3#4#5#6#7#8!#9\relax{\calx_tshr#9\relax#1#2#3#4!#5#6#7#8!}%
\def\calx_tshre\calx_tZq\calx_tshrb!#1\relax{}%
\def\calx_tshzs #1\xint:{\CALX_fnround{\calx_ttrigdigits}{F#1}}%
\def\calx_tshzc #1\xint:{ F+1.0.100000001!}%
```

{X' limbs}{T_N}<S><Q><F><b_0>.<z>.: the square level (<Q>{header})

```
\def\calx_tshb #1#2#3#4#5#6.#7.%
{%
    \expandafter#4\expandafter{\expanded{\expandafter\calx_ttku\expandafter
      \calx_ttkn\the\numexpr#6+1\expandafter.\expanded{#1\calx_tZzr#6.0!}/%
      \noexpand\calx_tZq{\calx_tZbt#1\calx_tZq!}}}{#2}#3#5#7.%
}%
\def\calx_ttkn #1{\noexpand\calx_tZn#1.}%
```

after the square level: U_0 list, the series (<S>{header})

```
\def\calx_tshu #1#2\calx_tZq#3#4#5#6.%
{%
    \expandafter#4\expandafter{\expanded{\unexpanded{#3}.\noexpand\calx_tZq
      {\expandafter\calx_tZzr\the\numexpr4*#6.\calx_tshux#20!0!0!0!}}}{#1}#5#6.%
}%
\def\calx_tshux #1.{#1!}%
```

end of the series: <F>{output\calx_tZq{X' list}}

```
\def\calx_tshw #1#2\calx_tZq#3#4{#4{#2\calx_tZq{#3}}}%
\def\calx_tshx #1#2.#3\calx_tZq#4#5.#6#7{\calx_tshf#4#5.#6#7{#2}{#3}}%
\def\calx_tshy #1.#2\calx_tZq#3#4#5.#6#7{\calx_tshf#4 0.+#7{#1}{#2}}%
\def\calx_tshf #1.#2.#3#4#5.#6#7#8#9%
{%
    \expandafter\calx_tshg\expanded{#3\expandafter\xint_gobble_i
      \the\numexpr10000+#8!#9#4\relax#1.\the\numexpr#2+#5.}#6#7%
}%
\def\calx_tshg #1\relax#2.#3.#4#5%
  {\csname calx_tsho#5\expandafter\endcsname
   \expanded{{\calx_trsa#2.#3.\calx_tbp#4#1}}}%
\def\calx_tshos{\CALX_fnround{\calx_ttrigdigits}}%
\let\calx_tshoc\calx_tshos
\def\calx_tshol{\CALX_fnround{\calx_ldigits}}%
\def\calx_tshou #1{\expandafter\space\romannumeral`&&@#1}%
\def\calx_tshoA #1#2%
  {\expandafter#2\expandafter{\romannumeral0\CALX_fnround
   {\calx_ttrigdigits+4}{#1}}}%
\let\calx_tshoB\calx_tshoA
```

Load time: the coefficients floor(10^140 c_k), k = 0..40 (iterated floor division by (2k)(2k+1), resp. (2k-1)(2k)), are stored in calxdata.tex as 35 groups of 4 digits <dddd>!, LS first in \calx_tcd<s|c><k>, MS first in \calx_tcm<s|c><k> (generator gencalxdata.py). The limbs of floor(c_k 10^(4A)) are groups 35 - A + 1 to 35 (LS first): \calx_ttks<m> skips m groups (7 per call), \calx_ttkt<n> (\calx_ttktx: 7 per call) takes n groups, each given as argument to a converter (\calx_ttkr: item 0,<limb>!; \calx_ttkq: <limb>!; \calx_ttkm: <limb>), then drops the rest up to /.
\calx_ttcoef{<coefficient table prefix>}{<k>}{<A>}{<count>}<conv> (<conv> \calx_ttr or \calx_ttq): <count> limbs of floor(c_k 10^(4A)), LS first, as items (\calx_ttr) or limbs (\calx_ttq).

```
\def\calx_ttcoef #1#2#3#4#5%
{%
    \expandafter\calx_ttka\expandafter#5\the\numexpr35-(#3)\expandafter.%
    \the\numexpr#4\expandafter.%
    \romannumeral`&&@\csname #1\the\numexpr#2\endcsname/%
}%
\def\calx_ttka #1#2.%
  {\csname calx_ttks#2\expandafter\endcsname
   \csname calx_ttk\ifx#1\calx_ttr r\else q\fi\endcsname}%
\def\calx_ttkr #1{0,#1!}\def\calx_ttkq #1{#1!}\def\calx_ttkm #1{#1}%
\def\calx_ttku #1#2.%
  {\csname calx_ttkt\ifnum#2>\xint_c_vi x\else#2\fi\endcsname#1#2.}%
\expandafter\def\csname calx_ttktx\endcsname #1#2.#3!#4!#5!#6!#7!#8!#9!%
  {#1{#3}#1{#4}#1{#5}#1{#6}#1{#7}#1{#8}#1{#9}%
   \expandafter\calx_ttku\expandafter#1\the\numexpr#2-7.}%
\expandafter\def\csname calx_ttkt0\endcsname#1#2.#3/{}%
\expandafter\def\csname calx_ttkt1\endcsname#1#2.#3!#4/{#1{#3}}%
\expandafter\def\csname calx_ttkt2\endcsname#1#2.#3!#4!#5/{#1{#3}#1{#4}}%
\expandafter\def\csname calx_ttkt3\endcsname#1#2.#3!#4!#5!#6/%
  {#1{#3}#1{#4}#1{#5}}%
\expandafter\def\csname calx_ttkt4\endcsname#1#2.#3!#4!#5!#6!#7/%
  {#1{#3}#1{#4}#1{#5}#1{#6}}%
\expandafter\def\csname calx_ttkt5\endcsname#1#2.#3!#4!#5!#6!#7!#8/%
  {#1{#3}#1{#4}#1{#5}#1{#6}#1{#7}}%
\expandafter\def\csname calx_ttkt6\endcsname#1#2.#3!#4!#5!#6!#7!#8!#9/%
  {#1{#3}#1{#4}#1{#5}#1{#6}#1{#7}#1{#8}}%
\expandafter\def\csname calx_ttks0\endcsname#1{\calx_ttku#1}%
\expandafter\def\csname calx_ttks1\endcsname#1#2.#3!{\calx_ttku#1#2.}%
\expandafter\def\csname calx_ttks2\endcsname#1#2.#3!#4!{\calx_ttku#1#2.}%
\expandafter\def\csname calx_ttks3\endcsname#1#2.#3!#4!#5!{\calx_ttku#1#2.}%
\expandafter\def\csname calx_ttks4\endcsname#1#2.#3!#4!#5!#6!%
  {\calx_ttku#1#2.}%
\expandafter\def\csname calx_ttks5\endcsname#1#2.#3!#4!#5!#6!#7!%
  {\calx_ttku#1#2.}%
\expandafter\def\csname calx_ttks6\endcsname#1#2.#3!#4!#5!#6!#7!#8!%
  {\calx_ttku#1#2.}%
\expandafter\def\csname calx_ttks7\endcsname#1#2.#3!#4!#5!#6!#7!#8!#9!%
  {\calx_ttku#1#2.}%
\def\calx_ttmpa #1%
{%
    \expandafter\edef\csname calx_ttks#1\endcsname##1##2.##3!##4!%
      ##5!##6!##7!##8!##9!{\expandafter\noexpand
      \csname calx_ttks\the\numexpr#1-7\endcsname##1##2.}%
    \ifnum#1<35 \expandafter\calx_ttmpa\expandafter{\the\numexpr#1+1}\fi
```

```
}%
\calx_ttmpa{8}\let\calx_ttmpa\undefined
```

\calx_ttdef{<s|c>}{<P>}{<A>}{<N>}{<n_N>}{<b_0>}{<b>.<n>. ...}: the schedule \calx_tts<s|c><P>, built in one \edef (the levels are unrolled by \calx_ttdg inside it): <b_0>.<A>.{T_N}<S><Q><F> and the macros \calx_ttsS, \calx_ttsQ, \calx_ttsF<s|c><P>. \calx_ttlet <s|c>{<P>}{<P'>}: same schedule as for P'. The table (calxdata.tex, \calx_ttdata) has P >= 33 only: for P <= 32 sin, cos, asin use the fixed path \calx_ttf.

```
\def\calx_ttdef #1#2#3#4#5#6#7{\calx_ttdf{#1}{#2}{#3}{#4}{#5}{#6}#7/}%
\def\calx_ttdf #1#2#3#4#5#6#7/%
{%
    \calx_tglobal\expandafter\edef\csname calx_tts#1#2\endcsname
    {#6.#3.{\expandafter\calx_tZtn\expanded{\expandafter\calx_tZrev
       \expanded{\calx_ttcoef{calx_tcd#1}{#4}{#3}{#5}\calx_ttq}\calx_tZq!}\relax}%
     \expandafter\noexpand\csname calx_ttsS#1#2\endcsname
     \expandafter\noexpand\csname calx_ttsQ#1#2\endcsname
     \expandafter\noexpand\csname calx_ttsF#1#2\endcsname}%
    \calx_tglobal\expandafter\edef\csname calx_ttsS#1#2\endcsname##1%
    {\noexpand\calx_tZd##1\expandafter\calx_ttdg\csname calx_tZu\if#1sn\else f\fi
       \endcsname{#1}{#3}{#4-1}{#5}#7/\noexpand\calx_tshw}%
    \calx_tglobal\expandafter\edef\csname calx_ttsQ#1#2\endcsname##1%
    {\noexpand\calx_tZd##1\calx_tZlevx\calx_tZvq{\the\numexpr#6+1\relax}%
       {\the\numexpr#6+1\relax}{\the\numexpr#6+1\relax}%
       {\expandafter\calx_tZzr\the\numexpr#6+1.}\calx_tZuu
       {\expandafter\calx_tZzr\the\numexpr#6+2.}\noexpand\calx_tshu}%
    \calx_tglobal\expandafter\edef\csname calx_ttsF#1#2\endcsname##1%
    {\if#1s\noexpand\calx_tZd##1\calx_tZlevx\calx_tZvp{\the\numexpr#3+4\relax}%
       {\the\numexpr#3+4\relax}{\the\numexpr#3+4\relax}%
       {\expandafter\calx_tZzr\the\numexpr#3+4.}\calx_tZuf
       {\expandafter\calx_tZzr\the\numexpr#3+5.}%
       {\noexpand\calx_tZl0.\noexpand\calx_tZl0.\noexpand\calx_tZl0.}%
       \noexpand\calx_tshx{\ifodd#3 \the\numexpr(#3+5)/2.\the\numexpr(#3+3)/2.%
       {0000!}{}\else\the\numexpr(#3+6)/2.%
                      \the\numexpr(#3+4)/2.{0000!}{0000!}\fi}%
     \else
       \noexpand\calx_tshy##1{\ifodd#3
                              \the\numexpr(#3+3)/2.\the\numexpr(#3+1)/2.%
       {0000!}{0000!}\else\the\numexpr(#3+2)/2.\the\numexpr#3/2.{0000!}{}\fi}%
     \fi}%
}%
```

levels k = N-1..0 from the pairs b_k.n_k. (m = n_(k+1)); level 0 with C_0 = 10^(4A) on A+1 limbs, carry #1: \calx_tZun (sin: negated items for the product level), \calx_tZuf (cos: floor carries, digits)

```
\def\calx_ttdg #1#2#3#4#5#6.#7.#8/%
{%
    \ifnum\numexpr#4\relax>\xint_c_
      \calx_tZlev{#6}{#5}{#7}{\expandafter\calx_tZrev\expanded{\calx_ttcoef{calx_tcd#2}%
        {#4}{#3}{#7}\calx_ttq}\calx_tZq!}\calx_tZu
      \calx_ttdg#1{#2}{#3}{#4-1}{#7}#8/%
    \else
      \calx_tZlev{#6}{#5}{#3+1}{1!\expandafter\calx_tZzr\the\numexpr#3.}#1%
    \fi
}%
```

35.9. Series kernels with fixed-arity short products (P up to 40)

Fixed path (P with A <= 11): square short products in the manner of l3fp's __fp_fixed_mul. Limbs MS first in chunks of 4 braced groups {l1}{l2}{l3}{l4} (padded); a level macro \calx_tm<s>_<n> (generated by tools/trig_genfix.py) takes the U chunks, the T chunks (s limbs each, a = ceil(s/4) chunks) and the C chunks (n limbs) and computes, column e = n-1 .. -2 (weight 10^(4e) at scale 10^(4A)), in one nested \the\numexpr chain (innermost column first, its carry added to the outer one by \calx_tpk):

    2100000000 + c_e - sum_{s-i-j=e} u_i t_j + carry,

products by \calx_thd<d> (diagonal d of a pair of chunks). Columns e < -2 are omitted (their sum is < s 10^-4 < 0.002 unit: the budget is 2.02 units per level in trig_sched2.py), T_k = columns 0..n-1 of C 10^8 - sum, |T_k - (C - R)| < 1.002. Column value x: -1.09992 10^9 < x < 10^5 for s <= 11 products, 2100000000 + x has 10 digits, the first six minus 210000 are the carry (\calx_tpk), the last four the limb; the top column has no carry out (0 <= T_k < 10^(4n)) except for k = 0 where C_0 = 10^(4A) is 0 modulo 10^(4n): modular, exact.

```
\def\calx_tpk #1#2#3#4#5#6{+#1#2#3#4#5#6-210000!}%
\def\calx_tpg #1.%
  {\ifnum#1>\xint_c_iii \expandafter\calx_tpga\else\expandafter\calx_tpgb\fi
   #1.}%
\def\calx_tpga #1.#2!#3!#4!#5!%
  {{{#2}{#3}{#4}{#5}}\expandafter\calx_tpg\the\numexpr#1-4.}%
\def\calx_tpgb #1.{\csname calx_tpgr#1\endcsname}%
\expandafter\def\csname calx_tpgr0\endcsname#1;{}%
\expandafter\def\csname calx_tpgr1\endcsname#1!#2;{{{#1}{0000}{0000}{0000}}}%
\expandafter\def\csname calx_tpgr2\endcsname#1!#2!#3;{{{#1}{#2}{0000}{0000}}}%
\expandafter\def\csname calx_tpgr3\endcsname#1!#2!#3!#4;{{{#1}{#2}{#3}{0000}}}%
\def\calx_tbl #1#2#3#4#5#6#7#8{{#1#2#3#4}{#5#6#7#8}}%
```

U_0 = floor(u 10^(16 a_0)), a_0 = ceil(b_0/4) chunks, from its native blocks (padded on top with zero blocks).

```
\expandafter\def\csname calx_tud1\endcsname 1#1!1#2!#3\calx_tqW%
  {{\calx_tbl#2\calx_tbl#1}}%
\expandafter\def\csname calx_tud2\endcsname 1#1!1#2!1#3!1#4!#5\calx_tqW%
  {{\calx_tbl#4\calx_tbl#3}{\calx_tbl#2\calx_tbl#1}}%
\expandafter\def\csname calx_tud3\endcsname 1#1!1#2!1#3!1#4!1#5!1#6!#7\calx_tqW%
  {{\calx_tbl#6\calx_tbl#5}{\calx_tbl#4\calx_tbl#3}{\calx_tbl#2\calx_tbl#1}}%
```

u: M (D <= 40 digits, n <= 5 blocks) as 4a limbs (a = ceil(n/2) chunks, \calx_txm<n>), exact square M^2 (\calx_tsq<a>, 2a chunks; columns of at most 10 products: 10^9 + column + carry < 2 10^9 has 10 digits, \calx_tpkq), then U_0 = floor(M^2 10^(16(e + a_0))): the shift is a whole number of chunks (\calx_tfuq). D > 40: native square and shifts (\calx_tfug).

```
\def\calx_tfu #1.#2.%
{%
    \ifnum#2>40 \expandafter\calx_tfug\else\expandafter\calx_tfuk\fi #1.#2.%
}%
\def\calx_tfuk #1.#2.#3.#4\xint:%
{%
    \expandafter\calx_tfuq\expanded{{\csname
      calx_txm\the\numexpr(2*#2+7)/16\endcsname#4}#1.#3.%
      \the\numexpr((2*#2+7)/16)/2.}%
}%
```

Shift k = e + a_0 < a_0 (U_0 < 10^(16 a_0)): k >= 0 appends k zero chunks, k < 0 keeps the first 2a + k chunks of M^2 (\calx_tkp<j>); the delta = 2a + k - a_0 top chunks (zero) are gobbled, or -delta zero chunks are added on top.

```
\def\calx_tfuq #1#2.#3.#4.%
{%
    \expandafter\calx_tfur\the\numexpr#3+#2\expandafter.%
    \expanded{{\csname calx_tsq#4\endcsname#1}}#2.#4.%
}%
\def\calx_tfur #1.#2#3.#4.%
{%
    \expandafter\calx_tfus\expanded{\ifnum#1<\xint_c_
      {\csname calx_tkp\the\numexpr2*#4+#1\endcsname#2\calx_tqW}%
      \else{#2\csname calx_tzc#1\endcsname}\fi}%
    \the\numexpr2*#4+#1-#3.%
}%
\def\calx_tfus #1#2.%
   {\csname calx_t\ifnum#2<\xint_c_ zc\the\numexpr-#2\else
    gb#2\fi\endcsname#1}%
\expandafter\def\csname calx_tzc0\endcsname{}%
\expandafter\def\csname calx_tzc1\endcsname{{{0000}{0000}{0000}{0000}}}%
\expandafter\def\csname calx_tzc2\endcsname
  {{{0000}{0000}{0000}{0000}}{{0000}{0000}{0000}{0000}}}%
\expandafter\def\csname calx_tzc3\endcsname
  {{{0000}{0000}{0000}{0000}}{{0000}{0000}{0000}{0000}}%
   {{0000}{0000}{0000}{0000}}}%
\expandafter\def\csname calx_tgb0\endcsname{}%
\expandafter\let\csname calx_tgb1\endcsname\xint_gobble_i
\expandafter\let\csname calx_tgb2\endcsname\xint_gobble_ii
\expandafter\let\csname calx_tgb3\endcsname\xint_gobble_iii
\expandafter\let\csname calx_tgb4\endcsname\xint_gobble_iv
\expandafter\let\csname calx_tgb5\endcsname\xint_gobble_v
\expandafter\let\csname calx_tgb6\endcsname\xint_gobble_vi
\expandafter\let\csname calx_tgb7\endcsname\xint_gobble_vii
\expandafter\def\csname calx_tkp1\endcsname#1#2\calx_tqW{{#1}}%
\expandafter\def\csname calx_tkp2\endcsname#1#2#3\calx_tqW{{#1}{#2}}%
\expandafter\def\csname calx_tkp3\endcsname#1#2#3#4\calx_tqW{{#1}{#2}{#3}}%
\expandafter\def\csname calx_tkp4\endcsname#1#2#3#4#5\calx_tqW
  {{#1}{#2}{#3}{#4}}%
\expandafter\def\csname calx_tkp5\endcsname#1#2#3#4#5#6\calx_tqW
  {{#1}{#2}{#3}{#4}{#5}}%
\def\calx_tpkq #1#2#3#4#5#6{+#1#2#3#4#5#6-100000!}%
\def\calx_tfug #1.#2.#3.#4\xint:%
{%
    \expandafter\calx_tfua\romannumeral0\CALX_nshr{\numexpr-16*#3-16*#1}%
      {\romannumeral0\CALX_nshl{\numexpr16*#3+16*#1}{\romannumeral0\CALX_nmul
      {N+\the\numexpr(2*#2+7)/16.#4}{N+\the\numexpr(2*#2+7)/16.#4}}}%
    \xint:#1.%
}%
\def\calx_tfua N#1#2.#3\xint:#4.%
{%
    \csname calx_tud#4\endcsname#3100000000!100000000!100000000!%
    100000000!100000000!100000000!\calx_tqW
}%
```

\calx_tfv{<m>}{<C chunks>}{<T chunks>}{<U_0 chunks>}: one level. The product macro \calx_tm<s>_<n> takes T, C, then the first ceil(s/4) chunks of U_0 untruncated (no extraction of the top s

limbs: the error is not larger, see tools/trig_genfix.py) and gobbles the others up to \calx_tqW.

```
\def\calx_tfv #1#2#3#4%
  {\expandafter\calx_tfw\expanded{{#1#3#2#4\calx_tqW}}{#4}}%
\def\calx_tfw #1#2#3#4#5{#3{#4}{#5}{#1}{#2}}%
\def\calx_tfend #1#2#3#4{ {#3}}%
```

Kernel entry, schedule \calx_ttf<s|c><P>: <a_0>.<b_0>.<A>.{m}{C}{T_N} for level N-1, then \calx_tfv{m}{C} ..., \calx_tfend{}{}.

```
\def\calx_tfha #1.#2.#3.#4\xint:#5#6.#7.#8\xint:#9%
{%
    \ifnum\numexpr#6+8*#7+2*#2<\xint_c_i \calx_tfhj\fi
    \expandafter\calx_tfhb\expanded{{\calx_tfu #1.#6.#7.#8\xint:}}%
    #4\xint:{#9}{#5#6.#7.#8}{#3}%
}%
\def\calx_tfhj\fi\expandafter\calx_tfhb\expanded#1#2\xint:#3#4#5%
   {\fi\csname calx_tshz#3\endcsname#4\xint:}%
\def\calx_tfhb #1#2#3#4#5\xint:%
{%
    \expandafter\calx_tfhc\expanded{\romannumeral0\calx_tfv{#2}{#3}{#4}{#1}%
      #5}
}%
\def\calx_tfhc #1#2#3#4{\csname calx_tfk#2\endcsname{#1}#3.#4.}%
```

End: S (or C) = chunks, D = 16c digits (c = ceil(A/4) chunks, top limb >= 6900), as a native float F+16c.-2c.<blocks> (\calx_tsb<c>). sin X = round_P(X S) by the exact product of \CALX_fm_g (calxfp.sty), cos X = round_P(C).

For D <= 40, \calx_tfku replaces \CALX_fm_g (\calx_tfkt for D > 40): exact product M S (M as a = ceil(n/2) chunks by \calx_txm<n>, S = T_0 as c chunks) by \calx_tpm<a>_<c> (generated, tools/trig_genpm.py), limbs MS first, normalized by \calx_trsa (of the reduction B1: strip, reverse, exact D), e = e_X - 2c, then one \CALX_fround to P (same result as \CALX_fm_g: exact product, one rounding).

```
\def\calx_tsbb #1#2#3#4{1#3#4!1#1#2!}%
\expandafter\def\csname calx_tsb1\endcsname #1{\calx_tsbb#1}%
\expandafter\def\csname calx_tsb2\endcsname #1#2{\calx_tsbb#2\calx_tsbb#1}%
\expandafter\def\csname calx_tsb3\endcsname #1#2#3%
  {\calx_tsbb#3\calx_tsbb#2\calx_tsbb#1}%
\def\calx_tfks #1#2#3.#4.#5.#6.%
{%
    \ifnum#3>40 \xint_dothis\calx_tfkt\fi
    \xint_orthat\calx_tfku{#1}#2#3.#4.#5.#6.%
}%
\def\calx_tfku #1#2#3.#4.#5.#6.%
{%
    \CALX_fnround{\calx_ttrigdigits}{\csname
      calx_tpm\the\numexpr((2*#3+7)/16)/2\relax
      _\the\numexpr(#6+1)/4\expandafter\endcsname
      \expanded{\csname calx_txm\the\numexpr(2*#3+7)/16\endcsname#5#1;#2%
      \the\numexpr2*(((2*#3+7)/16)/2+(#6+1)/4).%
      \the\numexpr2*((#6+1)/4)-#4.}}%
}%
\def\calx_tpmh #1#2#3#4#5#6{\calx_tpmi}%
\def\calx_tpmi #1;#2#3.#4.{\expanded{\calx_trsa#3.#4.\calx_tbp#2#1}}%
\def\calx_tfkt #1#2#3.#4.#5.#6.%
{%
```

```
    \expandafter\CALX_fm_g\expanded{{#2}{#3}{#4}{#5}{+}%
      {\the\numexpr16*((#6+1)/4)}{-\the\numexpr2*((#6+1)/4)}%
      {\csname calx_tsb\the\numexpr(#6+1)/4\endcsname#1}}{\calx_ttrigdigits}%
}%
\def\calx_tfkc #1#2#3.#4.#5.#6.%
{%
    \CALX_fnround{\calx_ttrigdigits}{\expanded{F+\the\numexpr16*((#6+1)/4).%
      -\the\numexpr2*((#6+1)/4).%
      \csname calx_tsb\the\numexpr(#6+1)/4\endcsname#1}}%
}%
```

Load time: chunks of floor(c_k 10^(4A)) mod 10^(4n), n limbs.

```
\def\calx_tfsl #1#2#3#4#5#6#7#8%
  {\ifx\calx_tqW#1\else{#1#2#3#4#5#6#7#8}\expandafter\calx_tfsl\fi}%
\def\calx_tfpl #1#2%
  {\ifx\calx_tqW#1\else{\calx_tbl#1\calx_tbl#2}\expandafter\calx_tfpl\fi}%
\def\calx_ttkd #1#2#3#4%
{%
    \expandafter\calx_ttke\the\numexpr(#3)-(#4)\expandafter.%
    \the\numexpr#4\expandafter.%
    \romannumeral`&&@\csname calx_tcm#1\the\numexpr#2\endcsname/%
}%
\def\calx_ttke #1.{\csname calx_ttks#1\endcsname\calx_ttkm}%
\def\calx_tfco #1#2#3#4%
  {\expandafter\calx_tfcoa\expanded{1\calx_ttkd{#1}{#2}{#3}{#4}}\xint:#4.}%
\def\calx_tfcoa 1#1\xint:#2.%
{%
    \expandafter\calx_tfpl\expanded{\expandafter\calx_tfsl\expanded{#1%
      \xintReplicate{16*((#2+1)/4)-4*#2}{0}}%
      \calx_tqW\calx_tqW\calx_tqW\calx_tqW\calx_tqW\calx_tqW\calx_tqW\calx_tqW}%
    \calx_tqW\calx_tqW
}%
\def\calx_tfdef #1#2#3#4#5#6#7%
  {\ifnum#3>9 \expandafter\calx_tfdf\else\expandafter\calx_tgdf\fi
    {#1}{#2}{#3}{#4}{#5}{#6}#7/}%
\def\calx_tfdf #1#2#3#4#5#6#7.#8.#9/%
{%
    \edef\calx_ttacc{\noexpand\calx_tfha\the\numexpr(#6+1)/4.#6.#3.%
      {\expandafter\noexpand\csname calx_tm#7_#8\endcsname}%
      {\calx_tfco{#1}{#4-1}{#3}{#8}}{\calx_tfco{#1}{#4}{#3}{#5}}}%
    \calx_tfdg{#1}{#3}{#4-2}#9/%
    \calx_tglobal\expandafter\def\csname calx_ttf#1#2\expandafter\endcsname\expandafter
      {\calx_ttacc\calx_tfend{}{}}%
}%
\def\calx_tfdg #1#2#3#4/%
  {\if\relax\detokenize{#4}\relax\else\calx_tfdh{#1}{#2}{#3}#4/\fi}%
\def\calx_tfdh #1#2#3#4.#5.#6/%
{%
    \edef\calx_ttacc{\unexpanded\expandafter{\calx_ttacc}\noexpand\calx_tfv
      {\expandafter\noexpand\csname calx_tm#4_#5\endcsname}%
      {\calx_tfco{#1}{#3}{#2}{#5}}}%
    \calx_tfdg{#1}{#2}{#3-1}#6/%
}%
```

Flat path (A <= 9, that is P <= 32): the levels are the fixed-point multiply-subtract macros \CALX_xf<s><n> of calxfp.sty, limbs flat from one level to the next (no packing, no \expanded per level). \calx_ttf<s|c><P> is <handler><a_0>.<b_0>.<A>.<chain> (the handler is \calx_tfha for A > 9); the chain \calx_tg<s|c><P> takes the U_0 chunks (#1 to #3, padded with empty groups, up to \xint:):

    \CALX_xf<..>#1{T_N limbs}<U..><C> \CALX_xf<..>{#1}<U..><C> ...
    \CALX_xf<..>{#1}<U..><C> \calx_tge<A>{}

<U..> is #2 (b = 5..8) or #2#3 (b = 9), <C> the c arguments and text of the contract of \CALX_xf (\calx_tgcf ... \calx_tgcoa{n-1}{b-6}); T_N by \calx_tgcf ... \calx_tgtna. \calx_tgcf computes as \calx_tfco, its final macro being given in the stream. \calx_tge<A> receives T_0 (A limbs) then the finisher \calx_tgk<k> placed by \calx_tgha (k the kind letter, one \csname per call), then {X}{A}; it passes T_0 as chunks. \calx_tgks, \calx_tgkc call \calx_tfks, \calx_tfkc as \calx_tfhc does.

```
\def\calx_tgha #1.#2.#3.#4\xint:#5#6.#7.#8\xint:#9%
{%
    \ifnum\numexpr#6+8*#7+2*#2<\xint_c_i \calx_tghj#9\fi
    \expandafter#4\expanded{\calx_tfu #1.#6.#7.#8\xint:{}{}\xint:%
      \expandafter\noexpand\csname calx_tgk#9\endcsname}{#5#6.#7.#8}{#3}%
}%
\def\calx_tghj #1\fi #2\expanded#3#4#5%
  {\fi\csname calx_tshz#1\endcsname#4\xint:}%
\def\calx_tgl #1%
  {\ifcase#1\or a\or b\or c\or d\or e\or f\or g\or h\or i\or j\fi}%
\def\calx_tgsh #1#2%
  {\expandafter\noexpand\csname CALX_xf\calx_tgl{#1}\calx_tgl{#2}\endcsname}%
\def\calx_tgut #1{\ifnum#1>4 ####2\fi\ifnum#1>8 ####3\fi}%
\def\calx_tgcf #1#2#3#4#5%
  {\expandafter#5\expanded{1\calx_ttkd{#1}{#2}{#3}{#4}}\xint:}%
\def\calx_tgcoa 1#1\xint:{\calx_tgcb#1\xint:{}}%
\def\calx_tgcb #1#2#3#4#5\xint:#6#7#8%
{%
    \if\relax\detokenize{#5}\relax\calx_tgcp{#1#2#3#4}{#7}{#8}#6%
    \else\calx_tgcb#5\xint:{\calx_tgcp{#1#2#3#4}{#7}{#8}#6}{#7-1}{#8}\fi
}%
\def\calx_tgcp #1#2#3%
{%
    \ifnum\numexpr#2<\numexpr#3\relax{#1}%
    \else+#1;\ifnum\numexpr#2=\xint_c_{}\fi\fi
}%
\def\calx_tgtna 1#1\xint:{\calx_tgtl#1\xint:}%
\def\calx_tgtl #1#2#3#4#5\xint:%
  {{#1#2#3#4}\if\relax\detokenize{#5}\relax\else\calx_tgtl#5\xint:\fi}%
\def\calx_tgdf #1#2#3#4#5#6#7.#8.#9/%
{%
    \edef\calx_tgacc{\calx_tgsh{#7}{#8}####1%
      \calx_tgcf{#1}{#4}{#3}{#5}\calx_tgtna\calx_tgut{#7}%
      \calx_tgcf{#1}{#4-1}{#3}{#8}\calx_tgcoa{#8-1}{#7-6}}%
    \calx_tgdg{#1}{#3}{#4-2}#9/%
    \edef\calx_tgtmp{\noexpand\def
      \expandafter\noexpand\csname calx_tg#1#2\endcsname
      ####1####2####3#####4\noexpand\xint:{\unexpanded\expandafter
      {\calx_tgacc}\expandafter\noexpand\csname calx_tge#3\endcsname{}}}%
```

```
    \calx_tglobal\calx_tgtmp
    \calx_tglobal\expandafter\edef\csname calx_ttf#1#2\endcsname
      {\noexpand\calx_tgha\the\numexpr(#6+1)/4.#6.#3.%
       \expandafter\noexpand\csname calx_tg#1#2\endcsname}%
}%
\def\calx_tgdg #1#2#3#4/%
  {\if\relax\detokenize{#4}\relax\else\calx_tgdh{#1}{#2}{#3}#4/\fi}%
\def\calx_tgdh #1#2#3#4.#5.#6/%
{%
    \edef\calx_tgacc{\unexpanded\expandafter{\calx_tgacc}\calx_tgsh{#4}{#5}%
      {####1}\calx_tgut{#4}\calx_tgcf{#1}{#3}{#2}{#5}\calx_tgcoa{#5-1}{#4-6}}%
    \calx_tgdg{#1}{#2}{#3-1}#6/%
}%
\expandafter\def\csname calx_tge4\endcsname #1#2#3#4#5%
  {#5{{{#1}{#2}{#3}{#4}}}}%
\expandafter\def\csname calx_tge5\endcsname #1#2#3#4#5#6%
  {#6{{{#1}{#2}{#3}{#4}}{{#5}{0000}{0000}{0000}}}}%
\expandafter\def\csname calx_tge6\endcsname #1#2#3#4#5#6#7%
  {#7{{{#1}{#2}{#3}{#4}}{{#5}{#6}{0000}{0000}}}}%
\expandafter\def\csname calx_tge7\endcsname #1#2#3#4#5#6#7#8%
  {#8{{{#1}{#2}{#3}{#4}}{{#5}{#6}{#7}{0000}}}}%
\expandafter\def\csname calx_tge8\endcsname #1#2#3#4#5#6#7#8#9%
  {#9{{{#1}{#2}{#3}{#4}}{{#5}{#6}{#7}{#8}}}}%
\expandafter\def\csname calx_tge9\endcsname #1#2#3#4#5#6#7#8%
  {\calx_tgv{{{#1}{#2}{#3}{#4}}{{#5}{#6}{#7}{#8}}}}%
\def\calx_tgv #1#2#3{#3{#1{{#2}{0000}{0000}{0000}}}}%
\def\calx_tgks #1#2#3{\calx_tfks{#1}#2.#3.}%
\def\calx_tgkc #1#2#3{\calx_tfkc{#1}#2.#3.}%
```

Shortened chains: for u small the top levels are not needed. Bound: the error of the full chain at u = 0.64, so the worst case of the full schedule is kept. The data give \calx_tg<k><P> (top level Mtop = N), \calx_tg<k><P>_<M> (top level M, same b_k, n_k), then \calx_tfsel <k>{<P>}{<Mtop>}{<M>.<t_M>. ...}: the full one becomes \calx_tg<k><P>_<Mtop> and \calx_tg<k><P> the selector: with t the top limb of U_0 (t = floor(10^4 u)), the chain is the one of the smallest M with t < t_M (then u < t_M 10^-4 <= u_max(M)), else Mtop. The \calx_ttf<k><P>_<M> made by \calx_tgdf are removed.

```
\def\calx_tfsel #1#2#3#4%
{%
    \calx_tglobal\expandafter\let\csname calx_tg#1#2_#3\expandafter\endcsname
      \csname calx_tg#1#2\endcsname
    \calx_tfsu{#1}{#2}#4/%
    \calx_tglobal\expandafter\edef\csname calx_tgq#1#2\endcsname ##1##2\xint:%
      {\noexpand\csname calx_tg#1#2_\calx_tfsn{#3}#4/\noexpand\endcsname}%
    \calx_tglobal\expandafter\edef\csname calx_tg#1#2\endcsname ##1%
      {\expandafter\noexpand\csname calx_tgq#1#2\endcsname
       ##1\noexpand\xint:{##1}}%
}%
\def\calx_tfsn #1#2/%
  {\if\relax\detokenize{#2}\relax#1\else\calx_tfsm{#1}#2/\fi}%
\def\calx_tfsm #1#2.#3.#4/%
  {\noexpand\ifnum##1<#3 \calx_tfsn{#2}#4/\noexpand\else#1\noexpand\fi}%
\def\calx_tfsu #1#2#3/%
  {\if\relax\detokenize{#3}\relax\else\calx_tfsv{#1}{#2}#3/\fi}%
```

```
\def\calx_tfsv #1#2#3.#4.#5/%
{%
    \calx_tglobal\expandafter\let\csname calx_ttf#1#2_#3\endcsname\undefined
    \calx_tfsu{#1}{#2}#5/%
}%
```

35.10. Inverse functions; tangent, secant, sinc, degrees

\calxFpAsin{X}: asin of round_P(X), P = \calx_ttrigdigits <= 100, after xinttrig's @asin_l. With x = |X|, u = sqrt(1 - x^2) (x^2 exact, 1 - x^2 rounded to 2P+8 digits, square root to P+4 digits); if x > sqrt(2)/2 (round(10^9 x) > 707106781) the roles of x and u are exchanged and the result is pi/2 minus the core value. Core (x <= sqrt(2)/2, y the other one): tau = round(10^9 x) (\calx_tas_tau), T = 10^-9 round(tau S/10^9) with S from the arcsine series in one \numexpr (\calx_tasb, from xinttrig's \XINT_Arcsin_Ib), |asin x - T| about 10^-9; then

    D = x cos(T) - y sin(T) = sin(asin x - T),

cos(T) and sin(T) by the sin/cos kernels rounded to P+4 digits (P <= 32: flat path, finishers \calx_tgkA, \calx_tgkB; P > 32: series path \calx_tsha, finishers \calx_tshoA, \calx_tshoB), the two products exact (\CALX_fmulx), their difference exact (2P+24 digits); core = T + D, plus D^3 S(D^2) if P > 16 (S the arcsine series, round((P+9)/18) terms, Horner at P digits, \calx_tash<K>), the sum exact. T < 10^-8: core = x + x^3/6 (P <= 16), x + x^3 S(x^2) (P > 16, \calx_tas_ser). x = 1: core 0, roles exchanged. Operations whose operands have at most the target number of digits use the entries without input rounding (\CALX_fsubu, \CALX_fmulu, \CALX_faddu, \CALX_fsqru, \CALX_fdivu); the sums and differences of exact values are exact (\CALX_faddx, \CALX_fsubx).

\calx_tas_out{core}{n|s}{a}{b}{d}{g} (n: A = core, s: A = pi/2 - core; a, b integers as \numexpr text, b = +-1) gives round_P(f (a pi/2 + b A)) = round_P(f (k pi/2 + m core)): k = a, m = b (n) or k = a + b, m = -b (s). k = 0: no sum; else k pi/2 + m core exact (2P+70 digits, pi/2 to 130 digits rounded to 2P+70). f = 1/d (d = 1 or 2) times 180/pi if g = 1, by \calx_taso<2d+g-2>: 1/2 by the exact \CALX_fhalf, 180/pi by a product at P+8 digits; then one rounding to P. Signs by \calx_tas_m (product of two sign characters with the sign of a native). asin, asind: a = 0, b = +-1; acos, acosd: a = 1, b = -1 (x >= 0) or 1 (x < 0); acsc, asec: the same on y = 1/x (P+4 digits; x = 0: DivisionByZero), with 1 - y^2 = (x^2 - 1)/x^2 at P+4 digits from the exact x^2 - 1 and x^2 (\calx_tac_b; no cancellation near |x| = 1).

atan (xinttrig's @atan, no square root): t = |x|, T = t^2, z = 1/(1+T), w = 1 - T (P+8 digits); the core on x' = 2tz, y' = |w|z (P+4 digits, 2t exact; x'^2 + y'^2 = 1) gives A = 2 atan(t) if w > 0, pi - 2 atan(t) if w < 0 (w = 0: A = pi/2); then d = 2. Entry \calx_tat_a{c}{s}{g}: c pi/2 + s atan(x) (atan: c = 0, s = 1). Arg(x,y), pArg, atan2(y,x) and degree versions: the cases of xinttrig's @Arg, @pArg on the signs of y - x, y + x and y (\calx_tar<kind><y>x><y>-x><y sign>), q = y/x, x/y or x/(-y) at P+4 digits, then \calx_tat_b with the (c, s) of the case; Arg(0,0) = 0.

tan, cot, sec, csc, sinc and sind, cosd, tand, cotd, secd, cscd: \calx_tsc_a gives the quadrant q and the reduced argument u. Radians: |x| < 0.8: q = 0, u = x, else the reduction \calx_tred (|x| < 10^4) or \calx_trslow; both kernels (\calx_tsc_k): S = sin u, C = cos u rounded to P+4 digits (finishers A and B; tiny u: \calx_tshzA, \calx_tshzB). Degrees: \calx_tsc_f (below); sind, cosd, secd, cscd compute only the one of S, C they need. Then the finisher \calx_tfn<f>{q}{sign}{S}{C}{x}: sin x and cos x from S, C by the quadrant (\calx_tscS, \calx_tscC), one division at P+4 digits, one rounding to P.

Special values (IEEE 754 / C99 Annex F, calx zero being unsigned): NaN (I?) gives I?; sin, cos, tan, cot, sec, csc, the degree versions, asin, acos, asind, acosd of +-infinity signal InvalidOperation (I?, \calx_ttr_sp); sinc(+-inf) = 0; acsc(+-inf) = 0, asec(+-inf) = pi/2; atan(+-inf) = +-pi/2 (atand: +-90); Arg, pArg, atan2 (and degrees) with an infinite argument: the atan2 table of C99 (\calx_tari<kind><y> <x>, classes p, m infinities, f finite >= 0, g finite < 0): +-pi/2, +-pi/4, +-3pi/4, 0, pi, -pi, pArg brought to [0, 2pi). Poles (tand(90), cot(0), ...): division by zero, signal DivisionByZero, I+ or I-.

```
\edef\calx_tasPi{\romannumeral0\CALX_ffromraw
  {157079632679489661923132169163975144209858469968755291048747229615390820 31%
  43104499314017412671058533991074043256641153323546922305[-129]]}}%
\def\calx_tas_a #1%
{%
    \if I#1 \xint_dothis\calx_tas_a_special\fi
    \xint_orthat\calx_tas_a_finite #1%
}%
\def\calx_tas_a_special #1#2#3\xint:#4#5{\calx_ttr_sp#2}%
\def\calx_tas_a_finite #1#2#3\xint:#4#5%
  {\calx_tas_b #2#3\xint:#4#5}%
\def\calx_tas_b #1#2.%
{%
    \ifnum#2=\xint_c_ \xint_dothis\calx_tas_b_zero\fi
    \xint_orthat\calx_tas_b_regular #1#2.%
}%
\def\calx_tas_b_zero #1#2.#3.#4\xint:#5%
  {\calx_tas_out{F+0.0.}{n}{\if c#51\else0\fi}{\if c#5-\fi#11}{1}}%
\def\calx_tas_b_regular #1#2.#3.#4\xint:#5%
{%
    \expandafter\calx_tas_c\the\numexpr\calx_ttrigdigits.%
      F+#2.#3.#4\xint:%
    {\if c#51\else0\fi}{\if c#5-\fi#11}{1}%
}%
\def\calx_tas_err #1#2#3#4#5{\CALX_nsig{InvalidOperation}{#1}{}}%
\def\calx_tas_c #1.#2\xint:%
{%
    \expandafter\calx_tas_d\romannumeral0\CALX_fsubx
      {F+1.0.100000001!}{\CALX_fmulx{#2}{#2}}\xint:%
    #1.#2\xint:%
}%
\def\calx_tas_d F#1#2.%
{%
    \if-#1 \xint_dothis\calx_tas_d_invalid\fi
    \ifnum#2=\xint_c_ \xint_dothis\calx_tas_d_zero\fi
    \xint_orthat\calx_tas_d_regular #1#2.%
}%
\def\calx_tas_d_invalid #1#2.#3\xint:#4.#5\xint:%
  {\calx_tas_err{asin, acos: |x|>1}}%
\def\calx_tas_d_zero #1#2.#3\xint:#4.#5\xint:%
  {\calx_tas_out{F+0.0.}{s}}%
\def\calx_tas_d_regular #1#2.#3\xint:#4.#5\xint:%
{%
    \expandafter\calx_tas_e\romannumeral0\CALX_fsqrt
      {\the\numexpr#4+4\relax}{F#1#2.#3}\xint:#5\xint:%
}%
\def\calx_tas_e #1\xint:#2\xint:%
{%
    \expandafter\calx_tas_f\the\numexpr\calx_tas_tau#2\xint:\expandafter.%
    \the\numexpr\calx_ttrigdigits.{#2}{#1}%
}%
\def\calx_tas_f #1.#2.%
```

```
{%
    \ifnum#1>707106781 \calx_tas_sw\fi
    \calx_tas_k #1.#2.n%
}%
\def\calx_tas_sw\fi\calx_tas_k #1.#2.n#3#4%
  {\fi\expandafter\calx_tas_k\the\numexpr\calx_tas_tau#4\xint:.#2.s{#4}{#3}}%
\def\calx_tas_k #1.#2.%
{%
    \expandafter\calx_tas_l\the\numexpr
      (\expandafter\calx_tasb\the\numexpr#1*#1/\xint_c_e_ix.)*#1/\xint_c_e_ix.%
    #2.%
}%
\def\calx_tas_l #1.#2.%
{%
    \ifnum#1<\xint_c_x \calx_tas_tj\fi
    \expandafter\calx_tas_n\romannumeral0\CALX_ffromraw{#1[-9]}\xint:%
    #2.%
}%
\def\calx_tas_tj\fi\expandafter\calx_tas_n\romannumeral0\CALX_ffromraw
    #1\xint:#2.#3#4#5{\fi\calx_tas_tiny#2.#3{#4}}%
\def\calx_tas_n F#1\xint:#2.%
  {\calx_tas_kc c{#1}A\calx_tas_o{#1}#2.}%
\def\calx_tas_o #1#2#3.%
  {\calx_tas_kc s{#2}B\calx_tas_p{#1}{#2}#3.}%
\def\calx_tas_p #1#2#3#4.#5#6#7%
{%
    \expandafter\calx_tas_q\romannumeral0%
      \CALX_fsubx{\CALX_fmulx{#6}{#2}}{\CALX_fmulx{#7}{#1}}\xint:{#3}#4.#5%
}%
\def\calx_tas_q #1\xint:#2#3.%
{%
    \ifnum#3>16 \calx_tas_qc\fi
    \calx_tas_r{#1}{#2}#3.%
}%
\def\calx_tas_qc\fi\calx_tas_r#1#2#3.%
{%
    \fi\expandafter\calx_tas_r\expandafter{\romannumeral0\CALX_faddx
      {#1}{\calx_tas_ser{#3}{#1}}}{#2}#3.%
}%
\def\calx_tas_r #1#2#3.%
{%
    \expandafter\calx_tas_out\expandafter{\romannumeral0\CALX_faddx{F#2}{#1}}%
}%
\def\calx_tas_tiny #1.#2#3%
{%
    \expandafter\calx_tas_ty\romannumeral0\ifnum#1>16
      \expandafter\calx_tas_ser\else\expandafter\calx_tas_tq\fi{#1}{#3}\xint:%
    #3\xint:#1.#2%
}%
\def\calx_tas_tq #1#2%
  {\CALX_fdivu{#1}{\CALX_fmul{#1}{#2}{\CALX_fsqr{#1}{#2}}}{F+1.0.100000006!}}%
\def\calx_tas_ty #1\xint:F#2\xint:{\calx_tas_r{#1}{#2}}%
```

```
\def\calx_tas_out #1#2#3#4#5#6%
{%
    \CALX_fnround{\calx_ttrigdigits}{\csname calx_taso\the\numexpr2*#5+#6-2%
      \endcsname{\ifnum\numexpr\if s#2#3+(#4)\else#3\fi=\xint_c_
      \expandafter\xint_secondoftwo\else\expandafter\xint_firstoftwo\fi
      {\CALX_faddx
      {\CALX_fmul{\the\numexpr2*\calx_ttrigdigits+70\relax}{\calx_tasPi}%
        {\CALX_ffromraw{\the\numexpr\if s#2#3+(#4)\else#3\fi\relax[0]}}}}%
      {\xint_firstofone}%
      {\calx_tas_m{\ifnum\numexpr(#4)\if s#2*(-1)\fi<\xint_c_-\else+\fi}+#1}}}%
}%
\expandafter\def\csname calx_taso0\endcsname #1{#1}%
\expandafter\def\csname calx_taso1\endcsname #1%
  {\CALX_fmul{\the\numexpr\calx_ttrigdigits+8\relax}{#1}{\calx_ttrigR}}%
\expandafter\let\csname calx_taso2\endcsname\CALX_fhalf
\expandafter\def\csname calx_taso3\endcsname #1%
  {\CALX_fmul{\the\numexpr\calx_ttrigdigits+8\relax}{\CALX_fhalf{#1}}%
   {\calx_ttrigR}}%
\def\calx_tas_m #1#2F#3#4.%
  {\expanded{F\ifnum#4=\xint_c_+\else\if\if#1#2+\else-\fi#3+\else-\fi\fi#4.}}%
\def\calx_tac_a #1%
{%
    \if I#1\xint_dothis\calx_tac_i\fi
    \xint_orthat{\calx_tac_f #1}%
}%
\def\calx_tac_i #1#2\xint:#3%
{%
    \if ?#1\expandafter\xint_firstoftwo\else
      \expandafter\xint_secondoftwo\fi{ I?}%
      {\calx_tas_out{F+0.0.}{n}{\if c#31\else0\fi}{0}{1}{0}}%
}%
\def\calx_tac_f F#1#2.%
{%
    \ifnum#2=\xint_c_ \xint_dothis\calx_tac_f_zero\fi
    \xint_orthat\calx_tac_f_regular #1#2.%
}%
\def\calx_tac_f_zero #1#2.#3.#4\xint:#5%
  {\CALX_nsig{DivisionByZero}{acsc, asec of 0}{}}%
\def\calx_tac_f_regular #1#2.#3.#4\xint:#5%
{%
    \expandafter\calx_tac_b\romannumeral0\CALX_fdivu
      {\the\numexpr\calx_ttrigdigits+4\relax}{F+1.0.100000001!}%
      {F#1#2.#3.#4}\xint:#5{F#1#2.#3.#4}%
}%
\def\calx_tac_z #1.#2\xint:{#1}%
\def\calx_tac_b F#1#2\xint:#3#4%
{%
    \expandafter\calx_tas_d\romannumeral0\CALX_fdiv
      {\the\numexpr\calx_ttrigdigits+4\relax}%
      {\CALX_fsubx{\CALX_fmulx{#4}{#4}}{F+1.0.100000001!}}%
      {\CALX_fmulx{#4}{#4}}\xint:\calx_ttrigdigits.F+#2\xint:%
    {\if c#31\else0\fi}{\if c#3-\fi#11}{1}0%
```

```
}%
\def\calx_tat_a #1%
{%
    \if I#1\xint_dothis\calx_tat_i\fi
    \xint_orthat{\calx_tat_b #1}%
}%
\def\calx_tat_i #1#2\xint:#3#4#5%
{%
    \if ?#1\expandafter\xint_firstoftwo\else
      \expandafter\xint_secondoftwo\fi{ I?}%
      {\calx_tas_out{F+0.0.}{n}{(#3)+(#4)*(#11)}{0}{1}{#5}}%
}%
\def\calx_tat_b F#1#2.%
{%
    \ifnum#2=\xint_c_ \xint_dothis\calx_tat_b_zero\fi
    \xint_orthat\calx_tat_b_regular #1#2.%
}%
\def\calx_tat_b_zero #1#2.#3.#4\xint:#5#6#7%
  {\calx_tas_out{F+0.0.}{n}{2*(#5)}{0}{2}{#7}}%
\def\calx_tat_b_regular #1#2.#3.#4\xint:#5#6#7%
{%
    \expandafter\calx_tat_c\romannumeral0\CALX_fsqru
      {\the\numexpr\calx_ttrigdigits+8\relax}{F+#2.#3.#4}\xint:%
      F+#2.#3.#4\xint:#1{#5}{#6}{#7}%
}%
\def\calx_tat_c #1\xint:#2\xint:%
{%
    \expandafter\calx_tat_cc\romannumeral0\CALX_fdivu
      {\the\numexpr\calx_ttrigdigits+8\relax}{F+1.0.100000001!}%
      {\CALX_faddu{\the\numexpr\calx_ttrigdigits+8\relax}{F+1.0.100000001!}%
      {#1}}\xint:#1\xint:#2\xint:%
}%
\def\calx_tat_cc #1\xint:#2\xint:%
{%
    \expandafter\calx_tat_d\romannumeral0\CALX_fsubu
      {\the\numexpr\calx_ttrigdigits+8\relax}{F+1.0.100000001!}{#2}\xint:%
    #1\xint:%
}%
\def\calx_tat_d F#1#2.%
{%
    \ifnum#2=\xint_c_ \xint_dothis\calx_tat_d_zero\fi
    \xint_orthat\calx_tat_d_regular #1#2.%
}%
\def\calx_tat_d_zero #1#2.#3\xint:#4\xint:#5\xint:#6#7#8#9%
  {\calx_tas_out{F+0.0.}{s}{2*(#7)}{(#8)*(#61)}{2}{#9}}%
\def\calx_tat_d_regular #1#2.#3\xint:#4\xint:#5\xint:#6#7#8#9%
{%
    \expandafter\calx_tat_e\romannumeral0\CALX_fmul
      {\the\numexpr\calx_ttrigdigits+4\relax}{\CALX_faddx{#5}{#5}}{#4}\xint:%
      \romannumeral0\CALX_fmul{\the\numexpr\calx_ttrigdigits+4\relax}%
      {F+#2.#3}{#4}\xint:%
      {\if-#1(2*(#7)+(#8)*2*(#61))\else2*(#7)\fi}%
```

```
      {(#8)*(#61)\if-#1*(-1)\fi}{2}{#9}%
}%
\def\calx_tat_e #1\xint:#2\xint:{\expandafter\calx_tas_e#2\xint:#1\xint:}%
\def\calx_tar_a #1#2#3#4%
{%
    \expandafter\calx_tar_b\romannumeral0%
      \CALX_fround{\calx_ttrigdigits}{#3}%
    {#4}#1#2%
}%
\def\calx_tar_b #1\xint:#2%
{%
    \expandafter\calx_tar_c\romannumeral0%
      \CALX_fround{\calx_ttrigdigits}{#2}%
    #1\xint:%
}%
\def\calx_tar_c #1#2\xint:#3#4\xint:#5#6%
{%
    \if I#1\xint_dothis{\calx_tar_inf{#1#2}{#3#4}#5#6}\fi
    \if I#3\xint_dothis{\calx_tar_inf{#1#2}{#3#4}#5#6}\fi
    \xint_orthat{\expandafter\calx_tar_d\romannumeral0\CALX_faddx
      {#1#2}{#3#4}\xint:#1#2\xint:#3#4\xint:#5#6}%
}%
\def\calx_tar_d F#1#2.#3\xint:#4\xint:#5\xint:%
{%
    \expandafter\calx_tar_e\the\numexpr\calx_tar_st#1#2.\expandafter.%
    \romannumeral0\CALX_fsubx{#4}{#5}\xint:#4\xint:#5\xint:%
}%
\def\calx_ttr_sp #1%
{%
    \if ?#1\expandafter\xint_firstoftwo\else\expandafter\xint_secondoftwo\fi
    { I?}{\CALX_nsig{InvalidOperation}{trigonometric function of infinity}{}}%
}%
\def\calx_tsc_iz\fi\fi\calx_ttr_sp#1{\fi\fi\space F+0.0.}%
\def\calx_tar_inf #1#2#3#4%
{%
    \ifcsname calx_tari#3\calx_tarc#1\xint:\calx_tarc#2\xint:\endcsname
      \expandafter\xint_firstoftwo\else\expandafter\xint_secondoftwo\fi
    {\csname calx_tari#3\calx_tarc#1\xint:\calx_tarc#2\xint:\endcsname{#4}}%
    { I?}%
}%
\def\calx_tarc #1#2\xint:%
  {\if I#1\if ?#2n\else\if+#2p\else m\fi\fi\else\calx_tarf#2\xint:\fi}%
\def\calx_tarf #1#2.#3\xint:{\if-#1\ifnum#2>\xint_c_ g\else f\fi\else f\fi}%
\expandafter\def\csname calx_tariafm\endcsname%
  {\calx_tas_out{F+0.0.}{n}{2}{0}{1}}%
\expandafter\def\csname calx_tariafp\endcsname%
  {\calx_tas_out{F+0.0.}{n}{0}{0}{1}}%
\expandafter\def\csname calx_tariagm\endcsname%
  {\calx_tas_out{F+0.0.}{n}{-2}{0}{1}}%
\expandafter\def\csname calx_tariagp\endcsname%
  {\calx_tas_out{F+0.0.}{n}{0}{0}{1}}%
\expandafter\def\csname calx_tariamf\endcsname%
```

```
  {\calx_tas_out{F+0.0.}{n}{-1}{0}{1}}%
\expandafter\def\csname calx_tariamg\endcsname%
  {\calx_tas_out{F+0.0.}{n}{-1}{0}{1}}%
\expandafter\def\csname calx_tariamm\endcsname%
  {\calx_tas_out{F+0.0.}{n}{-3}{0}{2}}%
\expandafter\def\csname calx_tariamp\endcsname%
  {\calx_tas_out{F+0.0.}{n}{-1}{0}{2}}%
\expandafter\def\csname calx_tariapf\endcsname%
  {\calx_tas_out{F+0.0.}{n}{1}{0}{1}}%
\expandafter\def\csname calx_tariapg\endcsname%
  {\calx_tas_out{F+0.0.}{n}{1}{0}{1}}%
\expandafter\def\csname calx_tariapm\endcsname%
  {\calx_tas_out{F+0.0.}{n}{3}{0}{2}}%
\expandafter\def\csname calx_tariapp\endcsname%
  {\calx_tas_out{F+0.0.}{n}{1}{0}{2}}%
\expandafter\def\csname calx_taripfm\endcsname%
  {\calx_tas_out{F+0.0.}{n}{2}{0}{1}}%
\expandafter\def\csname calx_taripfp\endcsname%
  {\calx_tas_out{F+0.0.}{n}{0}{0}{1}}%
\expandafter\def\csname calx_taripgm\endcsname%
  {\calx_tas_out{F+0.0.}{n}{2}{0}{1}}%
\expandafter\def\csname calx_taripgp\endcsname%
  {\calx_tas_out{F+0.0.}{n}{0}{0}{1}}%
\expandafter\def\csname calx_taripmf\endcsname%
  {\calx_tas_out{F+0.0.}{n}{3}{0}{1}}%
\expandafter\def\csname calx_taripmg\endcsname%
  {\calx_tas_out{F+0.0.}{n}{3}{0}{1}}%
\expandafter\def\csname calx_taripmm\endcsname%
  {\calx_tas_out{F+0.0.}{n}{5}{0}{2}}%
\expandafter\def\csname calx_taripmp\endcsname%
  {\calx_tas_out{F+0.0.}{n}{7}{0}{2}}%
\expandafter\def\csname calx_taripppf\endcsname%
  {\calx_tas_out{F+0.0.}{n}{1}{0}{1}}%
\expandafter\def\csname calx_tarippg\endcsname%
  {\calx_tas_out{F+0.0.}{n}{1}{0}{1}}%
\expandafter\def\csname calx_taripppm\endcsname%
  {\calx_tas_out{F+0.0.}{n}{3}{0}{2}}%
\expandafter\def\csname calx_tarippp\endcsname%
  {\calx_tas_out{F+0.0.}{n}{1}{0}{2}}%
\def\calx_tar_st #1#2.{\ifnum#2=\xint_c_ 2\else\if-#1 0\else1\fi\fi}%
\def\calx_tar_e #1.F#2#3.#4\xint:#5\xint:#6\xint:#7%
{%
    \ifnum\numexpr#1+\calx_tar_st#2#3.=\xint_c_iv \calx_tar_zero\fi
    \csname calx_tar#7\ifnum\calx_tar_st#2#3.=\xint_c_i 1\else0\fi
      \ifnum#1=\xint_c_i 1\else0\fi\calx_tar_ys#5\xint:\endcsname{#6}{#5}%
}%
\def\calx_tar_ys F#1#2.#3\xint:{\if-#1\ifnum#2>\xint_c_ m\else p\fi\else p\fi}%
\def\calx_tar_zero\fi\csname#1\endcsname#2#3%
  {\fi\calx_tas_out{F+0.0.}{n}{0}{0}{1}}%
\def\calx_tar_q #1#2%
{%
    \expandafter\calx_tat_b\romannumeral0\CALX_fdivu
```

```
      {\the\numexpr\calx_ttrigdigits+4\relax}{#1}{#2}\xint:%
}%
\expandafter\def\csname calx_tara11m\endcsname #1#2%
  {\calx_tar_q{#1}{#2}{1}{-1}}%
\expandafter\def\csname calx_tara11p\endcsname #1#2%
  {\calx_tar_q{#1}{#2}{1}{-1}}%
\expandafter\def\csname calx_tara10m\endcsname #1#2%
  {\calx_tar_q{#2}{#1}{-2}{1}}%
\expandafter\def\csname calx_tara10p\endcsname #1#2%
  {\calx_tar_q{#2}{#1}{2}{1}}%
\expandafter\def\csname calx_tara01m\endcsname #1#2%
  {\calx_tar_q{#2}{#1}{0}{1}}%
\expandafter\def\csname calx_tara01p\endcsname #1#2%
  {\calx_tar_q{#2}{#1}{0}{1}}%
\expandafter\def\csname calx_tara00m\endcsname #1#2%
  {\calx_tar_q{#1}{\calx_tas_m-+#2}{-1}{1}}%
\expandafter\def\csname calx_tara00p\endcsname #1#2%
  {\calx_tar_q{#1}{\calx_tas_m-+#2}{-1}{1}}%
\expandafter\def\csname calx_tarp11m\endcsname #1#2%
  {\calx_tar_q{#1}{#2}{1}{-1}}%
\expandafter\def\csname calx_tarp11p\endcsname #1#2%
  {\calx_tar_q{#1}{#2}{1}{-1}}%
\expandafter\def\csname calx_tarp10m\endcsname #1#2%
  {\calx_tar_q{#2}{#1}{2}{1}}%
\expandafter\def\csname calx_tarp10p\endcsname #1#2%
  {\calx_tar_q{#2}{#1}{2}{1}}%
\expandafter\def\csname calx_tarp01m\endcsname #1#2%
  {\calx_tar_q{#2}{#1}{4}{1}}%
\expandafter\def\csname calx_tarp01p\endcsname #1#2%
  {\calx_tar_q{#2}{#1}{0}{1}}%
\expandafter\def\csname calx_tarp00m\endcsname #1#2%
  {\calx_tar_q{#1}{\calx_tas_m-+#2}{3}{1}}%
\expandafter\def\csname calx_tarp00p\endcsname #1#2%
  {\calx_tar_q{#1}{\calx_tas_m-+#2}{3}{1}}%
\def\calx_tsc_a #1%
{%
    \if I#1\xint_dothis\calx_tsc_i\fi
    \xint_orthat\calx_tsc_b
}%
\def\calx_tsc_i #1#2\xint:#3#4%
{%
    \ifx#3\calx_tfnsinc\if ?#1\else\calx_tsc_iz\fi\fi
    \calx_ttr_sp#1%
}%
\def\calx_tsc_b #1#2.%
{%
    \ifnum#2=\xint_c_ \xint_dothis\calx_tsc_b_zero\fi
    \xint_orthat\calx_tsc_b_regular #1#2.%
}%
\def\calx_tsc_b_zero #1#2.#3.#4\xint:#5#6%
  {#5{0}{#1}{F+0.0.}{F+1.0.100000001!}{F+0.0.}}%
\def\calx_tsc_b_regular #1#2.#3.#4\xint:#5#6%
```

```
1791    {\csname calx_tsc_#6\endcsname #2.#3.#4\xint:{#1}#5{F#1#2.#3.#4}}%
```

P <= 8, tan, cot, sec, csc, sinc (\calx_tsc_rq, \calx_tsc_rqr: as \calx_tsc_r, \calx_tsc_rr): \calx_tsc_kq takes q and r as \calx_tsc_k, |r| = A 10^(E-9) (\calx_trta), U9 = round(10^9 r^2) (\calx_tqkz), S9 = 10^9 sin(r)/r, C9 = 10^9 cos(r) (\calx_tsinaux, \calx_tcosaux), then one \numexpr per value (all quantities < 2^31):

```
     tan |r| = round(A S9 / C9) 10^(E-9)                    (\calx_tqkt)
     cot |r| = round(10^17 / round(A S9 / C9)) 10^(-E-8)     (\calx_tqku)
     sec r   = round(10^18 / C9) 10^-9                       (\calx_tqke)
     csc |r| = round(round(10^17 / A) 10^9 / S9) 10^(-E-8)   (\calx_tqkr)
```

times the sign f of the quadrant table \calx_tkq<t|u|v|w><q> (signs of x and r): tan = T, -1/T, cot = 1/T, -T for q even, odd; sec = 1/C, -1/S, -1/C, 1/S; csc = 1/S, 1/C, -1/S, -1/C (q = 0..3). Output by \calx_tdgo. r is never 0 (B1 gives 10^-W for R = 0).

sinc: |x| < 0.8 (\calx_tsc_kp, \calx_tsc_kps): sinc x = S9 10^-9, no division. Otherwise (\calx_tsc_kqs), |x| = A_x 10^(E_x-9), V = round(A S9/10^9) (q even, sign of r, minus for q = 2) or V = C9 (q odd, minus for q = 3), then one scaled division (\calx_tsc_kqf):

```
     sinc x = s round(V 10^9 / A_x) 10^(e-9)   (V < A_x)
     sinc x = s round(V 10^8 / A_x) 10^(e-8)   (V >= A_x)
```

with e = E - E_x (q even) or -E_x (q odd).

```
1792 \def\calx_tsc_r_regular #1.#2.#3\xint:%
1793 {%
1794     \ifnum\numexpr(#1+8*#2)*10\ifnum\numexpr#1+8*#2=\xint_c_
1795       +\calx_ttrig_top#3\xint:\fi>\xint_c_vii \calx_tsc_rr\fi
1796     \calx_tsc_k 0.F+#1.#2.#3\xint:%
1797 }%
1798 \def\calx_tsc_rq #1.#2.#3\xint:%
1799 {%
1800     \ifnum\numexpr(#1+8*#2)*10\ifnum\numexpr#1+8*#2=\xint_c_
1801       +\calx_ttrig_top#3\xint:\fi>\xint_c_vii \calx_tsc_rqr\fi
1802     \calx_tsc_kp 0.F+#1.#2.#3\xint:%
1803 }%
1804 \def\calx_tsc_rqr\fi\calx_tsc_kp 0.F+#1.#2.#3\xint:%
1805 {%
1806     \fi
1807     \ifnum\numexpr#1+8*#2>\xint_c_iv \calx_trjs\fi
1808     \calx_tsc_kd\calx_tred #1.#2.#3\xint:\xint:%
1809 }%
1810 \def\calx_tsc_kd #1%
1811 {%
1812     \ifx#1\calx_tred\expandafter\calx_tsc_kdd\else
1813       \expandafter\calx_tsc_kds\fi
1814 }%
1815 \def\calx_tsc_kds{\expandafter\calx_tsc_kq\romannumeral0\calx_trslow}%
1816 \def\calx_tsc_kdd #1.#2.#3\xint:\xint:%
1817 {%
1818     \expandafter\calx_tsc_kde\the\numexpr#1+8*#2\expandafter.%
1819     \the\numexpr(2*#1+7)/16\expandafter.%
1820     \the\numexpr#1+8-8*((2*#1+7)/16).#3\xint:%
1821 }%
1822 \def\calx_tsc_kde #1.#2.#3.#4\xint:%
1823 {%
1824     \expandafter\calx_tsc_kdf\romannumeral0\expandafter\calx_trq
1825     \the\numexpr\calx_trta#2.#3.#4\xint:.#1.%
```

```
}%
\def\calx_tsc_kdf #1.#2#3.#4.#5.#6.#7#8%
{%
    \csname calx_tkdl\ifx#8\calx_tfnsinc z\else\ifx#8\calx_tfntan t\else
      \ifx#8\calx_tfncot u\else\ifx#8\calx_tfnsec v\else w\fi\fi\fi\fi
    \endcsname #1.#2#3.#4.#5.#6.#7%
}%
\expandafter\def\csname calx_tkdlz\endcsname #1.#2#3.#4.#5.#6.#7#8%
  {\calx_tsc_kqy#3.#4.#5.#6.#1#2}%
\def\calx_tsc_kdl #1%
{%
    \expandafter\def\csname calx_tkdl#1\endcsname
      ##1.##2##3.##4.##5.##6.##7##8%
      {\calx_tsc_kqb##3.##4.#1##1##2##7}%
}%
\calx_tsc_kdl t\calx_tsc_kdl u\calx_tsc_kdl v\calx_tsc_kdl w%
\let\calx_tsc_kdl\undefined
\def\calx_tsc_kp #1.F#2#3.#4.#5\xint:#6#7%
{%
    \ifx#7\calx_tfnsinc \xint_dothis\calx_tsc_kps\fi
    \xint_orthat\calx_tsc_kq #1.F#2#3.#4.#5\xint:{#6}#7%
}%
\def\calx_tsc_kps #1.F#2#3.#4.#5\xint:#6#7#8%
{%
    \expandafter\calx_tsc_kpa\the\numexpr#3+8*#4\expandafter.%
    \the\numexpr(2*#3+7)/16\expandafter.%
    \the\numexpr#3+8-8*((2*#3+7)/16).#5\xint:%
}%
\def\calx_tsc_kpa #1.#2.#3.#4\xint:%
  {\expandafter\calx_tsc_kpb\the\numexpr\calx_trta#2.#3.#4\xint:.#1.}%
\def\calx_tsc_kpb #1.#2.%
{%
    \expandafter\calx_tdgo\the\numexpr
      \expandafter\calx_tsinaux\the\numexpr\calx_tqkz#1.#2...-9.%
}%
\def\calx_tsc_kq #1.F#2#3.#4.#5\xint:#6#7%
{%
    \ifx#7\calx_tfnsinc \xint_dothis\calx_tsc_kqs\fi
    \ifx#7\calx_tfntan \xint_dothis{\calx_tsc_kqx t}\fi
    \ifx#7\calx_tfncot \xint_dothis{\calx_tsc_kqx u}\fi
    \ifx#7\calx_tfnsec \xint_dothis{\calx_tsc_kqx v}\fi
    \ifx#7\calx_tfncsc \xint_dothis{\calx_tsc_kqx w}\fi
    \xint_orthat\calx_tsc_k #1.F#2#3.#4.#5\xint:{#6}#7%
}%
\def\calx_tsc_kqs #1.F#2#3.#4.#5\xint:#6#7#8%
  {\calx_tsc_kqt #8\xint:#1.#2#3.#4.#5\xint:}%
\def\calx_tsc_kqt F#1#2.#3.#4\xint:%
{%
    \expandafter\calx_tsc_kqu\the\numexpr#2+8*#3\expandafter.%
    \the\numexpr(2*#2+7)/16\expandafter.%
    \the\numexpr#2+8-8*((2*#2+7)/16).#4\xint:%
}%
```

```
\def\calx_tsc_kqu #1.#2.#3.#4\xint:%
  {\expandafter\calx_tsc_kqv\the\numexpr\calx_trta#2.#3.#4\xint:.#1.}%
\def\calx_tsc_kqv #1.#2.#3.#4#5.#6.#7\xint:%
{%
    \expandafter\calx_tsc_kqw\the\numexpr#5+8*#6\expandafter.%
    \the\numexpr(2*#5+7)/16\expandafter.%
    \the\numexpr#5+8-8*((2*#5+7)/16).#7\xint:#1.#2.#3#4%
}%
\def\calx_tsc_kqw #1.#2.#3.#4\xint:%
  {\expandafter\calx_tsc_kqy\the\numexpr\calx_trta#2.#3.#4\xint:.#1.}%
\def\calx_tsc_kqy #1.#2.%
  {\expandafter\calx_tsc_kqz\the\numexpr\calx_tqkz#1.#2..#1.#2.}%
\def\calx_tsc_kqz #1.#2.#3.#4.#5.#6%
{%
    \ifodd#6 \expandafter\calx_tsc_kqo\else\expandafter\calx_tsc_kqe\fi
    #1.#2.#3.#4.#5.#6%
}%
\def\calx_tsc_kqo #1.#2.#3.#4.#5.#6#7%
{%
    \expandafter\calx_tsc_kqf\the\numexpr\calx_tcosaux#1.\expandafter.%
    \the\numexpr\ifnum#6=\xint_c_iii -\fi1\expandafter.\the\numexpr-#5.#4.%
}%
\def\calx_tsc_kqe #1.#2.#3.#4.#5.#6#7%
{%
    \expandafter\calx_tsc_kqf\the\numexpr#2*(\calx_tsinaux#1.)/1000000000%
    \expandafter.\the\numexpr\ifnum#6=\xint_c_ii -1*\fi#71\expandafter.%
    \the\numexpr#3-#5.#4.%
}%
\def\calx_tsc_kqf #1.#2.#3.#4.%
{%
    \expandafter\calx_tdgo\the\numexpr#2*(\ifnum#1<#4 #1*1000000000/#4\else
      #1*100000000/#4\fi)\expandafter.\the\numexpr#3-\ifnum#1<#4 9\else
      8\fi.%
}%
\def\calx_tsc_kqx #1#2.F#3#4.#5.#6\xint:#7#8#9%
{%
    \expandafter\calx_tsc_kqa\the\numexpr#4+8*#5\expandafter.%
    \the\numexpr(2*#4+7)/16\expandafter.%
    \the\numexpr#4+8-8*((2*#4+7)/16).#6\xint:#1#2#3#7%
}%
\def\calx_tsc_kqa #1.#2.#3.#4\xint:%
  {\expandafter\calx_tsc_kqb\the\numexpr\calx_trta#2.#3.#4\xint:.#1.}%
\def\calx_tsc_kqb #1.#2.#3#4#5#6{\csname calx_tkq#3#4\endcsname#6#5#1.#2.}%
\expandafter\def\csname calx_tkqt0\endcsname#1#2{\calx_tqkt#11*#21.}%
\expandafter\def\csname calx_tkqt1\endcsname#1#2{\calx_tqku-1*#11*#21.}%
\expandafter\def\csname calx_tkqt2\endcsname#1#2{\calx_tqkt#11*#21.}%
\expandafter\def\csname calx_tkqt3\endcsname#1#2{\calx_tqku-1*#11*#21.}%
\expandafter\def\csname calx_tkqu0\endcsname#1#2{\calx_tqku#11*#21.}%
\expandafter\def\csname calx_tkqu1\endcsname#1#2{\calx_tqkt-1*#11*#21.}%
\expandafter\def\csname calx_tkqu2\endcsname#1#2{\calx_tqku#11*#21.}%
\expandafter\def\csname calx_tkqu3\endcsname#1#2{\calx_tqkt-1*#11*#21.}%
\expandafter\def\csname calx_tkqv0\endcsname#1#2{\calx_tqke1.}%
```

```
\expandafter\def\csname calx_tkqv1\endcsname#1#2{\calx_tqkr-1*#21.}%
\expandafter\def\csname calx_tkqv2\endcsname#1#2{\calx_tqke-1.}%
\expandafter\def\csname calx_tkqv3\endcsname#1#2{\calx_tqkr#21.}%
\expandafter\def\csname calx_tkqw0\endcsname#1#2{\calx_tqkr#11*#21.}%
\expandafter\def\csname calx_tkqw1\endcsname#1#2{\calx_tqke#11.}%
\expandafter\def\csname calx_tkqw2\endcsname#1#2{\calx_tqkr-1*#11*#21.}%
\expandafter\def\csname calx_tkqw3\endcsname#1#2{\calx_tqke-1*#11.}%
\def\calx_tqkz #1.#2.%
  {\ifnum#2<-\xint_c_iv 0\else#1*#1/1000000000/\CALX_pten{-2*#2}\fi}%
\def\calx_tqkt #1.#2.#3.%
  {\expandafter\calx_tqkta\the\numexpr\calx_tqkz#2.#3..#1.#2.#3.}%
\def\calx_tqkta #1.#2.#3.#4.%
{%
    \expandafter\calx_tdgo\the\numexpr#2*#3*(\calx_tsinaux#1.)%
      /(\calx_tcosaux#1.).#4-9.%
}%
\def\calx_tqku #1.#2.#3.%
  {\expandafter\calx_tqkua\the\numexpr\calx_tqkz#2.#3..#1.#2.#3.}%
\def\calx_tqkua #1.#2.#3.#4.%
{%
    \expandafter\calx_tdgo\the\numexpr#2*(1000000000*100000000/(#3*%
      (\calx_tsinaux#1.)/(\calx_tcosaux#1.))).-#4-8.%
}%
\def\calx_tqke #1.#2.#3.%
{%
    \expandafter\calx_tdgo\the\numexpr#1*(1000000000*1000000000/(%
      \expandafter\calx_tcosaux\the\numexpr\calx_tqkz#2.#3..)).-9.%
}%
\def\calx_tqkr #1.#2.#3.%
{%
    \expandafter\calx_tdgo\the\numexpr#1*((1000000000*100000000/#2)%
      *1000000000/(\expandafter\calx_tsinaux\the\numexpr\calx_tqkz#2.#3..)).%
    -#3-8.%
}%
\def\calx_tsc_rr\fi\calx_tsc_k 0.F+#1.#2.#3\xint:%
{%
    \fi
    \ifnum\numexpr#1+8*#2>\xint_c_iv \calx_trjs\fi
    \expandafter\calx_tsc_k\romannumeral0\calx_tred #1.#2.#3\xint:\xint:%
}%
```

Degree reduction, \calx_tsc_f<kernel selector>|x|: \calx_tsc_ks (sind, cscd), \calx_tsc_kc (cosd, secd), \calx_tsc_k (tand, cotd). With t = D + 8e (|x| < 10^t):
  - t < 2: q = 0, u = |x| pi/180 at P+4 digits (\calx_tsc_fz, constant \calx_tpdg<P>);
  - 2 <= t <= 6: q = round(|x|/90) in \numexpr from the nine leading digits (\calx_trta); q = 0: as above; else (\calx_tsc_fk) r = |x| - 90q at P+8 digits (exact: r has at most P digits), 90q a one-block native (\calx_tsc_fv), u = r pi/180 at P+4 digits, quadrant q mod 4;
  - t > 6: \calx_tsc_fo, exact reduction modulo 360 (\calx_tred_a).

```
\def\calx_tsc_d{\calx_tsc_f\calx_tsc_k}%
\def\calx_tsc_s{\calx_tsc_f\calx_tsc_ks}%
\def\calx_tsc_c{\calx_tsc_f\calx_tsc_kc}%
\let\calx_tsc_t\calx_tsc_d \let\calx_tsc_u\calx_tsc_d
\let\calx_tsc_v\calx_tsc_c \let\calx_tsc_w\calx_tsc_s
```

```
\def\calx_tsc_f #1#2.#3.%
{%
    \ifnum\numexpr#2+8*#3>\xint_c_vi \xint_dothis\calx_tsc_fo\fi
    \ifnum\numexpr#2+8*#3<\xint_c_ii \xint_dothis{\calx_tsc_fz 0.}\fi
    \xint_orthat\calx_tsc_fg #1#2.#3.%
}%
\def\calx_tsc_fg #1#2.#3.#4\xint:%
{%
    \expandafter\calx_tsc_fh\the\numexpr(\calx_trta
      \the\numexpr(2*#2+7)/16\relax.\the\numexpr#2+8-8*((2*#2+7)/16).%
      #4\xint:)/(90*\CALX_pten{9-#2-8*#3}).#1#2.#3.#4\xint:%
}%
\def\calx_tsc_fh #1.%
  {\ifnum#1=\xint_c_ \expandafter\calx_tsc_fz\else
   \expandafter\calx_tsc_fk\fi #1.}%
\def\calx_tsc_fz 0.#1#2\xint:%
{%
    \expandafter#1\expandafter0\expandafter.\romannumeral0%
    \CALX_fmul{\the\numexpr\calx_ttrigdigits+4\relax}{F+#2}%
      {\calx_tpdg}\xint:%
}%
\def\calx_tsc_fk #1.#2#3\xint:%
{%
    \expandafter#2\the\numexpr#1-4*((2*#1-3)/8)\expandafter.%
    \romannumeral0\CALX_fmul{\the\numexpr\calx_ttrigdigits+4\relax}%
      {\romannumeral0\CALX_fsub{\the\numexpr\calx_ttrigdigits+8\relax}%
        {F+#3}{\expandafter\calx_tsc_fv\the\numexpr100000000+90*#1.}}%
      {\calx_tpdg}\xint:%
}%
\def\calx_tsc_fv #1.{\expandafter\calx_tsc_fw\the\numexpr\calx_tnd#1..#1.}%
\def\calx_tsc_fw #1.#2.{F+#1.0.#2!}%
\def\calx_tsc_fo #1#2\xint:%
  {\expandafter#1\romannumeral0\calx_tred_a F+#2\xint:\xint:}%
```

One kernel: \calx_tsc_ks q.u (sine and cosecant) runs the sine kernel for q even, the cosine kernel for q odd; \calx_tsc_kc q.u (cosine and secant) the reverse. \calx_tsc_ku: sine kernel, finisher B, result in the S slot (\calx_tsc_pu); \calx_tsc_kv: cosine kernel, finisher A, result in the C slot (\calx_tsc_pv); the other slot is empty. u = 0: \calx_tsc_k. \calx_tsc_k: both kernels (u = 0: \calx_tsc_kz, S = 0, C = 1).

```
\def\calx_tsc_ks #1.F#2#3.%
{%
    \ifnum#3=\xint_c_ \calx_tsc_kj\fi
    \ifodd#1 \expandafter\calx_tsc_kv\else\expandafter\calx_tsc_ku\fi
    #1.F#2#3.%
}%
\def\calx_tsc_kc #1.F#2#3.%
{%
    \ifnum#3=\xint_c_ \calx_tsc_kj\fi
    \ifodd#1 \expandafter\calx_tsc_ku\else\expandafter\calx_tsc_kv\fi
    #1.F#2#3.%
}%
\def\calx_tsc_kj\fi #1\fi{\fi\calx_tsc_k}%
\def\calx_tsc_ku #1.F#2\xint:%
```

```
  {\calx_tas_kc s{#2}B\calx_tsc_pu #1.}%
\def\calx_tsc_kv #1.F#2\xint:%
  {\calx_tas_kc c{#2}A\calx_tsc_pv #1.}%
\def\calx_tsc_pu #1#2.#3#4{#4{#2}{#3}{#1}{}}%
\def\calx_tsc_pv #1#2.#3#4{#4{#2}{#3}{}{#1}}%
\def\calx_tsc_k #1.F#2#3.#4\xint:%
{%
    \ifnum#3=\xint_c_ \calx_tsc_kz\fi
    \calx_tas_kc c{#2#3.#4}A\calx_tsc_m{#2#3.#4}#1.%
}%
\def\calx_tsc_kz\fi\calx_tas_kc#1#2A\calx_tsc_m#3#4.#5#6%
  {\fi#6{#4}{#5}{F+0.0.}{F+1.0.100000001!}}%
\def\calx_tsc_m #1#2#3.%
  {\calx_tas_kc s{#2}B\calx_tsc_n{#1}#3.}%
\def\calx_tsc_n #1#2#3.#4#5{#5{#3}{#4}{#1}{#2}}%
\def\calx_tshzA #1\xint:#2{#2{F+1.0.100000001!}}%
\def\calx_tshzB #1\xint:#2%
  {\expandafter#2\expandafter{\romannumeral0\CALX_fnround
   {\calx_ttrigdigits+4}{F#1}}}%
\def\calx_tscS #1{\csname calx_tscs\the\numexpr#1\endcsname}%
\def\calx_tscC #1{\csname calx_tscc\the\numexpr#1\endcsname}%
\expandafter\def\csname calx_tscs0\endcsname #1#2#3{\calx_tas_m#1+#2}%
\expandafter\def\csname calx_tscs1\endcsname #1#2#3{\calx_tas_m#1+#3}%
\expandafter\def\csname calx_tscs2\endcsname #1#2#3{\calx_tas_m#1-#2}%
\expandafter\def\csname calx_tscs3\endcsname #1#2#3{\calx_tas_m#1-#3}%
\expandafter\def\csname calx_tscc0\endcsname #1#2{\calx_tas_m++#2}%
\expandafter\def\csname calx_tscc1\endcsname #1#2{\calx_tas_m+-#1}%
\expandafter\def\csname calx_tscc2\endcsname #1#2{\calx_tas_m+-#2}%
\expandafter\def\csname calx_tscc3\endcsname #1#2{\calx_tas_m++#1}%
\def\calx_tsc_q #1#2%
  {\CALX_fnround{\calx_ttrigdigits}{\CALX_fdiv
   {\the\numexpr\calx_ttrigdigits+4\relax}{#1}{#2}}}%
\def\calx_tfnsin #1#2#3#4#5%
  {\CALX_fnround{\calx_ttrigdigits}{\calx_tscS{#1}#2{#3}{#4}}}%
\def\calx_tfncos #1#2#3#4#5%
  {\CALX_fnround{\calx_ttrigdigits}{\calx_tscC{#1}{#3}{#4}}}%
\def\calx_tfntan #1#2#3#4#5{\calx_tsc_q{\calx_tscS{#1}#2{#3}{#4}}%
  {\calx_tscC{#1}{#3}{#4}}}%
\def\calx_tfncot #1#2#3#4#5{\calx_tsc_q{\calx_tscC{#1}{#3}{#4}}%
  {\calx_tscS{#1}#2{#3}{#4}}}%
\def\calx_tfnsec #1#2#3#4#5{\calx_tsc_q{F+1.0.100000001!}%
  {\calx_tscC{#1}{#3}{#4}}}%
\def\calx_tfncsc #1#2#3#4#5{\calx_tsc_q{F+1.0.100000001!}%
  {\calx_tscS{#1}#2{#3}{#4}}}%
\def\calx_tfnsinc #1#2#3#4#5{\calx_tfnsc #5\xint:{#1}{#2}{#3}{#4}{#5}}%
\def\calx_tfnsc F#1#2.%
{%
    \ifnum#2=\xint_c_ \xint_dothis\calx_tfnsc_zero\fi
    \xint_orthat\calx_tfnsc_regular #1#2.%
}%
\def\calx_tfnsc_zero #1#2.#3\xint:#4#5#6#7#8%
  { F+1.0.100000001!}%
```

```
\def\calx_tfnsc_regular #1#2.#3\xint:#4#5#6#7#8%
  {\calx_tsc_q{\calx_tscS{#4}#5{#6}{#7}}{#8}}%
\def\calx_tas_tau F+#1.#2.#3\xint:%
{%
    \ifnum\numexpr#1+8*#2<-\xint_c_viii 0\else
    \ifnum\numexpr#1+8*#2>\xint_c_ 1000000000\else
    (\calx_trta\the\numexpr(2*#1+7)/16\relax.%
      \the\numexpr#1+8-8*((2*#1+7)/16)\relax.#3\xint:)%
    /\CALX_pten{-#1-8*#2}\fi\fi
}%
\def\calx_tas_ka #1#2{\csname calx_ttf#10\endcsname\xint:#2\xint:}%
\def\calx_tas_kb #1#2%
{%
    \expandafter\calx_tsha\romannumeral`&&@\csname calx_tts#10\endcsname
    \xint:#2\xint:%
}%
\def\calx_tgkA #1#2#3#4%
{%
    \expandafter#4\expandafter{\romannumeral0\CALX_fnround
      {\calx_ttrigdigits+4}{\calx_tasch{#1}#3.}}%
}%
\def\calx_tgkB #1#2#3#4%
{%
    \expandafter#4\expandafter{\romannumeral0\CALX_fmul
      {\the\numexpr\calx_ttrigdigits+4\relax}{F#2}{\calx_tasch{#1}#3.}}%
}%
\def\calx_tasch #1#2.%
  {\expanded{F+\the\numexpr16*((#2+1)/4).-\the\numexpr2*((#2+1)/4).%
   \csname calx_tsb\the\numexpr(#2+1)/4\endcsname#1}}%
\def\calx_tas_ser #1#2%
  {\expandafter\calx_tas_sea\romannumeral0\CALX_fsqr{#1}{#2}\xint:{#1}{#2}}%
\def\calx_tas_sea #1\xint:#2#3%
{%
    \CALX_fmul{#2}{\CALX_fmul{#2}{#3}{#1}}%
      {\csname calx_tash\calx_tgl{\numexpr(#2+9)/18\relax}\endcsname{#2}{#1}}%
}%
\edef\calx_tasca{\romannumeral0\CALX_ffromraw
  {16666666666666666666666666666666666666666666666666666666666666666666666666%
  66666666666666666666666666666666666666666666666[-120]}}%
\edef\calx_tascb{\romannumeral0\CALX_ffromraw
  {75000000000000000000000000000000000000000000000000000000000000000000000000%
  00000000000000000000000000000000000000000000000[-121]}}%
\edef\calx_tascc{\romannumeral0\CALX_ffromraw
  {44642857142857142857142857142857142857142857142857142857142857142857142857%
  28571428571428571428571428571428571428571428571428571428571428571428571428[-121]}}%
\edef\calx_tascd{\romannumeral0\CALX_ffromraw
  {30381944444444444444444444444444444444444444444444444444444444444444444444%
  44444444444444444444444444444444444444444444444[-121]}}%
\edef\calx_tasce{\romannumeral0\CALX_ffromraw
  {22372159090909090909090909090909090909090909090909090909090909090909090909%
  09090909090909090909090909090909090909090909090909090909090909090909090909[-121]}}%
\edef\calx_tascf{\romannumeral0\CALX_ffromraw
```

```
  {1735276442307692307692307692307692307692307692307692307692307692307692%
  3076923076923076923076923076923076923076923076923076923076923076923[-121]}}%
\def\calx_tasha #1#2%
  {\calx_tasca}%
\def\calx_tashb #1#2%
  {\CALX_fadd{#1}{\calx_tasca}{\CALX_fmul{#1}{#2}{\calx_tascb}}}%
\def\calx_tashc #1#2%
  {\CALX_fadd{#1}{\calx_tasca}{\CALX_fmul{#1}{#2}{\CALX_fadd{#1}{\calx_tascb}%
   {\CALX_fmul{#1}{#2}{\calx_tascc}}}}}%
\def\calx_tashd #1#2%
  {\CALX_fadd{#1}{\calx_tasca}{\CALX_fmul{#1}{#2}{\CALX_fadd{#1}{\calx_tascb}%
   {\CALX_fmul{#1}{#2}{\CALX_fadd{#1}{\calx_tascc}{\CALX_fmul{#1}{#2}%
   {\calx_tascd}}}}}}}%
\def\calx_tashe #1#2%
  {\CALX_fadd{#1}{\calx_tasca}{\CALX_fmul{#1}{#2}{\CALX_fadd{#1}{\calx_tascb}%
   {\CALX_fmul{#1}{#2}{\CALX_fadd{#1}{\calx_tascc}{\CALX_fmul{#1}{#2}%
   {\CALX_fadd{#1}{\calx_tascd}{\CALX_fmul{#1}{#2}{\calx_tasce}}}}}}}}}%
\def\calx_tashf #1#2%
  {\CALX_fadd{#1}{\calx_tasca}{\CALX_fmul{#1}{#2}{\CALX_fadd{#1}{\calx_tascb}%
   {\CALX_fmul{#1}{#2}{\CALX_fadd{#1}{\calx_tascc}{\CALX_fmul{#1}{#2}%
   {\CALX_fadd{#1}{\calx_tascd}{\CALX_fmul{#1}{#2}{\CALX_fadd{#1}{\calx_tasce}%
   {\CALX_fmul{#1}{#2}{\calx_tascf}}}}}}}}}}}%
\def\calx_tasb #1.%
{%((((((((((((((((
(((((((((((((((((((((((((((((((((((((((((%
  #1*1849/1980+\xint_c_e_ix)*%
  1681/1806)*#1/\xint_c_e_ix+\xint_c_e_ix)*%
  1521/1640)*#1/\xint_c_e_ix+\xint_c_e_ix)*%
  1369/1482)*#1/\xint_c_e_ix+\xint_c_e_ix)*%
  1225/1332)*#1/\xint_c_e_ix+\xint_c_e_ix)*%
  1089/1190)*#1/\xint_c_e_ix+\xint_c_e_ix)*%
  961/1056)*#1/\xint_c_e_ix+\xint_c_e_ix)*%
  841/930)*#1/\xint_c_e_ix+\xint_c_e_ix)*%
  729/812)*#1/\xint_c_e_ix+\xint_c_e_ix)*%
  625/702)*#1/\xint_c_e_ix+\xint_c_e_ix)*%
  529/600)*#1/\xint_c_e_ix+\xint_c_e_ix)*%
  441/506)*#1/\xint_c_e_ix+\xint_c_e_ix)*%
  361/420)*#1/\xint_c_e_ix+\xint_c_e_ix)*%
  289/342)*#1/\xint_c_e_ix+\xint_c_e_ix)*%
  225/272)*#1/\xint_c_e_ix+\xint_c_e_ix)*%
  169/210)*#1/\xint_c_e_ix+\xint_c_e_ix)*%
  121/156)*#1/\xint_c_e_ix+\xint_c_e_ix)*%
  81/110)*#1/\xint_c_e_ix+\xint_c_e_ix)*%
  49/72)*#1/\xint_c_e_ix+\xint_c_e_ix)*%
  25/42)*#1/\xint_c_e_ix+\xint_c_e_ix)*%
  9/20)*#1/\xint_c_e_ix+\xint_c_e_ix)*%
  1/6)*#1/\xint_c_e_ix+\xint_c_e_ix
}%
```

Generated by tools/trig_genpm.py: \calx_tpm<a>_<c>, a, c = 1..3.

```
\expandafter\def\csname calx_tpm1_1\endcsname#1#2%
{%
\expandafter\calx_tpmh\the\numexpr 1000000000%
```

```
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpa#1#2%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpb#1#2%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpc#1#2%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpd#1#2%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpe#1#2%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpf#1#2%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpg#1#2\relax!%
}%
\expandafter\def\csname calx_tpm1_2\endcsname#1#2#3%
{%
\expandafter\calx_tpmh\the\numexpr 1000000000%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpa#1#2%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpb#1#2%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpc#1#2%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpd#1#2%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpe#1#2\calx_thpa#1#3%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpf#1#2\calx_thpb#1#3%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpg#1#2\calx_thpc#1#3%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpd#1#3%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpe#1#3%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpf#1#3%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpg#1#3\relax!%
}%
\expandafter\def\csname calx_tpm1_3\endcsname#1#2#3#4%
{%
\expandafter\calx_tpmh\the\numexpr 1000000000%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpa#1#2%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpb#1#2%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpc#1#2%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpd#1#2%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpe#1#2\calx_thpa#1#3%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpf#1#2\calx_thpb#1#3%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpg#1#2\calx_thpc#1#3%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpd#1#3%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpe#1#3\calx_thpa#1#4%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpf#1#3\calx_thpb#1#4%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpg#1#3\calx_thpc#1#4%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpd#1#4%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpe#1#4%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpf#1#4%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpg#1#4\relax!%
}%
\expandafter\def\csname calx_tpm2_1\endcsname#1#2#3%
{%
\expandafter\calx_tpmh\the\numexpr 1000000000%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpa#1#3%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpb#1#3%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpc#1#3%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpd#1#3%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpe#1#3\calx_thpa#2#3%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpf#1#3\calx_thpb#2#3%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpg#1#3\calx_thpc#2#3%
```

```
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpd#2#3%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpe#2#3%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpf#2#3%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpg#2#3\relax!%
}%
\expandafter\def\csname calx_tpm2_2\endcsname#1#2#3#4%
{%
\expandafter\calx_tpmh\the\numexpr 1000000000%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpa#1#3%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpb#1#3%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpc#1#3%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpd#1#3%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpe#1#3\calx_thpa#1#4%
  \calx_thpa#2#3%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpf#1#3\calx_thpb#1#4%
  \calx_thpb#2#3%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpg#1#3\calx_thpc#1#4%
  \calx_thpc#2#3%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpd#1#4\calx_thpd#2#3%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpe#1#4\calx_thpe#2#3%
  \calx_thpa#2#4%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpf#1#4\calx_thpf#2#3%
  \calx_thpb#2#4%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpg#1#4\calx_thpg#2#3%
  \calx_thpc#2#4%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpd#2#4%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpe#2#4%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpf#2#4%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpg#2#4\relax!%
}%
\expandafter\def\csname calx_tpm2_3\endcsname#1#2#3#4#5%
{%
\expandafter\calx_tpmh\the\numexpr 1000000000%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpa#1#3%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpb#1#3%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpc#1#3%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpd#1#3%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpe#1#3\calx_thpa#1#4%
  \calx_thpa#2#3%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpf#1#3\calx_thpb#1#4%
  \calx_thpb#2#3%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpg#1#3\calx_thpc#1#4%
  \calx_thpc#2#3%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpd#1#4\calx_thpd#2#3%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpe#1#4\calx_thpa#1#5%
  \calx_thpe#2#3\calx_thpa#2#4%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpf#1#4\calx_thpb#1#5%
  \calx_thpf#2#3\calx_thpb#2#4%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpg#1#4\calx_thpc#1#5%
  \calx_thpg#2#3\calx_thpc#2#4%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpd#1#5\calx_thpd#2#4%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpe#1#5\calx_thpe#2#4%
```

```
2281   \calx_thpa#2#5%
2282 \expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpf#1#5\calx_thpf#2#4%
2283   \calx_thpb#2#5%
2284 \expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpg#1#5\calx_thpg#2#4%
2285   \calx_thpc#2#5%
2286 \expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpd#2#5%
2287 \expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpe#2#5%
2288 \expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpf#2#5%
2289 \expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpg#2#5\relax!%
2290 }%
2291 \expandafter\def\csname calx_tpm3_1\endcsname#1#2#3#4%
2292 {%
2293 \expandafter\calx_tpmh\the\numexpr 1000000000%
2294 \expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpa#1#4%
2295 \expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpb#1#4%
2296 \expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpc#1#4%
2297 \expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpd#1#4%
2298 \expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpe#1#4\calx_thpa#2#4%
2299 \expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpf#1#4\calx_thpb#2#4%
2300 \expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpg#1#4\calx_thpc#2#4%
2301 \expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpd#2#4%
2302 \expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpe#2#4\calx_thpa#3#4%
2303 \expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpf#2#4\calx_thpb#3#4%
2304 \expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpg#2#4\calx_thpc#3#4%
2305 \expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpd#3#4%
2306 \expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpe#3#4%
2307 \expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpf#3#4%
2308 \expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpg#3#4\relax!%
2309 }%
2310 \expandafter\def\csname calx_tpm3_2\endcsname#1#2#3#4#5%
2311 {%
2312 \expandafter\calx_tpmh\the\numexpr 1000000000%
2313 \expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpa#1#4%
2314 \expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpb#1#4%
2315 \expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpc#1#4%
2316 \expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpd#1#4%
2317 \expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpe#1#4\calx_thpa#1#5%
2318   \calx_thpa#2#4%
2319 \expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpf#1#4\calx_thpb#1#5%
2320   \calx_thpb#2#4%
2321 \expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpg#1#4\calx_thpc#1#5%
2322   \calx_thpc#2#4%
2323 \expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpd#1#5\calx_thpd#2#4%
2324 \expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpe#1#5\calx_thpe#2#4%
2325   \calx_thpa#2#5\calx_thpa#3#4%
2326 \expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpf#1#5\calx_thpf#2#4%
2327   \calx_thpb#2#5\calx_thpb#3#4%
2328 \expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpg#1#5\calx_thpg#2#4%
2329   \calx_thpc#2#5\calx_thpc#3#4%
2330 \expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpd#2#5\calx_thpd#3#4%
2331 \expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpe#2#5\calx_thpe#3#4%
2332   \calx_thpa#3#5%
```

```
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpf#2#5\calx_thpf#3#4%
  \calx_thpb#3#5%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpg#2#5\calx_thpg#3#4%
  \calx_thpc#3#5%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpd#3#5%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpe#3#5%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpf#3#5%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpg#3#5\relax!%
}%
\expandafter\def\csname calx_tpm3_3\endcsname#1#2#3#4#5#6%
{%
\expandafter\calx_tpmh\the\numexpr 1000000000%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpa#1#4%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpb#1#4%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpc#1#4%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpd#1#4%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpe#1#4\calx_thpa#1#5%
  \calx_thpa#2#4%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpf#1#4\calx_thpb#1#5%
  \calx_thpb#2#4%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpg#1#4\calx_thpc#1#5%
  \calx_thpc#2#4%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpd#1#5\calx_thpd#2#4%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpe#1#5\calx_thpa#1#6%
  \calx_thpe#2#4\calx_thpa#2#5\calx_thpa#3#4%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpf#1#5\calx_thpb#1#6%
  \calx_thpf#2#4\calx_thpb#2#5\calx_thpb#3#4%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpg#1#5\calx_thpc#1#6%
  \calx_thpg#2#4\calx_thpc#2#5\calx_thpc#3#4%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpd#1#6\calx_thpd#2#5%
  \calx_thpd#3#4%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpe#1#6\calx_thpe#2#5%
  \calx_thpa#2#6\calx_thpe#3#4\calx_thpa#3#5%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpf#1#6\calx_thpf#2#5%
  \calx_thpb#2#6\calx_thpf#3#4\calx_thpb#3#5%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpg#1#6\calx_thpg#2#5%
  \calx_thpc#2#6\calx_thpg#3#4\calx_thpc#3#5%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpd#2#6\calx_thpd#3#5%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpe#2#6\calx_thpe#3#5%
  \calx_thpa#3#6%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpf#2#6\calx_thpf#3#5%
  \calx_thpb#3#6%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpg#2#6\calx_thpg#3#5%
  \calx_thpc#3#6%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpd#3#6%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpe#3#6%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpf#3#6%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpg#3#6\relax!%
}%
```

35.11. Generated helpers and products

Generated by tools/trig_genfix.py: helpers.

```
\def\calx_thda #1#2#3#4#5#6#7#8{-#1*#5}%
\def\calx_thdb #1#2#3#4#5#6#7#8{-#1*#6-#2*#5}%
\def\calx_thdc #1#2#3#4#5#6#7#8{-#1*#7-#2*#6-#3*#5}%
\def\calx_thdd #1#2#3#4#5#6#7#8{-#1*#8-#2*#7-#3*#6-#4*#5}%
\def\calx_thde #1#2#3#4#5#6#7#8{-#2*#8-#3*#7-#4*#6}%
\def\calx_thdf #1#2#3#4#5#6#7#8{-#3*#8-#4*#7}%
\def\calx_thdg #1#2#3#4#5#6#7#8{-#4*#8}%
\def\calx_tcsa #1#2#3#4{#1}%
\def\calx_tcsb #1#2#3#4{#2}%
\def\calx_tcsc #1#2#3#4{#3}%
\def\calx_tcsd #1#2#3#4{#4}%
\def\calx_tpoaa #1#2#3#4#5#6{\calx_tpg 1.}%
\def\calx_tpoab #1#2#3#4#5#6{\calx_tpg 2.}%
\def\calx_tpoac #1#2#3#4#5#6{\calx_tpg 3.}%
\def\calx_tpoad #1#2#3#4#5#6{\calx_tpg 4.}%
\def\calx_tpoae #1#2#3#4#5#6{\calx_tpg 5.}%
\def\calx_tpoaf #1#2#3#4#5#6{\calx_tpg 6.}%
\def\calx_tpoag #1#2#3#4#5#6{\calx_tpg 7.}%
\def\calx_tpoah #1#2#3#4#5#6{\calx_tpg 8.}%
\def\calx_tpoai #1#2#3#4#5#6{\calx_tpg 9.}%
\def\calx_tpoaj #1#2#3#4#5#6{\calx_tpg 10.}%
\def\calx_tpoak #1#2#3#4#5#6{\calx_tpg 11.}%
```

Generated: square of M for u.

```
\def\calx_thpa #1#2#3#4#5#6#7#8{+#1*#5}%
\def\calx_thpb #1#2#3#4#5#6#7#8{+#1*#6+#2*#5}%
\def\calx_thpc #1#2#3#4#5#6#7#8{+#1*#7+#2*#6+#3*#5}%
\def\calx_thpd #1#2#3#4#5#6#7#8{+#1*#8+#2*#7+#3*#6+#4*#5}%
\def\calx_thpe #1#2#3#4#5#6#7#8{+#2*#8+#3*#7+#4*#6}%
\def\calx_thpf #1#2#3#4#5#6#7#8{+#3*#8+#4*#7}%
\def\calx_thpg #1#2#3#4#5#6#7#8{+#4*#8}%
\def\calx_tpqa #1#2#3#4#5#6{\calx_tpg 8.}%
\def\calx_tpqb #1#2#3#4#5#6{\calx_tpg 16.}%
\def\calx_tpqc #1#2#3#4#5#6{\calx_tpg 24.}%
\expandafter\def\csname calx_txm1\endcsname1#1!%
  {{{0000}{0000}\calx_tbl#1}}%
\expandafter\def\csname calx_txm2\endcsname1#1!1#2!%
  {{\calx_tbl#2\calx_tbl#1}}%
\expandafter\def\csname calx_txm3\endcsname1#1!1#2!1#3!%
  {{{0000}{0000}\calx_tbl#3}{\calx_tbl#2\calx_tbl#1}}%
\expandafter\def\csname calx_txm4\endcsname1#1!1#2!1#3!1#4!%
  {{\calx_tbl#4\calx_tbl#3}{\calx_tbl#2\calx_tbl#1}}%
\expandafter\def\csname calx_txm5\endcsname1#1!1#2!1#3!1#4!1#5!%
  {{{0000}{0000}\calx_tbl#5}{\calx_tbl#4\calx_tbl#3}{\calx_tbl#2\calx_tbl#1}}%
\expandafter\def\csname calx_tsq1\endcsname#1%
{%
\expandafter\calx_tpqa\the\numexpr 1000000000%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpa#1#1%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpb#1#1%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpc#1#1%
```

```
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpd#1#1%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpe#1#1%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpf#1#1%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpg#1#1\relax!;%
}%
\expandafter\def\csname calx_tsq2\endcsname#1#2%
{%
\expandafter\calx_tpqb\the\numexpr 1000000000%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpa#1#1%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpb#1#1%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpc#1#1%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpd#1#1%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpe#1#1\calx_thpa#1#2%
  \calx_thpa#2#1%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpf#1#1\calx_thpb#1#2%
  \calx_thpb#2#1%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpg#1#1\calx_thpc#1#2%
  \calx_thpc#2#1%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpd#1#2\calx_thpd#2#1%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpe#1#2\calx_thpe#2#1%
  \calx_thpa#2#2%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpf#1#2\calx_thpf#2#1%
  \calx_thpb#2#2%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpg#1#2\calx_thpg#2#1%
  \calx_thpc#2#2%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpd#2#2%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpe#2#2%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpf#2#2%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpg#2#2\relax!;%
}%
\expandafter\def\csname calx_tsq3\endcsname#1#2#3%
{%
\expandafter\calx_tpqc\the\numexpr 1000000000%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpa#1#1%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpb#1#1%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpc#1#1%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpd#1#1%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpe#1#1\calx_thpa#1#2%
  \calx_thpa#2#1%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpf#1#1\calx_thpb#1#2%
  \calx_thpb#2#1%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpg#1#1\calx_thpc#1#2%
  \calx_thpc#2#1%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpd#1#2\calx_thpd#2#1%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpe#1#2\calx_thpa#1#3%
  \calx_thpe#2#1\calx_thpa#2#2\calx_thpa#3#1%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpf#1#2\calx_thpb#1#3%
  \calx_thpf#2#1\calx_thpb#2#2\calx_thpb#3#1%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpg#1#2\calx_thpc#1#3%
  \calx_thpg#2#1\calx_thpc#2#2\calx_thpc#3#1%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpd#1#3\calx_thpd#2#2%
  \calx_thpd#3#1%
```

```
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpe#1#3\calx_thpe#2#2%
  \calx_thpa#2#3\calx_thpe#3#1\calx_thpa#3#2%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpf#1#3\calx_thpf#2#2%
  \calx_thpb#2#3\calx_thpf#3#1\calx_thpb#3#2%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpg#1#3\calx_thpg#2#2%
  \calx_thpc#2#3\calx_thpg#3#1\calx_thpc#3#2%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpd#2#3\calx_thpd#3#2%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpe#2#3\calx_thpe#3#2%
  \calx_thpa#3#3%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpf#2#3\calx_thpf#3#2%
  \calx_thpb#3#3%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpg#2#3\calx_thpg#3#2%
  \calx_thpc#3#3%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpd#3#3%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpe#3#3%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpf#3#3%
\expandafter\calx_tpkq\the\numexpr 1000000000\calx_thpg#3#3\relax!;%
}%
```

Generated: products, shapes (s, n) of the tables.

```
\expandafter\def\csname calx_tm1_2\endcsname#1#2#3%
{%
\expandafter\calx_tpoab\the\numexpr 2100000000+\calx_tcsa#2%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsb#2%
\expandafter\calx_tpk\the\numexpr 2100000000\calx_thda#3#1%
\expandafter\calx_tpk\the\numexpr 2100000000\calx_thdb#3#1\relax!;\calx_tgobW%
}%
\expandafter\def\csname calx_tm2_2\endcsname#1#2#3%
{%
\expandafter\calx_tpoab\the\numexpr 2100000000+\calx_tcsa#2%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsb#2\calx_thda#3#1%
\expandafter\calx_tpk\the\numexpr 2100000000\calx_thdb#3#1%
\expandafter\calx_tpk\the\numexpr 2100000000\calx_thdc#3#1\relax!;\calx_tgobW%
}%
\expandafter\def\csname calx_tm2_3\endcsname#1#2#3%
{%
\expandafter\calx_tpoac\the\numexpr 2100000000+\calx_tcsa#2%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsb#2%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsc#2\calx_thda#3#1%
\expandafter\calx_tpk\the\numexpr 2100000000\calx_thdb#3#1%
\expandafter\calx_tpk\the\numexpr 2100000000\calx_thdc#3#1\relax!;\calx_tgobW%
}%
\expandafter\def\csname calx_tm3_3\endcsname#1#2#3%
{%
\expandafter\calx_tpoac\the\numexpr 2100000000+\calx_tcsa#2%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsb#2\calx_thda#3#1%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsc#2\calx_thdb#3#1%
\expandafter\calx_tpk\the\numexpr 2100000000\calx_thdc#3#1%
\expandafter\calx_tpk\the\numexpr 2100000000\calx_thdd#3#1\relax!;\calx_tgobW%
}%
\expandafter\def\csname calx_tm3_4\endcsname#1#2#3%
{%
\expandafter\calx_tpoad\the\numexpr 2100000000+\calx_tcsa#2%
```

```
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsb#2%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsc#2\calx_thda#3#1%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsd#2\calx_thdb#3#1%
\expandafter\calx_tpk\the\numexpr 2100000000\calx_thdc#3#1%
\expandafter\calx_tpk\the\numexpr 2100000000\calx_thdd#3#1\relax!;\calx_tgobW%
}%
\expandafter\def\csname calx_tm4_4\endcsname#1#2#3%
{%
\expandafter\calx_tpoad\the\numexpr 2100000000+\calx_tcsa#2%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsb#2\calx_thda#3#1%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsc#2\calx_thdb#3#1%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsd#2\calx_thdc#3#1%
\expandafter\calx_tpk\the\numexpr 2100000000\calx_thdd#3#1%
\expandafter\calx_tpk\the\numexpr 2100000000\calx_thde#3#1\relax!;\calx_tgobW%
}%
\expandafter\def\csname calx_tm4_5\endcsname#1#2#3#4%
{%
\expandafter\calx_tpoae\the\numexpr 2100000000+\calx_tcsa#2%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsb#2%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsc#2\calx_thda#4#1%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsd#2\calx_thdb#4#1%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsa#3\calx_thdc#4#1%
\expandafter\calx_tpk\the\numexpr 2100000000\calx_thdd#4#1%
\expandafter\calx_tpk\the\numexpr 2100000000\calx_thde#4#1\relax!;\calx_tgobW%
}%
\expandafter\def\csname calx_tm5_5\endcsname#1#2#3#4#5#6%
{%
\expandafter\calx_tpoae\the\numexpr 2100000000+\calx_tcsa#3%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsb#3\calx_thda#5#1%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsc#3\calx_thdb#5#1%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsd#3\calx_thdc#5#1%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsa#4\calx_thdd#5#1%
\expandafter\calx_tpk\the\numexpr 2100000000\calx_thde#5#1\calx_thda#5#2%
  \calx_thda#6#1%
\expandafter\calx_tpk\the\numexpr 2100000000\calx_thdf#5#1\calx_thdb#5#2%
  \calx_thdb#6#1\relax!;\calx_tgobW%
}%
\expandafter\def\csname calx_tm5_6\endcsname#1#2#3#4#5#6%
{%
\expandafter\calx_tpoaf\the\numexpr 2100000000+\calx_tcsa#3%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsb#3%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsc#3\calx_thda#5#1%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsd#3\calx_thdb#5#1%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsa#4\calx_thdc#5#1%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsb#4\calx_thdd#5#1%
\expandafter\calx_tpk\the\numexpr 2100000000\calx_thde#5#1\calx_thda#5#2%
  \calx_thda#6#1%
\expandafter\calx_tpk\the\numexpr 2100000000\calx_thdf#5#1\calx_thdb#5#2%
  \calx_thdb#6#1\relax!;\calx_tgobW%
}%
\expandafter\def\csname calx_tm6_6\endcsname#1#2#3#4#5#6%
{%
```

```
\expandafter\calx_tpoaf\the\numexpr 2100000000+\calx_tcsa#3%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsb#3\calx_thda#5#1%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsc#3\calx_thdb#5#1%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsd#3\calx_thdc#5#1%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsa#4\calx_thdd#5#1%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsb#4\calx_thde#5#1%
  \calx_thda#5#2\calx_thda#6#1%
\expandafter\calx_tpk\the\numexpr 2100000000\calx_thdf#5#1\calx_thdb#5#2%
  \calx_thdb#6#1%
\expandafter\calx_tpk\the\numexpr 2100000000\calx_thdg#5#1\calx_thdc#5#2%
  \calx_thdc#6#1\relax!;\calx_tgobW%
}%
\expandafter\def\csname calx_tm6_7\endcsname#1#2#3#4#5#6%
{%
\expandafter\calx_tpoag\the\numexpr 2100000000+\calx_tcsa#3%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsb#3%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsc#3\calx_thda#5#1%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsd#3\calx_thdb#5#1%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsa#4\calx_thdc#5#1%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsb#4\calx_thdd#5#1%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsc#4\calx_thde#5#1%
  \calx_thda#5#2\calx_thda#6#1%
\expandafter\calx_tpk\the\numexpr 2100000000\calx_thdf#5#1\calx_thdb#5#2%
  \calx_thdb#6#1%
\expandafter\calx_tpk\the\numexpr 2100000000\calx_thdg#5#1\calx_thdc#5#2%
  \calx_thdc#6#1\relax!;\calx_tgobW%
}%
\expandafter\def\csname calx_tm7_7\endcsname#1#2#3#4#5#6%
{%
\expandafter\calx_tpoag\the\numexpr 2100000000+\calx_tcsa#3%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsb#3\calx_thda#5#1%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsc#3\calx_thdb#5#1%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsd#3\calx_thdc#5#1%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsa#4\calx_thdd#5#1%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsb#4\calx_thde#5#1%
  \calx_thda#5#2\calx_thda#6#1%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsc#4\calx_thdf#5#1%
  \calx_thdb#5#2\calx_thdb#6#1%
\expandafter\calx_tpk\the\numexpr 2100000000\calx_thdg#5#1\calx_thdc#5#2%
  \calx_thdc#6#1%
\expandafter\calx_tpk\the\numexpr 2100000000\calx_thdd#5#2\calx_thdd#6#1%
  \relax!;\calx_tgobW%
}%
\expandafter\def\csname calx_tm7_8\endcsname#1#2#3#4#5#6%
{%
\expandafter\calx_tpoah\the\numexpr 2100000000+\calx_tcsa#3%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsb#3%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsc#3\calx_thda#5#1%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsd#3\calx_thdb#5#1%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsa#4\calx_thdc#5#1%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsb#4\calx_thdd#5#1%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsc#4\calx_thde#5#1%
```

```
  \calx_thda#5#2\calx_thda#6#1%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsd#4\calx_thdf#5#1%
  \calx_thdb#5#2\calx_thdb#6#1%
\expandafter\calx_tpk\the\numexpr 2100000000\calx_thdg#5#1\calx_thdc#5#2%
  \calx_thdc#6#1%
\expandafter\calx_tpk\the\numexpr 2100000000\calx_thdd#5#2\calx_thdd#6#1%
  \relax!;\calx_tgobW%
}%
\expandafter\def\csname calx_tm8_8\endcsname#1#2#3#4#5#6%
{%
\expandafter\calx_tpoah\the\numexpr 2100000000+\calx_tcsa#3%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsb#3\calx_thda#5#1%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsc#3\calx_thdb#5#1%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsd#3\calx_thdc#5#1%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsa#4\calx_thdd#5#1%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsb#4\calx_thde#5#1%
  \calx_thda#5#2\calx_thda#6#1%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsc#4\calx_thdf#5#1%
  \calx_thdb#5#2\calx_thdb#6#1%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsd#4\calx_thdg#5#1%
  \calx_thdc#5#2\calx_thdc#6#1%
\expandafter\calx_tpk\the\numexpr 2100000000\calx_thdd#5#2\calx_thdd#6#1%
\expandafter\calx_tpk\the\numexpr 2100000000\calx_thde#5#2\calx_thde#6#1%
  \calx_thda#6#2\relax!;\calx_tgobW%
}%
\expandafter\def\csname calx_tm8_9\endcsname#1#2#3#4#5#6#7%
{%
\expandafter\calx_tpoai\the\numexpr 2100000000+\calx_tcsa#3%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsb#3%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsc#3\calx_thda#6#1%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsd#3\calx_thdb#6#1%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsa#4\calx_thdc#6#1%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsb#4\calx_thdd#6#1%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsc#4\calx_thde#6#1%
  \calx_thda#6#2\calx_thda#7#1%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsd#4\calx_thdf#6#1%
  \calx_thdb#6#2\calx_thdb#7#1%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsa#5\calx_thdg#6#1%
  \calx_thdc#6#2\calx_thdc#7#1%
\expandafter\calx_tpk\the\numexpr 2100000000\calx_thdd#6#2\calx_thdd#7#1%
\expandafter\calx_tpk\the\numexpr 2100000000\calx_thde#6#2\calx_thde#7#1%
  \calx_thda#7#2\relax!;\calx_tgobW%
}%
\expandafter\def\csname calx_tm9_9\endcsname#1#2#3#4#5#6#7#8#9%
{%
\expandafter\calx_tpoai\the\numexpr 2100000000+\calx_tcsa#4%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsb#4\calx_thda#7#1%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsc#4\calx_thdb#7#1%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsd#4\calx_thdc#7#1%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsa#5\calx_thdd#7#1%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsb#5\calx_thde#7#1%
  \calx_thda#7#2\calx_thda#8#1%
```

```
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsc#5\calx_thdf#7#1%
  \calx_thdb#7#2\calx_thdb#8#1%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsd#5\calx_thdg#7#1%
  \calx_thdc#7#2\calx_thdc#8#1%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsa#6\calx_thdd#7#2%
  \calx_thdd#8#1%
\expandafter\calx_tpk\the\numexpr 2100000000\calx_thde#7#2\calx_thda#7#3%
  \calx_thde#8#1\calx_thda#8#2\calx_thda#9#1%
\expandafter\calx_tpk\the\numexpr 2100000000\calx_thdf#7#2\calx_thdb#7#3%
  \calx_thdf#8#1\calx_thdb#8#2\calx_thdb#9#1\relax!;\calx_tgobW%
}%
\expandafter\def\csname calx_tm9_10\endcsname#1#2#3#4#5#6#7#8#9%
{%
\expandafter\calx_tpoaj\the\numexpr 2100000000+\calx_tcsa#4%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsb#4%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsc#4\calx_thda#7#1%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsd#4\calx_thdb#7#1%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsa#5\calx_thdc#7#1%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsb#5\calx_thdd#7#1%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsc#5\calx_thde#7#1%
  \calx_thda#7#2\calx_thda#8#1%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsd#5\calx_thdf#7#1%
  \calx_thdb#7#2\calx_thdb#8#1%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsa#6\calx_thdg#7#1%
  \calx_thdc#7#2\calx_thdc#8#1%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsb#6\calx_thdd#7#2%
  \calx_thdd#8#1%
\expandafter\calx_tpk\the\numexpr 2100000000\calx_thde#7#2\calx_thda#7#3%
  \calx_thde#8#1\calx_thda#8#2\calx_thda#9#1%
\expandafter\calx_tpk\the\numexpr 2100000000\calx_thdf#7#2\calx_thdb#7#3%
  \calx_thdf#8#1\calx_thdb#8#2\calx_thdb#9#1\relax!;\calx_tgobW%
}%
\expandafter\def\csname calx_tm10_10\endcsname#1#2#3#4#5#6#7#8#9%
{%
\expandafter\calx_tpoaj\the\numexpr 2100000000+\calx_tcsa#4%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsb#4\calx_thda#7#1%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsc#4\calx_thdb#7#1%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsd#4\calx_thdc#7#1%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsa#5\calx_thdd#7#1%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsb#5\calx_thde#7#1%
  \calx_thda#7#2\calx_thda#8#1%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsc#5\calx_thdf#7#1%
  \calx_thdb#7#2\calx_thdb#8#1%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsd#5\calx_thdg#7#1%
  \calx_thdc#7#2\calx_thdc#8#1%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsa#6\calx_thdd#7#2%
  \calx_thdd#8#1%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsb#6\calx_thde#7#2%
  \calx_thda#7#3\calx_thde#8#1\calx_thda#8#2\calx_thda#9#1%
\expandafter\calx_tpk\the\numexpr 2100000000\calx_thdf#7#2\calx_thdb#7#3%
  \calx_thdf#8#1\calx_thdb#8#2\calx_thdb#9#1%
\expandafter\calx_tpk\the\numexpr 2100000000\calx_thdg#7#2\calx_thdc#7#3%
```

```
  \calx_thdg#8#1\calx_thdc#8#2\calx_thdc#9#1\relax!;\calx_tgobW%
}%
\expandafter\def\csname calx_tm10_11\endcsname#1#2#3#4#5#6#7#8#9%
{%
\expandafter\calx_tpoak\the\numexpr 2100000000+\calx_tcsa#4%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsb#4%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsc#4\calx_thda#7#1%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsd#4\calx_thdb#7#1%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsa#5\calx_thdc#7#1%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsb#5\calx_thdd#7#1%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsc#5\calx_thde#7#1%
  \calx_thda#7#2\calx_thda#8#1%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsd#5\calx_thdf#7#1%
  \calx_thdb#7#2\calx_thdb#8#1%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsa#6\calx_thdg#7#1%
  \calx_thdc#7#2\calx_thdc#8#1%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsb#6\calx_thdd#7#2%
  \calx_thdd#8#1%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsc#6\calx_thde#7#2%
  \calx_thda#7#3\calx_thde#8#1\calx_thda#8#2\calx_thda#9#1%
\expandafter\calx_tpk\the\numexpr 2100000000\calx_thdf#7#2\calx_thdb#7#3%
  \calx_thdf#8#1\calx_thdb#8#2\calx_thdb#9#1%
\expandafter\calx_tpk\the\numexpr 2100000000\calx_thdg#7#2\calx_thdc#7#3%
  \calx_thdg#8#1\calx_thdc#8#2\calx_thdc#9#1\relax!;\calx_tgobW%
}%
\expandafter\def\csname calx_tm11_11\endcsname#1#2#3#4#5#6#7#8#9%
{%
\expandafter\calx_tpoak\the\numexpr 2100000000+\calx_tcsa#4%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsb#4\calx_thda#7#1%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsc#4\calx_thdb#7#1%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsd#4\calx_thdc#7#1%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsa#5\calx_thdd#7#1%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsb#5\calx_thde#7#1%
  \calx_thda#7#2\calx_thda#8#1%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsc#5\calx_thdf#7#1%
  \calx_thdb#7#2\calx_thdb#8#1%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsd#5\calx_thdg#7#1%
  \calx_thdc#7#2\calx_thdc#8#1%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsa#6\calx_thdd#7#2%
  \calx_thdd#8#1%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsb#6\calx_thde#7#2%
  \calx_thda#7#3\calx_thde#8#1\calx_thda#8#2\calx_thda#9#1%
\expandafter\calx_tpk\the\numexpr 2100000000+\calx_tcsc#6\calx_thdf#7#2%
  \calx_thdb#7#3\calx_thdf#8#1\calx_thdb#8#2\calx_thdb#9#1%
\expandafter\calx_tpk\the\numexpr 2100000000\calx_thdg#7#2\calx_thdc#7#3%
  \calx_thdg#8#1\calx_thdc#8#2\calx_thdc#9#1%
\expandafter\calx_tpk\the\numexpr 2100000000\calx_thdd#7#3\calx_thdd#8#2%
  \calx_thdd#9#1\relax!;\calx_tgobW%
}%
```

35.12. Schedules of the fixed path

Generated: schedules (P with A <= 11).

35.13. Argument reduction

Reduction B1, for 0.8 <= |X| < 10^4 (E = D + 8e <= 4; E > 4: \calx_trslow, below), m = ceil((P+6)/8) (m = 2 if P <= 8, \calx_ttrigm), W = 8m >= P + 6 and H = round(pi/2 10^W):

k = round(2|X|/pi), R = |X| 10^W - k H, r = R 10^-W.

k: Xh = the top 9 digits of M (rounded), Xh = |X| 10^(9-E) (1 + d), |d| < 10^-8; k = round(round(Xh c / 10^9) / 10^(9-E)), c = 636619772 (2/pi 10^9 rounded, relative error < 8 10^-10): the error on 2|X|/pi is < 0.0017, so |r| < pi/4 + 0.0017 pi/2 < 0.79 < 0.8 (domain of the series kernels). E = 0: k = 1. k <= 6367 (one limb; the integer part of |X| is in the low limb of the units group).

R: 4-digit limbs, columns j = 2m..0 of x_j - k h_j + carry in one nested \numexpr chain (\calx_trd<m>, generated by tools/trig_genred.py, h_j literal; m >= 8: in two macros, \calx_trdb<m> taking the inner part of the chain built by \calx_trd<m>), carry by offset 1200000000. |X| 10^W is an integer (-8e <= D <= P + 7 < W + 8, hence z = e + m >= 0): its blocks are z zero blocks then the blocks of M (and a zero units block if E = 0), m + 1 groups of 8 digits. |r| < 1: the units limb is 0000 (R >= 0) or 9999 (R < 0), the higher limbs are not computed (exact modulo 10^(4(2m+1))).

Output (\calx_trh): q = k mod 4 and r as a native float, sign of R, e = -m, leading zero blocks stripped, D exact (\calx_tnd); for R < 0 the blocks are 99999999 - b (= |R| - 1). R = 0 gives r = 10^-W. Error on r (radians): < (k/2 + 1) 10^-W <= 3185 10^-(P+6) < 0.0032 10^-P.

```
\def\calx_tred #1.#2.%
{%
    \ifnum\numexpr#1+8*#2=\xint_c_ \xint_dothis{\calx_trkc 1.}\fi
    \xint_orthat{\expandafter\calx_trka\the\numexpr(2*#1+7)/16\expandafter.%
      \the\numexpr#1+8-8*((2*#1+7)/16).}%
    #1.#2.%
}%
\def\calx_trka #1.#2.#3.#4.#5\xint:%
{%
    \expandafter\calx_trkc\the\numexpr(\calx_trta#1.#2.#5\xint:)%
      *636619772/1000000000/\CALX_pten{9-#3-8*#4}.#3.#4.#5\xint:%
}%
```

\calx_trta <n>.<d>.<blocks>\xint: (inside \numexpr): Xh from the top block T (d digits) and the next one B: T 10^(9-d) + round(B/10^(d-1)).

```
\def\calx_trta #1.%
  {\ifnum#1>\xint_c_ii\expandafter\calx_trtb\else\expandafter\calx_trtc\fi#1.}%
\def\calx_trtb #1.#2.1#3!{\expandafter\calx_trta\the\numexpr#1-1.#2.}%
\def\calx_trtc #1.{\csname calx_trt#1\endcsname}%
\expandafter\def\csname calx_trt1\endcsname #1.1#2!#3\xint:%
  {#2*\CALX_pten{9-#1}}%
\expandafter\def\csname calx_trt2\endcsname #1.1#2!1#3!#4\xint:%
  {#3*\CALX_pten{9-#1}+#2/\CALX_pten{#1-1}}%
```

\calx_trkc <k>.<D>.<e>.<blocks>\xint: calls \calx_trd<m> with {k}, the m + 1 groups (\calx_tzg<z>, \calx_tbg), then ;<m>.<q>.

```
%
\def\calx_trkc #1.#2.#3.#4\xint:%
{%
    \csname calx_trd\the\numexpr\calx_ttrigm\expandafter\endcsname
    \expanded{{#1}%
      \csname calx_tzg\the\numexpr#3+\calx_ttrigm\endcsname
```

```
      \calx_tbg#4\xint:\ifnum\numexpr#2+8*#3=\xint_c_{00000000}\fi
      ;\the\numexpr\calx_ttrigm.\the\numexpr#1-4*((2*#1-3)/8).}%
}%
\def\calx_tbg #1{\xint_gob_til_xint: #1\calx_tbge\xint:\calx_tbh#1}%
\def\calx_tbge\xint:\calx_tbh\xint:{}%
\def\calx_tbh 1#1!{{#1}\calx_tbg}%
\expandafter\def\csname calx_tzg0\endcsname{}%
\expandafter\def\csname calx_tzg1\endcsname{{00000000}}%
\expandafter\def\csname calx_tzg2\endcsname{{00000000}{00000000}}%
\expandafter\def\csname calx_tzg3\endcsname
  {{00000000}{00000000}{00000000}}%
\expandafter\def\csname calx_tzg4\endcsname
  {{00000000}{00000000}{00000000}{00000000}}%
\expandafter\def\csname calx_tzg5\endcsname
  {{00000000}{00000000}{00000000}{00000000}{00000000}}%
\expandafter\def\csname calx_tzg6\endcsname
  {{00000000}{00000000}{00000000}{00000000}{00000000}{00000000}}%
\expandafter\edef\csname calx_tzg7\endcsname{\csname calx_tzg6\endcsname
  {00000000}}%
\expandafter\edef\csname calx_tzg8\endcsname{\csname calx_tzg7\endcsname
  {00000000}}%
\expandafter\edef\csname calx_tzg9\endcsname{\csname calx_tzg8\endcsname
  {00000000}}%
\expandafter\edef\csname calx_tzg10\endcsname{\csname calx_tzg9\endcsname
  {00000000}}%
\expandafter\edef\csname calx_tzg11\endcsname{\csname calx_tzg10\endcsname
  {00000000}}%
\expandafter\edef\csname calx_tzg12\endcsname{\csname calx_tzg11\endcsname
  {00000000}}%
\expandafter\edef\csname calx_tzg13\endcsname{\csname calx_tzg12\endcsname
  {00000000}}%
\expandafter\edef\csname calx_tzg14\endcsname{\csname calx_tzg13\endcsname
  {00000000}}%
\def\calx_tlo #1#2#3#4{}%
\def\calx_thi #1#2#3#4#5#6#7#8{#1#2#3#4}%
\def\calx_trp #1#2#3#4#5#6{+#1#2#3#4#5#6-120000!}%
```

\calx_trh: the chain output <top column>!<l_(2m-1)>!...!<l_0>!;<m>.<q>. (limbs MS first); the low limb of the top column gives the sign. \calx_trsa <count>.<m>.<conv><sign> strips the leading zero blocks (pairs of limbs), \calx_trv reverses the others (LS first) inside \expanded. <conv>: \calx_tbp (1<l><l>) or \calx_tbn (complement).

```
\def\calx_trh #1#2#3#4#5#6#7!#8;#9.%
{%
    \expandafter\calx_trz\expanded{\csname calx_trs\ifnum#7=\xint_c_ p\else
      n\fi\endcsname#9.#9.#8}\xint:%
}%
\def\calx_trz #1\xint:#2.{ #2.#1}%
\def\calx_trsp #1.#2.{\calx_trsa#1.#2.\calx_tbp+}%
\def\calx_trsn #1.#2.{\calx_trsa#1.#2.\calx_tbn-}%
\def\calx_tbp #1#2{1#1#2}%
\def\calx_tbn #1#2{\the\numexpr199999999-#1#2\relax}%
\def\calx_trsa #1.#2.#3#4%
{%
```

```
    \ifnum#1=\xint_c_ \xint_dothis{F+1.-#2.100000001!}\fi
    \xint_orthat{\calx_trsb#1.#2.#3#4}%
}%
\def\calx_trsb #1.#2.#3#4#5!#6!%
{%
    \ifnum#3{#5}{#6}=100000000
      \expandafter\xint_firstoftwo\else\expandafter\xint_secondoftwo\fi
    {\expandafter\calx_trsa\the\numexpr#1-1.#2.#3#4}%
    {F#4\the\numexpr8*#1-8+\expandafter\calx_tnd\the\numexpr#3{#5}{#6}.%
     .-#2.\expandafter\calx_trv\the\numexpr#1-1.{#3{#5}{#6}!}#3}%
}%
\def\calx_trv #1.{\ifnum#1=\xint_c_ \calx_trve\fi\calx_trvb#1.}%
\def\calx_trve\fi\calx_trvb#1.#2#3{\fi#2}%
\def\calx_trvb #1.#2#3#4!#5!%
  {\expandafter\calx_trv\the\numexpr#1-1.{#3{#4}{#5}!#2}#3}%
\def\calx_tnd #1.%
{%
    \ifnum#1<100010000
      \ifnum#1<100000100 \ifnum#1<100000010 1\else2\fi
      \else\ifnum#1<100001000 3\else4\fi\fi
    \else
      \ifnum#1<101000000 \ifnum#1<100100000 5\else6\fi
      \else\ifnum#1<110000000 7\else8\fi\fi
    \fi
}%
```

Generated by tools/trig_genred.py: \calx_trd<m>, m = 2..14 (and \calx_trdb<m>, m >= 8).

```
\expandafter\def\csname calx_trd2\endcsname#1#2#3#4%
{%
\expandafter\calx_trh\the\numexpr 1200000000+\calx_tlo#4-#1*1%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#3-#1*5707%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#3-#1*9632%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#2-#1*6794%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#2-#1*8966\relax!%
}%
\expandafter\def\csname calx_trd3\endcsname#1#2#3#4#5%
{%
\expandafter\calx_trh\the\numexpr 1200000000+\calx_tlo#5-#1*1%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#4-#1*5707%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#4-#1*9632%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#3-#1*6794%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#3-#1*8966%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#2-#1*1923%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#2-#1*1322\relax!%
}%
\expandafter\def\csname calx_trd4\endcsname#1#2#3#4#5#6%
{%
\expandafter\calx_trh\the\numexpr 1200000000+\calx_tlo#6-#1*1%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#5-#1*5707%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#5-#1*9632%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#4-#1*6794%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#4-#1*8966%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#3-#1*1923%
```

```
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#3-#1*1321%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#2-#1*6916%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#2-#1*3975\relax!%
}%
\expandafter\def\csname calx_trd5\endcsname#1#2#3#4#5#6#7%
{%
\expandafter\calx_trh\the\numexpr 1200000000+\calx_tlo#7-#1*1%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#6-#1*5707%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#6-#1*9632%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#5-#1*6794%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#5-#1*8966%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#4-#1*1923%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#4-#1*1321%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#3-#1*6916%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#3-#1*3975%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#2-#1*1442%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#2-#1*986\relax!%
}%
\expandafter\def\csname calx_trd6\endcsname#1#2#3#4#5#6#7#8%
{%
\expandafter\calx_trh\the\numexpr 1200000000+\calx_tlo#8-#1*1%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#7-#1*5707%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#7-#1*9632%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#6-#1*6794%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#6-#1*8966%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#5-#1*1923%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#5-#1*1321%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#4-#1*6916%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#4-#1*3975%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#3-#1*1442%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#3-#1*985%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#2-#1*8469%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#2-#1*9688\relax!%
}%
\expandafter\def\csname calx_trd7\endcsname#1#2#3#4#5#6#7#8#9%
{%
\expandafter\calx_trh\the\numexpr 1200000000+\calx_tlo#9-#1*1%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#8-#1*5707%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#8-#1*9632%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#7-#1*6794%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#7-#1*8966%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#6-#1*1923%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#6-#1*1321%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#5-#1*6916%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#5-#1*3975%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#4-#1*1442%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#4-#1*985%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#3-#1*8469%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#3-#1*9687%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#2-#1*5529%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#2-#1*1049\relax!%
}%
```

```
\expandafter\def\csname calx_trd8\endcsname#1#2#3#4#5#6#7#8#9%
{%
\csname calx_trdb8\endcsname{#1}{%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#9-#1*5707%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#9-#1*9632%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#8-#1*6794%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#8-#1*8966%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#7-#1*1923%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#7-#1*1321%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#6-#1*6916%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#6-#1*3975%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#5-#1*1442%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#5-#1*985%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#4-#1*8469%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#4-#1*9687%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#3-#1*5529%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#3-#1*1048%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#2-#1*7472%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#2-#1*2962\relax!%
}%
}%
\expandafter\def\csname calx_trdb8\endcsname#1#2#3%
{%
\expandafter\calx_trh\the\numexpr 1200000000+\calx_tlo#3-#1*1%
#2}%
\expandafter\def\csname calx_trd9\endcsname#1#2#3#4#5#6#7#8#9%
{%
\csname calx_trdb9\endcsname{#1}{%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#9-#1*6794%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#9-#1*8966%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#8-#1*1923%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#8-#1*1321%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#7-#1*6916%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#7-#1*3975%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#6-#1*1442%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#6-#1*985%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#5-#1*8469%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#5-#1*9687%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#4-#1*5529%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#4-#1*1048%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#3-#1*7472%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#3-#1*2961%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#2-#1*5390%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#2-#1*8203\relax!%
}%
}%
\expandafter\def\csname calx_trdb9\endcsname#1#2#3#4%
{%
\expandafter\calx_trh\the\numexpr 1200000000+\calx_tlo#4-#1*1%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#3-#1*5707%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#3-#1*9632%
#2}%
```

```
\expandafter\def\csname calx_trd10\endcsname#1#2#3#4#5#6#7#8#9%
{%
\csname calx_trdb10\endcsname{#1}{%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#9-#1*1923%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#9-#1*1321%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#8-#1*6916%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#8-#1*3975%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#7-#1*1442%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#7-#1*985%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#6-#1*8469%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#6-#1*9687%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#5-#1*5529%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#5-#1*1048%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#4-#1*7472%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#4-#1*2961%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#3-#1*5390%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#3-#1*8203%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#2-#1*1431%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#2-#1*450\relax!%
}%
}%
\expandafter\def\csname calx_trdb10\endcsname#1#2#3#4#5%
{%
\expandafter\calx_trh\the\numexpr 1200000000+\calx_tlo#5-#1*1%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#4-#1*5707%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#4-#1*9632%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#3-#1*6794%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#3-#1*8966%
#2}%
\expandafter\def\csname calx_trd11\endcsname#1#2#3#4#5#6#7#8#9%
{%
\csname calx_trdb11\endcsname{#1}{%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#9-#1*6916%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#9-#1*3975%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#8-#1*1442%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#8-#1*985%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#7-#1*8469%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#7-#1*9687%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#6-#1*5529%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#6-#1*1048%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#5-#1*7472%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#5-#1*2961%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#4-#1*5390%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#4-#1*8203%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#3-#1*1431%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#3-#1*449%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#2-#1*9314%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#2-#1*174\relax!%
}%
}%
\expandafter\def\csname calx_trdb11\endcsname#1#2#3#4#5#6%
{%
```

```
\expandafter\calx_trh\the\numexpr 1200000000+\calx_tlo#6-#1*1%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#5-#1*5707%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#5-#1*9632%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#4-#1*6794%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#4-#1*8966%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#3-#1*1923%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#3-#1*1321%
#2}%
\expandafter\def\csname calx_trd12\endcsname#1#2#3#4#5#6#7#8#9%
{%
\csname calx_trdb12\endcsname{#1}{%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#9-#1*1442%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#9-#1*985%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#8-#1*8469%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#8-#1*9687%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#7-#1*5529%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#7-#1*1048%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#6-#1*7472%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#6-#1*2961%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#5-#1*5390%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#5-#1*8203%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#4-#1*1431%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#4-#1*449%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#3-#1*9314%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#3-#1*174%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#2-#1*1267%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#2-#1*1059\relax!%
}%
}%
\expandafter\def\csname calx_trdb12\endcsname#1#2#3#4#5#6#7%
{%
\expandafter\calx_trh\the\numexpr 1200000000+\calx_tlo#7-#1*1%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#6-#1*5707%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#6-#1*9632%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#5-#1*6794%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#5-#1*8966%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#4-#1*1923%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#4-#1*1321%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#3-#1*6916%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#3-#1*3975%
#2}%
\expandafter\def\csname calx_trd13\endcsname#1#2#3#4#5#6#7#8#9%
{%
\csname calx_trdb13\endcsname{#1}{%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#9-#1*8469%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#9-#1*9687%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#8-#1*5529%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#8-#1*1048%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#7-#1*7472%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#7-#1*2961%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#6-#1*5390%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#6-#1*8203%
```

```
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#5-#1*1431%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#5-#1*449%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#4-#1*9314%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#4-#1*174%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#3-#1*1267%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#3-#1*1058%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#2-#1*5339%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#2-#1*9107\relax!%
}%
}%
\expandafter\def\csname calx_trdb13\endcsname#1#2#3#4#5#6#7#8%
{%
\expandafter\calx_trh\the\numexpr 1200000000+\calx_tlo#8-#1*1%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#7-#1*5707%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#7-#1*9632%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#6-#1*6794%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#6-#1*8966%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#5-#1*1923%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#5-#1*1321%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#4-#1*6916%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#4-#1*3975%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#3-#1*1442%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#3-#1*985%
#2}%
\expandafter\def\csname calx_trd14\endcsname#1#2#3#4#5#6#7#8#9%
{%
\csname calx_trdb14\endcsname{#1}{%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#9-#1*5529%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#9-#1*1048%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#8-#1*7472%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#8-#1*2961%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#7-#1*5390%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#7-#1*8203%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#6-#1*1431%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#6-#1*449%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#5-#1*9314%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#5-#1*174%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#4-#1*1267%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#4-#1*1058%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#3-#1*5339%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#3-#1*9107%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#2-#1*4043%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#2-#1*2566\relax!%
}%
}%
\expandafter\def\csname calx_trdb14\endcsname#1#2#3#4#5#6#7#8#9%
{%
\expandafter\calx_trh\the\numexpr 1200000000+\calx_tlo#9-#1*1%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#8-#1*5707%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#8-#1*9632%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#7-#1*6794%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#7-#1*8966%
```

```
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#6-#1*1923%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#6-#1*1321%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#5-#1*6916%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#5-#1*3975%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#4-#1*1442%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#4-#1*985%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_thi#3-#1*8469%
\expandafter\calx_trp\the\numexpr 1200000000+\calx_tlo#3-#1*9687%
#2}%
```

Reduction for 10^4 <= |X| < 10^E, E = D + 8e <= min(P, 126 - P), from the callers (\calx_tsinr, \calx_tcos_r, \calx_tsc_rr) when E > 4: jumpers \calx_trj (sin, cos) and \calx_trjs (tan, cot, sec, csc, sinc), which replace \calx_tred by \calx_trslow, or, if E > min(P, 126 - P), drop the continuation (\calx_trjx, \calx_trjxs: also the finisher and its three arguments) and signal ControversialOperation (result I?).

Integers (M: the integer of the blocks of |X|, |X| = M 10^(8e)):

W = 8 ceil((P+E+2)/8) <= 128, G = E + 3,

T = round(2/pi 10^G), k = round(M T 10^(8e-G)),

H = round(pi/2 10^W), R = M 10^(W+8e) - k H, r = R 10^-W,

T and H by rounded shifts (\CALX_nshrr) of \calx_trT = round(2/pi 10^130) and \calx_trH = round(pi/2 10^129). The error on 2|X|/pi is < 0.006 (|r| < 0.8); the error on r is < (k/2 + 1) 10^-W <= 0.005 10^-P. q = k mod 4 from the low block of k; r rounded to P+4 digits.

```
\edef\calx_trT{\romannumeral0\CALX_nfromraw
  {6366197723675813430755350534900574481378385829618257949906693762355871 9%
  0536906140360455211065012343824291370907031832147571647384 5}}%
\edef\calx_trH{\romannumeral0\CALX_nfromraw
  {1570796326794896619231321691639751442098584699687552910487472296153908 2%
  0314310449931401741267105853399107404325664115332354692230 5}}%
\def\calx_trj\fi#1\calx_tred #2.#3.%
{%
    \fi
    \ifnum\numexpr#2+8*#3>\calx_trlimit \calx_trjx\fi
    #1\calx_trslow #2.#3.%
}%
\def\calx_trjs\fi#1\calx_tred #2.#3.%
{%
    \fi
    \ifnum\numexpr#2+8*#3>\calx_trlimit \calx_trjxs\fi
    #1\calx_trslow #2.#3.%
}%
\def\calx_trjx\fi#1\xint:\xint:%
  {\fi\CALX_nsig{ControversialOperation}{trig: argument too large}{}}%
\def\calx_trjxs\fi#1\xint:\xint:#2#3#4%
  {\fi\CALX_nsig{ControversialOperation}{trig: argument too large}{}}%
```

\calx_trslow, P <= 8 (\calx_trse; then 5 <= E <= P, |x| < 10^8): A exact (9 digits), F = 10^(9-E), a = floor(A/F) = 10^4 a1 + a0, b = A - a F;

k = 6366 a1 + round((a1 1977236758/10^5 + a0 636619772/10^4

+ b 636619772/10^(13-E)) / 10^5)

(2/pi to 10 digits, error on 2|x|/pi < 0.001); B1 with m = 3 (W = 24) and k = 10^4 k1 + k0 (\calx_trwk, columns x_j - k0 h_j - k1 h_(j-1), H = round(pi/2 10^24); x_5, x_6 from G = (A 10^(E-5)) mod 10^8, the lower limbs of |x| 10^24 being 0); output by \calx_trh as \calx_tred. Error on r < (k/2 + 1) 10^-24. P > 8: \calx_trsl.

```
%
```

```
\def\calx_trse #1.#2.#3\xint:%
{%
    \expandafter\calx_trsf\the\numexpr#1+8*#2\expandafter.%
    \the\numexpr(2*#1+7)/16\expandafter.%
    \the\numexpr#1+8-8*((2*#1+7)/16).#3\xint:%
}%
\def\calx_trsf #1.#2.#3.#4\xint:%
  {\expandafter\calx_trsg\the\numexpr\calx_trta#2.#3.#4\xint:.#1.}%
\def\calx_trsg #1.#2.%
{%
    \expandafter\calx_trsh\the\numexpr(2*#1-\CALX_pten{9-#2})%
      /(2*\CALX_pten{9-#2}).#1.#2.%
}%
\def\calx_trsh #1.%
  {\expandafter\calx_trsi\the\numexpr(2*#1-10000)/20000.#1.}%
\def\calx_trsi #1.#2.#3.#4.%
{%
    \expandafter\calx_trsj\the\numexpr#1*6366+(#1*1977236758/100000%
      +(#2-#1*10000)*636619772/10000%
      +(#3-#2*\CALX_pten{9-#4})*636619772/\CALX_pten{13-#4})/100000.%
    #3.#4.%
}%
\def\calx_trsj #1.#2.#3.%
{%
    \expandafter\calx_trsm\the\numexpr(#2-((2*#2-\CALX_pten{13-#3})%
      /(2*\CALX_pten{13-#3}))*\CALX_pten{13-#3})*\CALX_pten{#3-5}%
    \expandafter.\the\numexpr(2*#1-10000)/20000\expandafter.%
    \the\numexpr#1-4*((2*#1-3)/8).#1.%
}%
\def\calx_trsm #1.%
  {\expandafter\calx_trwk\the\numexpr(2*#1-10000)/20000.#1.}%
\def\calx_trwk #1.#2.#3.#4.#5.%
{%
    \expandafter\calx_trh\the\numexpr1200000000+#1-(#5-#3*10000)-#3*5707%
    \expandafter\calx_trp\the\numexpr1200000000+#2-#1*10000%
      -(#5-#3*10000)*5707-#3*9632%
    \expandafter\calx_trp\the\numexpr1200000000-(#5-#3*10000)*9632-#3*6794%
    \expandafter\calx_trp\the\numexpr1200000000-(#5-#3*10000)*6794-#3*8966%
    \expandafter\calx_trp\the\numexpr1200000000-(#5-#3*10000)*8966-#3*1923%
    \expandafter\calx_trp\the\numexpr1200000000-(#5-#3*10000)*1923-#3*1322%
    \expandafter\calx_trp\the\numexpr1200000000-(#5-#3*10000)*1322\relax!%
    ;3.#4.%
}%
\def\calx_trsl #1.#2.%
{%
    \expandafter\calx_trxa\the\numexpr8*((\calx_ttrigdigits+#1+8*#2+5)/8)%
    \expandafter.\the\numexpr#1+8*#2\expandafter.%
    \the\numexpr(2*#1+7)/16.#2.%
}%
\def\calx_trxa #1.#2.#3.#4.#5\xint:%
{%
    \expandafter\calx_trxb\romannumeral0\CALX_nshrr{#2+3-8*#4}%
```

```
      {\romannumeral0\CALX_nmul{N+#3.#5}%
        {\romannumeral0\CALX_nshrr{127-#2}{\calx_trT}}}%
    \xint:#1.#3.#4.#5\xint:%
}%
\def\calx_trxb N+#1.#2!#3\xint:#4.#5.#6.#7\xint:%
{%
    \expandafter\calx_tred_z\the\numexpr#2-4*((2*#2-3)/8)\expandafter.%
    \romannumeral0\CALX_fnround{\calx_ttrigdigits+4}%
      {\CALX_ffromraw{\expanded{\romannumeral0\CALX_ntoraw
        {\romannumeral0\CALX_nsub
          {\romannumeral0\CALX_nshl{#4+8*#6}{N+#5.#7}}%
          {\romannumeral0\CALX_nmul{N+#1.#2!#3}%
            {\romannumeral0\CALX_nshrr{129-#4}{\calx_trH}}}}%
        [-#4]}}}%
}%
```

Degree reduction \calx_tred_a (\calx_tsc_fo, route of xinttrig), Y = |x| in degrees: T = floor(Y 10^L), L = 8 ceil((P+5)/8) (xinttrig: P+5 decimals); I = floor(T/10^L), m = I mod 360 (10^8 = 280 mod 360 and 280^2 = 280 mod 360: m = (b_0 + 280 sum_{j>0} b_j) mod 360 on the blocks b_j of I); q = floor((m+45)/90) = round((2m+1)/180) in 0..4; r_fix = T - (I - m + 90q) 10^L, |r_fix| <= 45 10^L; r = round_{P+4}(r_fix 10^-L Dg), Dg = pi/180 rounded to P+4 digits; |r| <= 0.786 (u <= 0.62).
  Absolute error of r (radians): < 10^-(P+5) + 10^-(P+4) (Dg, rounding of r).
  Radians (\calx_tsin_s, \calx_tcos_s, q from \calx_tred or \calx_trslow): quadrant q mod 4: sin = S, C, -S, -C (times sign of X), cos = C, -S, -C, S, with S, C the series kernels at r (K = max(2, ceil((P+4)/8)), final rounding to P).

```
%
\def\calx_tsin_r_small #1#2.#3.#4\xint:%
{%
    \ifnum\numexpr#2+8*#3>\xint_c_iv
      \xint_dothis{\calx_tsin_r_regular#1#2.#3.#4\xint:}\fi
    \xint_orthat{\calx_trpq s#1#2.#3.#4\xint:}%
}%
\def\calx_tsin_r_regular #1#2.#3.#4\xint:%
{%
    \ifnum\numexpr#2+8*#3>\xint_c_iv \calx_trj\fi
    \expandafter\calx_tsin_s\expandafter#1%
    \romannumeral0\calx_tred #2.#3.#4\xint:\xint:%
}%
\def\calx_tcos_r_small #1#2.#3.#4\xint:%
{%
    \ifnum\numexpr#2+8*#3>\xint_c_iv
      \xint_dothis{\calx_tcos_r_regular#1#2.#3.#4\xint:}\fi
    \xint_orthat{\calx_trpq c+#2.#3.#4\xint:}%
}%
\def\calx_tcos_r_regular #1#2.#3.#4\xint:%
{%
    \ifnum\numexpr#2+8*#3>\xint_c_iv \calx_trj\fi
    \expandafter\calx_tcos_s\romannumeral0\calx_tred #2.#3.#4\xint:\xint:%
}%
```

P <= 8, 0.8 <= |x| < 10^4: reduction in \numexpr without natives (\calx_trq <A>.<E>., |x| = A 10^(E-9) exact, E <= 4), output <q>.<s><A'>.<E'>.<A>.<E>. (r = s A' 10^(E'-9), A' 9 digits):
     k = round(2|x|/pi) (c = 636619772), pi/2 = (H1 + H2 10^-9 +
     H3 10^-18) 10^-9, H1 = 1570796326, H2 = 794896619, H3 = 231321691,

```
     10^9 r = R1 - R2 10^-9,   R1 = A 10^E - k H1 - floor(k H2/10^9),
     R2 = (k H2 mod 10^9) + round(k H3/10^9)
(A 10^E - k H1 by halves of 10^5, k H2 by halves of 10^4 and 10^5: all intermediate values < 2^31).
|R1| >= 2:  A' from R1 and R2 with the digit count of |R1| (minus 1 if R1 > 0); |R1| <= 1:  from
R1 10^9 - R2 (relative accuracy kept when r is small).  Callers:  \calx_trpq (sin, cos:  \calx
↪_trps<q>, \calx_trpc<q>, kernels \calx_trks, \calx_trkk) and \calx_tsc_kd (tan, cot, sec, csc,
sinc).
3298 \def\calx_trpq #1#2#3.#4.#5\xint:%
3299 {%
3300     \expandafter\calx_trpa\the\numexpr#3+8*#4\expandafter.%
3301     \the\numexpr(2*#3+7)/16\expandafter.%
3302     \the\numexpr#3+8-8*((2*#3+7)/16).#5\xint:#1#2%
3303 }%
3304 \def\calx_trpa #1.#2.#3.#4\xint:%
3305 {%
3306     \expandafter\calx_trpb\romannumeral0\expandafter\calx_trq
3307     \the\numexpr\calx_trta#2.#3.#4\xint:.#1.%
3308 }%
3309 \def\calx_trpb #1.#2#3.#4.#5.#6.#7#8%
3310   {\csname calx_trp#7#1\endcsname#8#2#3.#4.}%
3311 \expandafter\def\csname calx_trps0\endcsname#1#2{\calx_trks#11*#21.}%
3312 \expandafter\def\csname calx_trps1\endcsname#1#2{\calx_trkk#11.}%
3313 \expandafter\def\csname calx_trps2\endcsname#1#2{\calx_trks-1*#11*#21.}%
3314 \expandafter\def\csname calx_trps3\endcsname#1#2{\calx_trkk-1*#11.}%
3315 \expandafter\def\csname calx_trpc0\endcsname#1#2{\calx_trkk1.}%
3316 \expandafter\def\csname calx_trpc1\endcsname#1#2{\calx_trks-1*#21.}%
3317 \expandafter\def\csname calx_trpc2\endcsname#1#2{\calx_trkk-1.}%
3318 \expandafter\def\csname calx_trpc3\endcsname#1#2{\calx_trks#21.}%
3319 \def\calx_trks #1.#2.#3.%
3320 {%
3321     \ifnum#3<-\xint_c_iv \xint_dothis\calx_trks_tiny\fi
3322     \xint_orthat\calx_trks_regular #1.#2.#3.%
3323 }%
3324 \def\calx_trks_tiny #1.#2.#3.%
3325   {\expandafter\calx_tdgo\the\numexpr#1*#2\expandafter.\the\numexpr#3-9.}%
3326 \def\calx_trks_regular #1.#2.#3.%
3327   {\expandafter\calx_trksa\the\numexpr\calx_tqkz#2.#3..#1.#2.#3.}%
3328 \def\calx_trksa #1.#2.#3.#4.%
3329 {%
3330     \expandafter\calx_tdgo
3331     \the\numexpr#2*#3*(\calx_tsinaux#1.)/1000000000.#4-9.%
3332 }%
3333 \def\calx_trkk #1.#2.#3.%
3334 {%
3335     \expandafter\calx_tdgo\the\numexpr#1*(%
3336       \expandafter\calx_tcosaux\the\numexpr\calx_tqkz#2.#3..).-9.%
3337 }%
3338 \def\calx_trq #1.#2.%
3339 {%
3340     \expandafter\calx_trqa
3341     \the\numexpr(#1*636619772/1000000000)/\CALX_pten{9-#2}.#1.#2.%
3342 }%
```

```
\def\calx_trqa #1.#2.#3.%
{%
    \expandafter\calx_trqb\the\numexpr(2*#2-\CALX_pten{5-#3})%
      /(2*\CALX_pten{5-#3}).#1.#2.#3.%
}%
\def\calx_trqb #1.#2.#3.#4.%
{%
    \expandafter\calx_trqc\the\numexpr(#1-#2*15707)*100000%
      +(#3-#1*\CALX_pten{5-#4})*\CALX_pten{#4}-#2*96326.#2.#3.#4.%
}%
\def\calx_trqc #1.#2.%
  {\expandafter\calx_trqd\the\numexpr(2*#2*7948-10000)/20000.#1.#2.}%
\def\calx_trqd #1.#2.#3.%
{%
    \expandafter\calx_trqe\the\numexpr(#3*7948-#1*10000)*100000+#3*96619.%
    #1.#2.#3.%
}%
\def\calx_trqe #1.#2.#3.#4.%
{%
    \expandafter\calx_trqf\the\numexpr#3-#2-\ifnum#1<1000000000 0\else1\fi
    \expandafter.\the\numexpr#1-\ifnum#1<1000000000 0\else1000000000\fi
    +#4*231321691/1000000000.#4.%
}%
\def\calx_trqf #1.#2.#3.%
{%
    \expandafter\calx_trqg\the\numexpr#3-4*((2*#3-3)/8)\expandafter.%
    \romannumeral0\ifnum\ifnum#1<\xint_c_ -\fi#1<\xint_c_ii
      \expandafter\calx_trqs\else\expandafter\calx_trql\fi #1.#2.%
}%
\def\calx_trqg #1.#2.#3.{ #1.#2.#3.}%
\def\calx_trql #1.#2.%
{%
    \expandafter\calx_trqm\the\numexpr\ifnum#1>\xint_c_ #1-1\else-#1\fi
    .#1.#2.%
}%
\def\calx_trqm #1.%
  {\expandafter\calx_trqn\the\numexpr\CALX_nd_c #19987654321\xint:.}%
\def\calx_trqn #1.#2.#3.%
{%
    \ifnum#2>\xint_c_ \expandafter\calx_trqp\else\expandafter\calx_trqo\fi
    #1.#2.#3.%
}%
\def\calx_trqp #1.#2.#3.%
{%
    \expandafter\calx_trqw\the\numexpr#2*\CALX_pten{9-#1}-#3/\CALX_pten{#1}%
    \expandafter.\the\numexpr#1-9.+%
}%
\def\calx_trqo #1.#2.#3.%
{%
    \expandafter\calx_trqw\the\numexpr-#2*\CALX_pten{9-#1}+#3/\CALX_pten{#1}%
    \expandafter.\the\numexpr#1-9.-%
}%
```

```
\def\calx_trqw #1.#2.#3{ #3#1.#2.}%
\def\calx_trqs #1.#2.%
  {\expandafter\calx_trqt\the\numexpr#1*1000000000-#2.}%
\def\calx_trqt #1#2.%
{%
    \if-#1\expandafter\calx_trqu\expandafter-\else
      \expandafter\calx_trqu\expandafter+\expandafter#1\fi #2.%
}%
\def\calx_trqu #1#2.%
{%
    \expandafter\calx_trqv\the\numexpr
    \CALX_nd_c #2{10}987654321\xint:.#1#2.%
}%
\def\calx_trqv #1.#2#3.%
{%
    \expandafter\calx_trqw\the\numexpr\ifnum#1=10 #3/10\else
      #3*\CALX_pten{9-#1}\fi\expandafter.\the\numexpr#1-18.#2%
}%
\def\calx_tsin_s #1#2.#3\xint:%
{%
    \ifcase#2 \expandafter\calx_tfin_p\expandafter s%
    \or       \expandafter\calx_tfin_p\expandafter c%
    \or       \expandafter\calx_tfin_n\expandafter s%
    \else     \expandafter\calx_tfin_n\expandafter c%
    \fi #1{#3}%
}%
\def\calx_tcos_s #1.#2\xint:%
{%
    \ifcase#1 \expandafter\calx_tfin_p\expandafter c%
    \or       \expandafter\calx_tfin_n\expandafter s%
    \or       \expandafter\calx_tfin_n\expandafter c%
    \else     \expandafter\calx_tfin_p\expandafter s%
    \fi +{#2}%
}%
\def\calx_tfin_p #1#2%
  {\if-#2\expandafter\calx_tfin_neg\else\expandafter\calx_tser\fi{#1}}%
\def\calx_tfin_n #1#2%
  {\if-#2\expandafter\calx_tser\else\expandafter\calx_tfin_neg\fi{#1}}%
\def\calx_tfin_neg #1#2%
  {\expandafter\CALX_fopp_a\romannumeral0\calx_tser{#1}{#2}\xint:}%
\def\calx_tser #1#2{\calx_tser_a #1#2\xint:}%
\def\calx_tser_a #1F#2\xint:%
{%
    \if s#1\expandafter\xint_firstoftwo\else\expandafter\xint_secondoftwo\fi
    {\calx_tsin_kernel}{\calx_tcos_kernel}#2\xint:%
}%
\def\calx_tred_a F+#1.#2.#3\xint:%
{%
    \expandafter\calx_tred_b\romannumeral0\CALX_nshr
      {\numexpr-8*#2-\calx_ttrigL}{\romannumeral0\CALX_nshl
      {\numexpr8*#2+\calx_ttrigL}{\expanded{N+\the\numexpr(2*#1+7)/16.#3}}}%
    \xint:%
```

```
}%
\def\calx_tred_b #1\xint:%
{%
    \expandafter\calx_tred_c
    \romannumeral0\CALX_nshr{\calx_ttrigL}{#1}\xint:#1\xint:%
}%
\def\calx_tred_c #1\xint:#2\xint:%
{%
    \expandafter\calx_tred_d\the\numexpr\calx_tmmod #1\xint:.#1\xint:#2\xint:%
}%
\def\calx_tred_d #1.%
  {\expandafter\calx_tred_e\the\numexpr(2*#1+1)/180.#1.}%
\def\calx_tred_e #1.#2.#3\xint:#4\xint:%
{%
    \expandafter\calx_tred_f\romannumeral0\CALX_nsub{#4}%
      {\romannumeral0\CALX_nshl{\calx_ttrigL}%
        {\romannumeral0\CALX_nsub{#3}%
          {\expanded{\calx_tnsmall{\numexpr#2-90*#1}}}}}%
    \xint:#1.%
}%
\def\calx_tred_f #1\xint:#2.%
{%
    \expandafter\calx_tred_g\romannumeral`&&@%
      \calx_tpdg\xint:#1\xint:#2.%
}%
\def\calx_tred_g F+#1.#2.#3\xint:#4\xint:#5.%
{%
    \expandafter\calx_tred_z\the\numexpr\ifnum#5=4 0\else#5\fi
    \expandafter.\romannumeral0\CALX_fnround{\calx_ttrigdigits+4}%
      {\CALX_ffromraw{\expanded{\romannumeral0\CALX_ntoraw
        {\romannumeral0\CALX_nmul{#4}%
          {\expanded{N+\the\numexpr(2*#1+7)/16.#3}}}%
        [\the\numexpr8*#2-\calx_ttrigL]}}}%
}%
\def\calx_tred_z #1.{ #1.}%
```

\calx_tmmod N<s><n>.<blocks>\xint: (inside \numexpr): I mod 360.

```
\def\calx_tmmod N#1.1#2!#3\xint:%
{%
    \expandafter\calx_tmmod_b\the\numexpr\calx_tmsum 0.#31;!.#2.%
}%
\def\calx_tmsum #1.1#2#3!%
{%
    \if;#2\xint_dothis{#1}\fi
    \xint_orthat{\expandafter\calx_tmsum\the\numexpr#1+#2#3.}%
}%
\def\calx_tmod #1{(#1-360*((2*(#1)-359)/720))}%
\def\calx_tmmod_b #1.#2.{\calx_tmod{\calx_tmod{#2}+280*\calx_tmod{#1}}}%
```

\calx_tnsmall{c}: native integer c, |c| < 10^8 (use inside \expanded).

```
\def\calx_tnsmall #1{\expandafter\calx_tnsmall_a\the\numexpr#1.}%
\def\calx_tnsmall_a #1.%
{%
    \ifnum#1=\xint_c_ N+0.\else
```

```
    N\ifnum#1<\xint_c_-\else+\fi1.%
    \the\numexpr100000000\ifnum#1<\xint_c_-(#1)\else+#1\fi!\fi
}%
```

Back to caller calx.sty. We restore letter catcodes. Memo: the docstrip extraction will add \endinput anyhow.

```
\catcode`F 11 \catcode`I 11 \catcode`N 11
```

35.14. Trigonometric functions of \xinteval, constants

Trigonometric functions of \xinteval: \calxRaw<Name> applies \calxFp<Name> (at \calx_ttrigdigits, raw argument rounded on entry) and converts the F result back to raw A[N]. One macro token per function, as required by the \xintdeffunc hooks (\XINT:NE:f:one:from:one, two).

Variables twoPi, threePiover2, Pi, Piover2, oneDegree, oneRadian (raw, P digits, from 110-digit literals), redefined when the trigonometric library is reconstructed. A future constant (e.g. for exp(1)) goes into \calx_ttrigvars too.

```
\def\calx_ttmpb #1#2%
{%
    \expandafter\edef\csname calxRaw#2\endcsname##1%
       {\noexpand\romannumeral0\noexpand\CALX_ftoraw
        {\expandafter\noexpand\csname calxFp#2\endcsname{##1}}}%
    \expandafter\edef\csname XINT_expr_func_#1\endcsname##1##2##3%
    {%
        \noexpand\expandafter ##1\noexpand\expandafter ##2%
        \noexpand\expandafter{%
        \noexpand\romannumeral`&&@\noexpand\XINT:NEhook:f:one:from:one
        {\noexpand\romannumeral`&&@\expandafter\noexpand
         \csname calxRaw#2\endcsname##3}}%
    }%
}%
\calx_ttmpb{sin}{Sin}%
\calx_ttmpb{cos}{Cos}%
\calx_ttmpb{tan}{Tan}%
\calx_ttmpb{cot}{Cot}%
\calx_ttmpb{sec}{Sec}%
\calx_ttmpb{csc}{Csc}%
\calx_ttmpb{sinc}{Sinc}%
\calx_ttmpb{sind}{Sind}%
\calx_ttmpb{cosd}{Cosd}%
\calx_ttmpb{tand}{Tand}%
\calx_ttmpb{cotd}{Cotd}%
\calx_ttmpb{secd}{Secd}%
\calx_ttmpb{cscd}{Cscd}%
\calx_ttmpb{asin}{Asin}%
\calx_ttmpb{acos}{Acos}%
\calx_ttmpb{asind}{Asind}%
\calx_ttmpb{acosd}{Acosd}%
\calx_ttmpb{atan}{Atan}%
\calx_ttmpb{atand}{Atand}%
\calx_ttmpb{asec}{Asec}%
\calx_ttmpb{acsc}{Acsc}%
\def\calx_ttmpb #1#2%
{%
```

```
    \expandafter\edef\csname calxRaw#2\endcsname##1##2%
       {\noexpand\romannumeral0\noexpand\CALX_ftoraw
        {\expandafter\noexpand\csname calxFp#2\endcsname{##1}{##2}}}%
    \expandafter\edef\csname XINT_expr_func_#1\endcsname##1##2##3%
    {%
        \noexpand\expandafter ##1\noexpand\expandafter ##2%
        \noexpand\expandafter{%
        \noexpand\romannumeral`&&@\noexpand\XINT:NEhook:f:one:from:two
        {\noexpand\romannumeral`&&@\expandafter\noexpand
         \csname calxRaw#2\endcsname##3}}%
    }%
}%
\calx_ttmpb{Arg}{Arg}%
\calx_ttmpb{Argd}{Argd}%
\calx_ttmpb{pArg}{pArg}%
\calx_ttmpb{pArgd}{pArgd}%
\calx_ttmpb{atan2}{Atantwo}%
\calx_ttmpb{atan2d}{Atantwod}%
\let\calx_ttmpb\undefined
\let\XINT_expr_func_tg\XINT_expr_func_tan
\let\XINT_expr_func_cotg\XINT_expr_func_cot
\let\XINT_expr_func_arcsin\XINT_expr_func_asin
\let\XINT_expr_func_arccos\XINT_expr_func_acos
\let\XINT_expr_func_arctg\XINT_expr_func_atan
\let\XINT_expr_func_arcsec\XINT_expr_func_asec
\let\XINT_expr_func_arccsc\XINT_expr_func_acsc
\def\calx_ttrigvars
{%
\xintdefloatvar twoPi := 0+
6.28318530717958647692528676655900576839433879875021164%
19498919846156328125724179972560696506842341359642961 73;%
\xintdefloatvar threePiover2 := 0+
4.71238898038468985769396507491925432629575409906265873%
14624168884617246094293134979420522380131756019732221298;%
\xintdefloatvar Pi := 0+
3.14159265358979323846264338327950288419716939937510582%
09749445923078164062862089986280348253421170679821480865;%
\xintdefloatvar Piover2 := 0+
1.57079632679489661923132169163975144209858469968755291%
04874722961539082031431044993140174126710585339910740433;%
\xintdefloatvar oneDegree := 0+
0.01745329251994329576923690768488612713442871888541725%
45609719144017100911460344944368224156963450948221230449 25;%
\xintdefloatvar oneRadian := 0+
57.29577951308232087679815481410517033240547246656432 15%
49160243861202847148321552632440968995851110944186223382;%
}%
```

The \endinput added during docstrip extraction.

35.15. Reconstruction at the scientific precision

The data file executes only the schedule records for the working precision. For P <= 8 the inverse functions use the fixed schedule for P = 9. All installed state follows \XINT_global. A local installation must be rebuilt if a global one is requested, even at the same precision.

```
\mathchardef\calx_ttrigdigits=0
\mathchardef\calx_toutdigits=0
\mathchardef\calx_tscope=1
\def\calx_tclear #1{%
    \XINT_global\expandafter\let\csname calx_t#1\endcsname\undefined
}%
\def\calx_tclearchains #1.{%
    \calx_tclear{gs0_#1}\calx_tclear{gc0_#1}%
    \calx_tclear{tfs0_#1}\calx_tclear{tfc0_#1}%
    \ifnum#1<15 \expandafter\calx_tclearchains\the\numexpr#1+1\expandafter.\fi
}%
```

Source coefficients are used only while generating schedules. Their assignments and removal are local even for a global installation. \calx_tcoeffonly selects coefficient input without trig reconstruction.

```
\def\calx_treadData{%
    \edef\calx_trestore{\XINTrestorecatcodes\catcode63=\the\catcode63\relax}%
    \XINTsetcatcodes
    \input calxtrigdata.tex\relax
    \calx_trestore\let\calx_trestore\undefined
}%
\def\calx_tfreecoeff #1.{%
    \expandafter\let\csname calx_tcds#1\endcsname\undefined
    \expandafter\let\csname calx_tcdc#1\endcsname\undefined
    \expandafter\let\csname calx_tcms#1\endcsname\undefined
    \expandafter\let\csname calx_tcmc#1\endcsname\undefined
    \ifnum#1<40 \expandafter\calx_tfreecoeff\the\numexpr#1+1\expandafter.\fi
}%
\def\calx_tbuildworking{%
    \XINT_global\mathchardef\calx_ttrigL=\numexpr8*((\calx_ttrigdigits+8)/8)\relax
    \XINT_global\mathchardef\calx_trlimit=\numexpr\ifnum\calx_ttrigdigits>63 126-\fi
        \calx_ttrigdigits\relax
    \ifnum\calx_ttrigdigits<9
        \XINT_global\let\calx_tdg_a\calx_tdg_small
        \XINT_global\let\calx_tad_a\calx_tad_small
        \XINT_global\let\calx_tae_a\calx_tae_small
        \XINT_global\let\calx_ttq_a\calx_ttq_small
        \XINT_global\let\calx_tsc_r\calx_tsc_rq
        \XINT_global\let\calx_trslow\calx_trse
        \XINT_global\let\calx_tsin_r\calx_tsin_r_small
        \XINT_global\let\calx_tcos_r\calx_tcos_r_small
        \XINT_global\mathchardef\calx_ttrigm=2
    \else
        \XINT_global\let\calx_tdg_a\calx_tsc_a
        \XINT_global\let\calx_tad_a\calx_tas_a
        \XINT_global\let\calx_tae_a\calx_tac_a
        \XINT_global\let\calx_ttq_a\calx_tat_a
        \XINT_global\let\calx_tsc_r\calx_tsc_r_regular
```

```
        \XINT_global\let\calx_trslow\calx_trsl
        \XINT_global\let\calx_tsin_r\calx_tsin_r_regular
        \XINT_global\let\calx_tcos_r\calx_tcos_r_regular
        \XINT_global\mathchardef\calx_ttrigm=\numexpr(\calx_ttrigdigits+9)/8\relax
    \fi
    \calx_tclear{tfs0}\calx_tclear{tfc0}%
    \calx_tclear{gs0}\calx_tclear{gc0}\calx_tclear{gqs0}\calx_tclear{gqc0}%
    \calx_tclear{tss0}\calx_tclear{tsc0}%
    \calx_tclear{tsSs0}\calx_tclear{tsSc0}%
    \calx_tclear{tsQs0}\calx_tclear{tsQc0}%
    \calx_tclear{tsFs0}\calx_tclear{tsFc0}%
    \calx_tclearchains 2.%
    \calx_treadData
    \calx_tfreecoeff 0.%
    \let\calx_tgacc\undefined \let\calx_tgtmp\undefined
    \let\calx_ttacc\undefined
    \XINT_global\edef\calx_tpdg{\romannumeral0\CALX_fnround
                                {\calx_ttrigdigits+4}{\calx_ttrigDg}}%
    \ifnum\calx_ttrigdigits<9
        \XINT_global\let\calx_tsin_kernel\calx_tsin_q
        \XINT_global\let\calx_tcos_kernel\calx_tcos_q
    \else\ifnum\calx_ttrigdigits<41
        \XINT_global\let\calx_tsin_kernel\calx_tsin_fixed
        \XINT_global\let\calx_tcos_kernel\calx_tcos_fixed
    \else
        \XINT_global\let\calx_tsin_kernel\calx_tsin_scheduled
        \XINT_global\let\calx_tcos_kernel\calx_tcos_scheduled
    \fi\fi
    \ifnum\calx_ttrigdigits>32
        \XINT_global\let\calx_tas_kc\calx_tas_kb
    \else
        \XINT_global\let\calx_tas_kc\calx_tas_ka
    \fi
}%
\def\calx_tinstall#1{%
    \ifnum\calx_ttrigdigits=\calx_toutdigits
       \XINT_global\expandafter\let\csname calxFp#1\expandafter\endcsname
          \csname calx_t#1\endcsname
    \else
       \XINT_global\expandafter\edef\csname calxFp#1\endcsname##1{%
          \noexpand\romannumeral0\noexpand\CALX_fnround\noexpand\calx_toutdigits
          {\expandafter\noexpand\csname calx_t#1\endcsname{##1}}}%
    \fi
}%
\def\calx_tinstalltwo#1{%
    \ifnum\calx_ttrigdigits=\calx_toutdigits
       \XINT_global\expandafter\let\csname calxFp#1\expandafter\endcsname
          \csname calx_t#1\endcsname
    \else
       \XINT_global\expandafter\edef\csname calxFp#1\endcsname##1##2{%
          \noexpand\romannumeral0\noexpand\CALX_fnround\noexpand\calx_toutdigits
          {\expandafter\noexpand\csname calx_t#1\endcsname{##1}{##2}}}%
```

```
    \fi
}%
```

35.16. \xintreloadcalxtrig, \xintreloadxinttrig

```
\def\xintreloadcalxtrig{%
    \ifnum\XINTdigits>100
        \xintMessage{calxtrig}{Warning}{Cannot reload at \the\XINTdigits\space digits.
            Maximum is 100; retaining \the\calx_toutdigits\space digits.}%
    \else\expandafter\calx_treload\fi
}%
\def\calx_treload{%
    \ifxintglobaldefs
        \ifnum\calx_tscope=0
            \mathchardef\calx_ttrigdigits=0
            \mathchardef\calx_toutdigits=0
        \fi
    \fi
    \ifnum\calx_toutdigits=\XINTdigits\else
        \calx_treload_a
    \fi
}%
\def\calx_treload_a{%
    \XINT_global\mathchardef\calx_toutdigits=\XINTdigits
    \mathchardef\calx_tnewdigits=\numexpr4*((\calx_toutdigits+1)/4)\relax
    \ifnum\calx_ttrigdigits=\calx_tnewdigits\else
        \XINT_global\let\calx_ttrigdigits\calx_tnewdigits
        \calx_tbuildworking
    \fi
    \calx_tinstall{Sin}%
    \calx_tinstall{Cos}%
    \calx_tinstall{Asin}%
    \calx_tinstall{Acos}%
    \calx_tinstall{Asind}%
    \calx_tinstall{Acosd}%
    \calx_tinstall{Acsc}%
    \calx_tinstall{Asec}%
    \calx_tinstall{Atan}%
    \calx_tinstall{Atand}%
    \calx_tinstall{Tan}%
    \calx_tinstall{Cot}%
    \calx_tinstall{Sec}%
    \calx_tinstall{Csc}%
    \calx_tinstall{Sinc}%
    \calx_tinstall{Sind}%
    \calx_tinstall{Cosd}%
    \calx_tinstall{Tand}%
    \calx_tinstall{Cotd}%
    \calx_tinstall{Secd}%
    \calx_tinstall{Cscd}%
    \calx_tinstalltwo{Arg}%
    \calx_tinstalltwo{Argd}%
    \calx_tinstalltwo{pArg}%
```

```
    \calx_tinstalltwo{pArgd}%
    \calx_tinstalltwo{Atantwo}%
    \calx_tinstalltwo{Atantwod}%
    \calx_ttrigvars
    \XINT_global\mathchardef\calx_tscope=\ifxintglobaldefs 1%
        \else\ifnum\currentgrouplevel=0 1\else 0\fi\fi\relax
}%
\let\xintreloadxinttrig\xintreloadcalxtrig
\xintreloadcalxtrig
```

---

36. Library calxtrig data tables

Added at calx 2.0a1.

calxtrigdata.tex: coefficient and schedule data from gencalxdata.py. Data of calxtrig.tex, input by it under the standard catcode regime set by \XINTsetcatcodes. No explicit \endinput is added at the file end, as it will get added by docstrip extraction.

This file is loaded from calxtrig.tex in a context having done \XINTsetcatcodes, in particular the exclamation mark has catcode 12.

The definition of \ver@calxtrigdata.tex control sequence done also under existence of LaTeX \ProvidesFile is due to a miniltx feature (under investigation).

```
\ifx\calx_tdataready\undefined
\ifdefined\ProvidesFile
  \ProvidesFile{calxtrigdata.tex}%
  [2026/11/26 v2.0a2 A calx component (JFB)]%
\fi
\expandafter\def\csname ver@calxtrigdata.tex\endcsname
  {2026/11/26 v2.0a2 A calx component (JFB)}%
\let\calx_tdataready\empty
\fi
\@namedef{calx_tcds0}{0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!}%
\@namedef{calx_tcms0}{0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!}%
\@namedef{calx_tcds1}{6666!6666!6666!6666!6666!6666!6666!6666!6666!6666!%
6666!6666!6666!6666!6666!6666!6666!6666!6666!6666!6666!6666!6666!6666!%
6666!6666!6666!6666!6666!6666!6666!6666!6666!1666!}%
\@namedef{calx_tcms1}{1666!6666!6666!6666!6666!6666!6666!6666!6666!6666!%
6666!6666!6666!6666!6666!6666!6666!6666!6666!6666!6666!6666!6666!6666!%
6666!6666!6666!6666!6666!6666!6666!6666!6666!6666!}%
\@namedef{calx_tcds2}{3333!3333!3333!3333!3333!3333!3333!3333!3333!3333!%
3333!3333!3333!3333!3333!3333!3333!3333!3333!3333!3333!3333!3333!3333!%
3333!3333!3333!3333!3333!3333!3333!3333!3333!0083!}%
\@namedef{calx_tcms2}{0083!3333!3333!3333!3333!3333!3333!3333!3333!3333!%
3333!3333!3333!3333!3333!3333!3333!3333!3333!3333!3333!3333!3333!3333!%
3333!3333!3333!3333!3333!3333!3333!3333!3333!3333!}%
\@namedef{calx_tcds3}{9841!4126!2698!9841!4126!2698!9841!4126!2698!9841!%
4126!2698!9841!4126!2698!9841!4126!2698!9841!4126!2698!9841!4126!2698!9841!%
4126!2698!9841!4126!2698!9841!4126!2698!9841!0001!}%
\@namedef{calx_tcms3}{0001!9841!2698!4126!9841!2698!4126!9841!2698!4126!%
9841!2698!4126!9841!2698!4126!9841!2698!4126!9841!2698!4126!9841!2698!4126!%
9841!2698!4126!9841!2698!4126!9841!2698!4126!9841!}%
\@namedef{calx_tcds4}{3192!2557!9065!9858!9223!5731!6525!5890!2398!3192!%
2557!9065!9858!9223!5731!6525!5890!2398!3192!2557!9065!9858!9223!5731!6525!%
5890!2398!3192!2557!9065!9858!9223!5731!0275!0000!}%
\@namedef{calx_tcms4}{0000!0275!5731!9223!9858!9065!2557!3192!2398!5890!%
6525!5731!9223!9858!9065!2557!3192!2398!5890!6525!5731!9223!9858!9065!2557!%
3192!2398!5890!6525!5731!9223!9858!9065!2557!3192!}%
\@namedef{calx_tcds5}{5210!7750!1718!8544!1083!5052!1877!4417!8385!5210!%
7750!1718!8544!1083!5052!1877!4417!8385!5210!7750!1718!8544!1083!5052!1877!%
4417!8385!5210!7750!1718!8544!1083!5052!0002!0000!}%
```

```
\@namedef{calx_tcms5}{0000!0002!5052!1083!8544!1718!7750!5210!8385!4417!%
1877!5052!1083!8544!1718!7750!5210!8385!4417!1877!5052!1083!8544!1718!7750!%
5210!8385!4417!1877!5052!1083!8544!1718!7750!5210!}%
\@namedef{calx_tcds6}{9392!1459!8216!3836!5904!2660!8281!0348!7105!2725!%
4793!1549!7170!9237!5993!1614!3682!0438!6059!8126!4882!0503!2571!9327!4947!%
7015!3771!9392!1459!8216!3836!5904!0160!0000!0000!}%
\@namedef{calx_tcms6}{0000!0000!0160!5904!3836!8216!1459!9392!3771!7015!%
4947!9327!2571!0503!4882!8126!6059!0438!3682!1614!5993!9237!7170!1549!4793!%
2725!7105!0348!8281!2660!5904!3836!8216!1459!9392!}%
\@namedef{calx_tcds7}{2854!9340!5562!9732!7170!8631!6610!5001!1700!3489!%
2832!4769!0367!0663!7838!7245!8493!0906!4124!6324!3975!1002!4155!7044!7880!%
1985!0113!4759!9816!3181!1637!7647!0000!0000!0000!}%
\@namedef{calx_tcms7}{0000!0000!0000!7647!1637!3181!9816!4759!0113!1985!%
7880!7044!4155!1002!3975!6324!4124!0906!8493!7245!7838!0663!0367!4769!2832!%
3489!1700!5001!6610!8631!7170!9732!5562!9340!2854!}%
\@namedef{calx_tcds8}{2216!0990!6638!5991!4989!0693!5061!7812!9454!5306!%
5929!7039!0258!5223!9918!4033!9222!3973!3103!2045!2735!3349!0162!0320!8301!%
9455!3198!2076!3455!7254!1145!0028!0000!0000!0000!}%
\@namedef{calx_tcms8}{0000!0000!0000!0028!1145!7254!3455!2076!3198!9455!%
8301!0320!0162!3349!2735!2045!3103!3973!9222!4033!9918!5223!0258!7039!5929!%
5306!9454!7812!5061!0693!4989!5991!6638!0990!2216!}%
\@namedef{calx_tcds9}{6497!0704!7417!7239!3201!8452!4166!8385!0904!3378!%
5017!0254!1375!1828!7368!7321!4266!4134!2611!9918!0738!4922!2807!6872!8123!%
9559!9716!2432!2466!0635!0822!0000!0000!0000!0000!}%
\@namedef{calx_tcms9}{0000!0000!0000!0000!0822!0635!2466!2432!9716!9559!%
8123!6872!2807!4922!0738!9918!2611!4134!4266!7321!7368!1828!1375!0254!5017!%
3378!0904!8385!4166!8452!3201!7239!7417!0704!6497!}%
\@namedef{calx_tcds10}{6777!8501!9993!2421!5055!3972!8700!6781!2811!6603!%
1297!1310!1146!5909!8136!3969!5105!2176!9006!4737!5287!3035!5054!4373!4757!%
2308!1261!6339!9410!9572!0001!0000!0000!0000!0000!}%
\@namedef{calx_tcms10}{0000!0000!0000!0000!0001!9572!9410!6339!1261!2308!%
4757!4373!5054!3035!5287!4737!9006!2176!5105!3969!8136!5909!1146!1310!1297!%
6603!2811!6781!8700!3972!5055!2421!9993!8501!6777!}%
\@namedef{calx_tcds11}{8214!0431!3162!8127!1274!5818!7151!8570!9313!2522!%
6445!3164!5081!7442!2960!6470!4713!2573!2646!7934!3231!2812!3152!9119!7716!%
8403!6306!0170!6817!0038!0000!0000!0000!0000!0000!}%
\@namedef{calx_tcms11}{0000!0000!0000!0000!0000!0038!6817!0170!6306!8403!%
7716!9119!3152!2812!3231!7934!2646!2573!4713!6470!2960!7442!5081!3164!6445!%
2522!9313!8570!7151!5818!1274!8127!3162!0431!8214!}%
\@namedef{calx_tcds12}{0530!6034!8788!1246!0302!9193!9511!5230!8715!7420!%
9410!8021!4041!6012!1171!1894!2891!4104!8904!0529!6872!9204!5321!1948!3396!%
8447!2843!6950!0644!0000!0000!0000!0000!0000!0000!}%
\@namedef{calx_tcms12}{0000!0000!0000!0000!0000!0000!0644!6950!2843!8447!%
3396!1948!5321!9204!6872!0529!8904!4104!2891!1894!1171!6012!4041!8021!9410!%
7420!8715!5230!9511!9193!0302!1246!8788!6034!0530!}%
\@namedef{calx_tcds13}{0912!5735!4143!8178!2464!6010!1295!3070!1750!3999!%
9315!9284!0945!4923!3363!3179!9434!6871!7861!1339!4739!6836!2571!1484!5546!%
6379!6898!9183!0000!0000!0000!0000!0000!0000!0000!}%
\@namedef{calx_tcms13}{0000!0000!0000!0000!0000!0000!0000!9183!6898!6379!%
5546!1484!2571!6836!4739!1339!7861!6871!9434!3179!3363!4923!0945!9284!9315!%
3999!1750!3070!1295!6010!2464!8178!4143!5735!0912!}%
\@namedef{calx_tcds14}{6885!6398!8502!9024!4411!1571!4164!1161!3438!8810!%
```

```
4050!0713!0974!0437!9844!0865!0147!4405!6992!3831!5769!8764!3155!7169!6447!%
6288!3099!0011!0000!0000!0000!0000!0000!0000!0000!}%
\@namedef{calx_tcms14}{0000!0000!0000!0000!0000!0000!0000!0011!3099!6288!%
6447!7169!3155!8764!5769!3831!6992!4405!0147!0865!9844!0437!0974!0713!4050!%
8810!3438!1161!4164!1571!4411!9024!8502!6398!6885!}%
\@namedef{calx_tcds15}{7727!3576!8869!1740!8725!2732!8187!4280!8928!9686!%
7455!3914!4377!1871!9419!1043!4785!4972!2921!5692!7468!6299!7949!5351!0415!%
6125!0121!0000!0000!0000!0000!0000!0000!0000!0000!}%
\@namedef{calx_tcms15}{0000!0000!0000!0000!0000!0000!0000!0000!0121!6125!%
0415!5351!7949!6299!7468!5692!2921!4972!4785!1043!9419!1871!4377!3914!7455!%
9686!8928!4280!8187!2732!8725!1740!8869!3576!7727!}%
\@namedef{calx_tcds16}{7658!0353!5122!0200!6334!3894!3871!3337!0671!3636!%
1162!1907!5146!0863!7613!8040!1481!2211!3298!8149!5868!0280!7195!6207!6335!%
1151!0000!0000!0000!0000!0000!0000!0000!0000!0000!}%
\@namedef{calx_tcms16}{0000!0000!0000!0000!0000!0000!0000!0000!0000!1151!%
6335!6207!7195!0280!5868!8149!3298!2211!1481!8040!7613!0863!5146!1907!1162!%
3636!0671!3337!3871!3894!6334!0200!5122!0353!7658!}%
\@namedef{calx_tcds17}{0703!9933!6726!7260!7013!7574!4263!5557!1479!5734!%
1992!2993!9441!2546!7149!6401!7488!9228!6380!8981!9920!1890!5863!5929!9677!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!}%
\@namedef{calx_tcms17}{0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
9677!5929!5863!1890!9920!8981!6380!9228!7488!6401!7149!2546!9441!2993!1992!%
5734!1479!5557!4263!7574!7013!7260!6726!9933!0703!}%
\@namedef{calx_tcds18}{2170!4767!6431!4697!9209!1739!5476!2879!2178!2917!%
8297!5084!5427!3132!8451!9043!2287!0307!2745!1538!0713!9153!6017!2654!0007!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!}%
\@namedef{calx_tcms18}{0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0007!2654!6017!9153!0713!1538!2745!0307!2287!9043!8451!3132!5427!5084!8297!%
2917!2178!2879!5476!1739!9209!4697!6431!4767!2170!}%
\@namedef{calx_tcds19}{1141!2439!8135!0306!4102!9326!1467!7991!6917!1506!%
9857!9132!7857!0224!6949!9027!3975!1599!7694!4339!5135!9756!0246!0049!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!}%
\@namedef{calx_tcms19}{0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0049!0246!9756!5135!4339!7694!1599!3975!9027!6949!0224!7857!9132!9857!%
1506!6917!7991!1467!9326!4102!0306!8135!2439!1141!}%
\@namedef{calx_tcds20}{3897!3995!0407!0866!0039!8463!2872!0718!1870!1080!%
2762!4987!6736!0908!2120!4487!2191!7891!4510!4240!8271!9310!0298!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!}%
\@namedef{calx_tcms20}{0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0298!9310!8271!4240!4510!7891!2191!4487!2120!0908!6736!4987!2762!%
1080!1870!0718!2872!8463!0039!0866!0407!3995!3897!}%
\@namedef{calx_tcds21}{8678!7338!5747!0238!1866!1721!2327!8483!9823!1251!%
1956!5739!5950!4939!7138!5633!9829!1886!2195!6774!2108!1655!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!}%
\@namedef{calx_tcms21}{0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!1655!2108!6774!2195!1886!9829!5633!7138!4939!5950!5739!1956!%
1251!9823!8483!2327!1721!1866!0238!5747!7338!8678!}%
\@namedef{calx_tcds22}{6761!6200!8174!3969!4046!1854!8602!4261!9151!4828!%
5450!1467!1917!7972!5422!2472!9833!2803!4718!6508!8359!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!}%
\@namedef{calx_tcms22}{0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!8359!6508!4718!2803!9833!2472!5422!7972!1917!1467!5450!%
```

```
4828!9151!4261!8602!1854!4046!3969!8174!6200!6761!}%
\@namedef{calx_tcds23}{9568!1186!2839!2638!1898!7762!2302!5122!7062!2971!%
8684!4195!7105!6978!9197!8682!5938!3960!2851!8666!0003!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!}%
\@namedef{calx_tcms23}{0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0003!8666!2851!3960!5938!8682!9197!6978!7105!4195!8684!%
2971!7062!5122!2302!7762!1898!2638!2839!1186!9568!}%
\@namedef{calx_tcds24}{4638!3023!1489!8819!1437!8362!0893!8531!0253!4406!%
7499!4291!4773!8153!3515!7903!3165!4708!4397!0016!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!}%
\@namedef{calx_tcms24}{0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0016!4397!4708!3165!7903!3515!8153!4773!4291!7499!%
4406!0253!8531!0893!8362!1437!8819!1489!3023!4638!}%
\@namedef{calx_tcds25}{2052!0350!2428!9364!0831!3113!1102!0015!8455!5095!%
2124!3425!1778!0454!7268!4571!9640!4695!0064!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!}%
\@namedef{calx_tcms25}{0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0064!4695!9640!4571!7268!0454!1778!3425!2124!%
5095!8455!0015!1102!3113!0831!9364!2428!0350!2052!}%
\@namedef{calx_tcds26}{6582!9506!7968!0028!9554!0080!3930!3697!2031!3006!%
6216!6452!5000!7721!6065!1525!9245!0233!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!}%
\@namedef{calx_tcms26}{0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0233!9245!1525!6065!7721!5000!6452!6216!%
3006!2031!3697!3930!0080!9554!0028!7968!9506!6582!}%
\@namedef{calx_tcds27}{0628!5956!6460!8363!9531!5925!0361!4247!8741!9953!%
8358!3301!9466!1803!3049!6246!0787!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!}%
\@namedef{calx_tcms27}{0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0787!6246!3049!1803!9466!3301!8358!%
9953!8741!4247!0361!5925!9531!8363!6460!5956!0628!}%
\@namedef{calx_tcds28}{9586!9838!9835!4397!6334!3062!2127!4904!9053!8417!%
9441!3064!0789!0956!4957!2467!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!}%
\@namedef{calx_tcms28}{0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!2467!4957!0956!0789!3064!9441!%
8417!9053!4904!2127!3062!6334!4397!9835!9838!9586!}%
\@namedef{calx_tcds29}{3047!2618!2770!9475!0486!9168!7545!9951!3184!1624!%
0213!5936!6189!6829!7210!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!}%
\@namedef{calx_tcms29}{0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!7210!6829!6189!5936!0213!%
1624!3184!9951!7545!9168!0486!9475!2770!2618!3047!}%
\@namedef{calx_tcds30}{7044!6694!6052!3461!9967!1272!8338!1215!5039!1183!%
1683!6802!3195!9701!0001!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!}%
\@namedef{calx_tcms30}{0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0001!9701!3195!6802!1683!%
1183!5039!1215!8338!1272!9967!3461!6052!6694!7044!}%
\@namedef{calx_tcds31}{7492!2428!9584!3003!7949!4828!5442!9265!3070!0064!%
6493!6061!0438!0005!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!}%
```

```
\@namedef{calx_tcms31}{0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0005!0438!6061!6493!%
0064!3070!9265!5442!4828!7949!3003!9584!2428!7492!}%
\@namedef{calx_tcds32}{9684!3615!7413!5262!2761!0198!4196!1997!8039!3492!%
6494!1246!0012!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!}%
\@namedef{calx_tcms32}{0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0012!1246!6494!%
3492!8039!1997!4196!0198!2761!5262!7413!3615!9684!}%
\@namedef{calx_tcds33}{3233!8863!6577!3707!9851!7611!4765!5995!0354!6188!%
4189!0027!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!}%
\@namedef{calx_tcms33}{0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0027!4189!%
6188!0354!5995!4765!7611!9851!3707!6577!8863!3233!}%
\@namedef{calx_tcds34}{1737!7344!5071!4184!2864!1879!1627!6996!8516!4376!%
0058!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!}%
\@namedef{calx_tcms34}{0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0058!%
4376!8516!6996!1627!1879!2864!4184!5071!7344!1737!}%
\@namedef{calx_tcds35}{3891!1653!6025!3127!4123!0839!6793!5546!5808!0117!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!}%
\@namedef{calx_tcms35}{0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0117!5808!5546!6793!0839!4123!3127!6025!1653!3891!}%
\@namedef{calx_tcds36}{4850!8434!9557!6809!8674!7505!6808!7078!0223!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!}%
\@namedef{calx_tcms36}{0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0223!7078!6808!7505!8674!6809!9557!8434!4850!}%
\@namedef{calx_tcds37}{2134!1095!1189!8693!7307!3979!0772!0403!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!}%
\@namedef{calx_tcms37}{0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0403!0772!3979!7307!8693!1189!1095!2134!}%
\@namedef{calx_tcds38}{1762!3932!6994!8550!4052!7854!0688!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!}%
\@namedef{calx_tcms38}{0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0688!7854!4052!8550!6994!3932!1762!}%
\@namedef{calx_tcds39}{6923!9393!6563!6213!7952!1117!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!}%
\@namedef{calx_tcms39}{0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!1117!7952!6213!6563!9393!6923!}%
\@namedef{calx_tcds40}{7582!2351!8848!9926!1724!0000!0000!0000!0000!0000!%
```

```
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!}%
\@namedef{calx_tcms40}{0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!1724!9926!8848!2351!7582!}%
\@namedef{calx_tcdc0}{0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!}%
\@namedef{calx_tcmc0}{0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!}%
\@namedef{calx_tcdc1}{0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!5000!}%
\@namedef{calx_tcmc1}{5000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!}%
\@namedef{calx_tcdc2}{6666!6666!6666!6666!6666!6666!6666!6666!6666!6666!%
6666!6666!6666!6666!6666!6666!6666!6666!6666!6666!6666!6666!6666!6666!%
6666!6666!6666!6666!6666!6666!6666!6666!6666!0416!}%
\@namedef{calx_tcmc2}{0416!6666!6666!6666!6666!6666!6666!6666!6666!6666!%
6666!6666!6666!6666!6666!6666!6666!6666!6666!6666!6666!6666!6666!6666!%
6666!6666!6666!6666!6666!6666!6666!6666!6666!6666!}%
\@namedef{calx_tcdc3}{8888!8888!8888!8888!8888!8888!8888!8888!8888!8888!%
8888!8888!8888!8888!8888!8888!8888!8888!8888!8888!8888!8888!8888!8888!%
8888!8888!8888!8888!8888!8888!8888!8888!8888!0013!}%
\@namedef{calx_tcmc3}{0013!8888!8888!8888!8888!8888!8888!8888!8888!8888!%
8888!8888!8888!8888!8888!8888!8888!8888!8888!8888!8888!8888!8888!8888!%
8888!8888!8888!8888!8888!8888!8888!8888!8888!8888!}%
\@namedef{calx_tcdc4}{8730!3015!1587!8730!3015!1587!8730!3015!1587!8730!%
3015!1587!8730!3015!1587!8730!3015!1587!8730!3015!1587!8730!3015!1587!8730!%
3015!1587!8730!3015!1587!8730!3015!1587!2480!0000!}%
\@namedef{calx_tcmc4}{0000!2480!1587!3015!8730!1587!3015!8730!1587!3015!%
8730!1587!3015!8730!1587!3015!8730!1587!3015!8730!1587!3015!8730!1587!3015!%
8730!1587!3015!8730!1587!3015!8730!1587!3015!8730!}%
\@namedef{calx_tcdc5}{7319!5255!8906!3985!1922!5573!0652!8589!2239!7319!%
5255!8906!3985!1922!5573!0652!8589!2239!7319!5255!8906!3985!1922!5573!0652!%
8589!2239!7319!5255!8906!3985!1922!5573!0027!0000!}%
\@namedef{calx_tcmc5}{0000!0027!5573!1922!3985!8906!5255!7319!2239!8589!%
0652!5573!1922!3985!8906!5255!7319!2239!8589!0652!5573!1922!3985!8906!5255!%
7319!2239!8589!0652!5573!1922!3985!8906!5255!7319!}%
\@namedef{calx_tcdc6}{2100!8979!6809!9878!6756!4587!7656!4534!2365!5434!%
2312!0143!3212!0090!7921!0989!7868!5698!8767!5645!3476!6545!3423!1254!4323!%
1201!9032!2100!8979!6809!9878!6756!2087!0000!0000!}%
\@namedef{calx_tcmc6}{0000!0000!2087!6756!9878!6809!8979!2100!9032!1201!%
4323!1254!3423!6545!3476!5645!8767!5698!7868!0989!7921!0090!3212!0143!2312!%
5434!2365!4534!7656!4587!6756!9878!6809!8979!2100!}%
\@namedef{calx_tcdc7}{2813!0104!3444!5988!7564!9475!9162!5024!5507!2337!%
2485!1539!5512!9945!7570!8686!7405!3602!1861!4866!9634!5035!2326!5666!8210!%
9786!1697!1385!7247!7729!4559!4707!0011!0000!0000!}%
\@namedef{calx_tcmc7}{0000!0000!0011!4707!4559!7729!7247!1385!1697!9786!%
8210!5666!2326!5035!9634!4866!1861!3602!7405!8686!7570!9945!5512!1539!2485!%
```

```
2337!5507!5024!9162!9475!7564!5988!3444!0104!2813!}%
\@namedef{calx_tcdc8}{7678!6833!2847!1858!4823!1789!6038!2812!0731!0218!%
0802!9673!4397!8791!8614!8577!6780!7556!2757!4770!6498!6937!2759!5440!1117!%
0749!4382!5297!8738!3323!9477!0477!0000!0000!0000!}%
\@namedef{calx_tcmc8}{0000!0000!0000!0477!9477!3323!8738!5297!4382!0749!%
1117!5440!2759!6937!6498!4770!2757!7556!6780!8577!8614!8791!4397!9673!0802!%
0218!0731!2812!6038!1789!4823!1858!2847!6833!7678!}%
\@namedef{calx_tcdc9}{3456!3388!0924!7555!0832!0594!9170!9322!7191!4183!%
5329!4835!6125!4734!9995!9112!1067!8554!9616!8446!4040!3519!3342!0573!4350!%
1636!4622!6226!6858!2069!5619!0001!0000!0000!0000!}%
\@namedef{calx_tcmc9}{0000!0000!0000!0001!5619!2069!6858!6226!4622!1636!%
4350!0573!3342!3519!4040!8446!9616!8554!1067!9112!9995!4734!6125!4835!5329!%
4183!7191!9322!9170!0594!0832!7555!0924!3388!3456!}%
\@namedef{calx_tcdc10}{2324!8535!9870!0861!6160!3422!2708!2419!9045!8668!%
7250!7512!4068!4091!0868!3366!7213!5706!9130!9495!1036!3746!6140!1843!9906!%
8477!6485!3121!7623!1031!0041!0000!0000!0000!0000!}%
\@namedef{calx_tcmc10}{0000!0000!0000!0000!0041!1031!7623!3121!6485!8477!%
9906!1843!6140!3746!1036!9495!9130!5706!7213!3366!0868!4091!4068!7512!7250!%
8668!9045!2419!2708!3422!6160!0861!9870!8535!2324!}%
\@namedef{calx_tcdc11}{8944!9931!2726!6928!9320!3816!4486!7126!4218!8027!%
8240!2786!6870!1177!8097!8816!8413!9189!0863!2488!4331!4683!2502!9744!7488!%
3286!5057!3924!6791!0889!0000!0000!0000!0000!0000!}%
\@namedef{calx_tcmc11}{0000!0000!0000!0000!0000!0889!6791!3924!5057!3286!%
7488!9744!2502!4683!4331!2488!0863!9189!8413!8816!8097!1177!6870!2786!8240!%
8027!4218!7126!4486!3816!9320!6928!2726!9931!8944!}%
\@namedef{calx_tcdc12}{3258!0851!9715!1171!7553!9825!7797!0773!7888!5521!%
5268!0548!1045!0310!9290!7352!2279!2607!2610!3247!1801!0117!3048!8713!4904!%
1183!1096!3757!6117!0001!0000!0000!0000!0000!0000!}%
\@namedef{calx_tcmc12}{0000!0000!0000!0000!0000!0001!6117!3757!1096!1183!%
4904!8713!3048!0117!1801!3247!2610!2607!2279!7352!9290!0310!1045!0548!5268!%
5521!7888!0773!7797!9825!7553!1171!9715!0851!3258!}%
\@namedef{calx_tcdc13}{4635!4847!1876!0817!6550!2276!4981!2893!7258!7977!%
1515!0693!5540!2923!0814!5842!4726!5542!2265!6174!7956!4584!9435!0074!9746!%
2247!6263!7959!0024!0000!0000!0000!0000!0000!0000!}%
\@namedef{calx_tcmc13}{0000!0000!0000!0000!0000!0000!0024!7959!6263!2247!%
9746!0074!9435!4584!7956!6174!2265!5542!4726!5842!0814!2923!5540!0693!1515!%
7977!7258!2893!4981!2276!6550!0817!1876!4847!4635!}%
\@namedef{calx_tcdc14}{9675!5561!6576!1720!7945!5571!0760!3681!9705!5499!%
7475!0688!8248!2675!5477!5113!4265!7745!2780!1119!7312!4172!1520!7910!6983!%
2370!9889!0327!0000!0000!0000!0000!0000!0000!0000!}%
\@namedef{calx_tcmc14}{0000!0000!0000!0000!0000!0000!0000!0327!9889!2370!%
6983!7910!1520!4172!7312!1119!2780!7745!4265!5113!5477!2675!8248!0688!7475!%
5499!9705!3681!0760!5571!7945!1720!6576!5561!9675!}%
\@namedef{calx_tcdc15}{9562!0879!4950!3967!0480!4719!3805!2705!6781!0293!%
1135!1357!5699!8014!1994!2362!8338!4146!0566!6461!1525!5292!6438!5905!2881!%
9876!3769!0000!0000!0000!0000!0000!0000!0000!0000!}%
\@namedef{calx_tcmc15}{0000!0000!0000!0000!0000!0000!0000!0000!3769!9876!%
2881!5905!6438!5292!1525!6461!0566!4146!8338!2362!1994!8014!5699!1357!1135!%
0293!6781!2705!3805!4719!0480!3967!4950!0879!9562!}%
\@namedef{calx_tcdc16}{2741!1674!9027!6616!9022!8522!7755!0133!2154!9990!%
8357!2934!9824!8495!1231!5345!8899!2967!8841!8927!3670!9259!7435!4854!9075!%
8003!0003!0000!0000!0000!0000!0000!0000!0000!0000!}%
```

```
\@namedef{calx_tcmc16}{0000!0000!0000!0000!0000!0000!0000!0000!0003!8003!%
9075!4854!7435!9259!3670!8927!8841!2967!8899!5345!1231!8495!9824!2934!8357!%
9990!2154!0133!7755!8522!9022!6616!9027!1674!2741!}%
\@namedef{calx_tcdc17}{4637!7657!5444!4123!5480!5114!9231!4509!1784!0695!%
9740!4761!0445!9143!0223!4060!2102!3006!3332!4357!7231!6184!5211!7535!8715!%
0033!0000!0000!0000!0000!0000!0000!0000!0000!0000!}%
\@namedef{calx_tcmc17}{0000!0000!0000!0000!0000!0000!0000!0000!0000!0033!%
8715!7535!5211!6184!7231!4357!3332!3006!2102!4060!0223!9143!0445!4761!9740!%
0695!1784!4509!9231!5114!5480!4123!5444!7657!4637!}%
\@namedef{calx_tcdc18}{0297!6387!7964!3812!0750!4377!2618!6543!0596!7937!%
6999!8138!0817!5904!2698!4622!4652!1367!1566!6916!6386!8663!2662!8220!0268!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!}%
\@namedef{calx_tcmc18}{0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0268!8220!2662!8663!6386!6916!1566!1367!4652!4622!2698!5904!0817!8138!6999!%
7937!0596!6543!2618!4377!0750!3812!7964!6387!0297!}%
\@namedef{calx_tcdc19}{4530!5125!7274!1965!9979!3729!7249!1654!9794!8760!%
4428!6186!6458!8766!1011!2080!5060!2376!0072!9251!0281!0504!9632!1911!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!}%
\@namedef{calx_tcmc19}{0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!1911!9632!0504!0281!9251!0072!2376!5060!2080!1011!8766!6458!6186!4428!%
8760!9794!1654!7249!3729!9979!1965!7274!5125!4530!}%
\@namedef{calx_tcdc20}{9778!3810!6703!5507!1602!6983!7786!9449!6672!4287!%
3246!4478!6196!7255!6923!3975!9849!3539!4942!3858!9128!1743!2256!0001!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!}%
\@namedef{calx_tcmc20}{0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0001!2256!1743!9128!3858!4942!3539!9849!3975!6923!7255!6196!4478!3246!%
4287!6672!9449!7786!6983!1602!5507!6703!3810!9778!}%
\@namedef{calx_tcdc21}{3188!5571!7152!0258!0239!4011!0068!4779!2425!3835!%
4113!6785!5874!2402!6955!2249!2671!1140!4393!1291!0673!1174!0007!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!}%
\@namedef{calx_tcmc21}{0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0007!1174!0673!1291!4393!1140!2671!2249!6955!2402!5874!6785!4113!%
3835!2425!4779!0068!4011!0239!0258!7152!5571!3188!}%
\@namedef{calx_tcdc22}{4288!9030!7857!8641!2087!3448!7098!1783!1814!7301!%
5271!6039!6271!8748!4025!1264!2496!6179!2322!2881!6184!0037!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!}%
\@namedef{calx_tcmc22}{0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0037!6184!2881!2322!6179!2496!1264!4025!8748!6271!6039!5271!%
7301!1814!1783!7098!3448!2087!8641!7857!9030!4288!}%
\@namedef{calx_tcdc23}{9712!5786!3438!3999!9218!4822!8230!0744!1938!9670!%
8161!7205!3954!7999!2291!8097!9126!6147!4015!7315!0181!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!}%
\@namedef{calx_tcmc23}{0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0181!7315!4015!6147!9126!8097!2291!7999!3954!7205!8161!%
9670!1938!0744!8230!4822!9218!3999!3438!5786!9712!}%
\@namedef{calx_tcdc24}{7282!8149!2975!2138!0456!9745!3797!8023!2438!5895!%
7472!0295!3898!9520!2274!7264!5123!0707!5476!0805!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!}%
\@namedef{calx_tcmc24}{0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0805!5476!0707!5123!7264!2274!9520!3898!0295!7472!%
5895!2438!8023!3797!9745!0456!2138!2975!8149!7282!}%
\@namedef{calx_tcdc25}{4692!7860!3829!7576!2428!8767!6217!0770!1205!9888!%
```

```
8349!4685!0695!3163!0670!3158!1663!9494!3287!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!}%
\@namedef{calx_tcmc25}{0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!3287!9494!1663!3158!0670!3163!0695!4685!8349!%
9888!1205!0770!6217!8767!2428!7576!3829!7860!4692!}%
\@namedef{calx_tcdc26}{8885!3852!2354!1526!6362!4290!8290!5961!7662!9328!%
9463!1988!5034!9239!1485!0857!9993!2397!0001!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!}%
\@namedef{calx_tcmc26}{0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0001!2397!9993!0857!1485!9239!5034!1988!9463!%
9328!7662!5961!8290!4290!6362!1526!2354!3852!8885!}%
\@namedef{calx_tcdc27}{4566!7583!5332!0000!4251!5927!9887!3586!0778!7463!%
9744!1600!0648!9217!7704!3546!3319!0004!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!}%
\@namedef{calx_tcmc27}{0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0004!3319!3546!7704!9217!0648!1600!9744!%
7463!0778!3586!9887!5927!4251!0000!5332!7583!4566!}%
\@namedef{calx_tcdc28}{6439!0820!0651!0685!1063!4570!1256!9540!6048!9820!%
8184!4701!4990!4496!2554!0647!0014!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!}%
\@namedef{calx_tcmc28}{0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0014!0647!2554!4496!4990!4701!8184!%
9820!6048!9540!1256!4570!1063!0685!0651!0820!6439!}%
\@namedef{calx_tcdc29}{9820!4479!3445!9041!8729!0914!5209!7153!7914!5834!%
2576!0225!5186!2947!5430!0042!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!}%
\@namedef{calx_tcmc29}{0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0042!5430!2947!5186!0225!2576!%
5834!7914!7153!5209!0914!8729!9041!3445!4479!9820!}%
\@namedef{calx_tcdc30}{9717!8376!9212!1157!8008!7652!8625!4165!7386!2193!%
2670!4932!4936!1780!0120!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!}%
\@namedef{calx_tcmc30}{0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0120!1780!4936!4932!2670!%
2193!7386!4165!8625!7652!8008!1157!9212!8376!9717!}%
\@namedef{calx_tcdc31}{2049!3011!3807!9249!0805!4214!2876!3729!3468!4051!%
9059!1883!7632!0317!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!}%
\@namedef{calx_tcmc31}{0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0317!7632!1883!9059!%
4051!3468!3729!2876!4214!0805!9249!3807!3011!2049!}%
\@namedef{calx_tcdc32}{9492!5037!1868!2078!9499!2887!2741!9832!2547!7032!%
2132!1032!0788!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!}%
\@namedef{calx_tcmc32}{0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0788!1032!2132!%
7032!2547!9832!2741!2887!9499!2078!1868!5037!9492!}%
\@namedef{calx_tcdc33}{6661!3842!0718!8413!0041!0003!9306!1696!3758!4598!%
0704!1837!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!}%
\@namedef{calx_tcmc33}{0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!1837!0704!%
```

```
4598!3758!1696!9306!0003!0041!8413!0718!3842!6661!}%
\@namedef{calx_tcdc34}{9900!6747!9949!8730!7644!9670!2275!2735!7652!2002!%
4032!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!}%
\@namedef{calx_tcmc34}{0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!4032!%
2002!7652!2735!2275!9670!7644!8730!9949!6747!9900!}%
\@namedef{calx_tcdc35}{6310!7390!7786!2059!2755!9598!2308!3814!2407!8348!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!}%
\@namedef{calx_tcmc35}{0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
8348!2407!3814!2308!9598!2755!2059!7786!7390!6310!}%
\@namedef{calx_tcdc36}{4081!5717!7722!7126!3251!7928!7038!6743!6330!0001!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!}%
\@namedef{calx_tcmc36}{0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0001!6330!6743!7038!7928!3251!7126!7722!5717!4081!}%
\@namedef{calx_tcdc37}{0065!2141!9183!1983!8090!8479!7929!0230!0003!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!}%
\@namedef{calx_tcmc37}{0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0003!0230!7929!8479!8090!1983!9183!2141!0065!}%
\@namedef{calx_tcdc38}{5685!2777!8568!8403!2069!4789!3036!0005!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!}%
\@namedef{calx_tcmc38}{0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0005!3036!4789!2069!8403!8568!2777!5685!}%
\@namedef{calx_tcdc39}{6945!2101!8551!0878!8257!8305!0008!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!}%
\@namedef{calx_tcmc39}{0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0008!8305!8257!0878!8551!2101!6945!}%
\@namedef{calx_tcdc40}{4211!0492!6707!4077!9724!0013!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!}%
\@namedef{calx_tcmc40}{0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!0000!%
0000!0000!0000!0000!0013!9724!4077!6707!0492!4211!}%
```

The coefficients above are temporary. The logarithmic library reads them too, but does not execute the trigonometric schedule records. $P \leq 8$ uses the $P = 9$ inverse schedule.

```
\ifdefined\calx_tcoeffonly\else
\ifcase\numexpr\ifnum\calx_ttrigdigits<9 0\else\calx_ttrigdigits/4-2\fi\relax
```

$P \leq 8$.

```
\calx_tfdef s{0}{4}{6}{2}{4}{2.3.3.3.3.4.4.4.4.4.4.4.}%
\calx_tfdef s{0_5}{4}{5}{3}{4}{3.3.3.4.4.4.4.4.4.4.}%
\calx_tfdef s{0_4}{4}{4}{3}{4}{3.4.4.4.4.4.4.4.}%
\calx_tfdef s{0_3}{4}{3}{4}{4}{4.4.4.4.4.4.}%
```

```
\calx_tfdef s{0_2}{4}{2}{4}{4}{4.4.4.4.}%
\calx_tfsel s{0}{6}{5.2440.4.670.3.105.2.5.}%
\calx_tfdef c{0}{4}{7}{2}{4}{2.2.2.3.3.3.3.4.4.4.4.4.4.4.}%
\calx_tfdef c{0_6}{4}{6}{2}{4}{2.3.3.3.3.4.4.4.4.4.4.4.}%
\calx_tfdef c{0_5}{4}{5}{3}{4}{3.3.3.4.4.4.4.4.4.4.}%
\calx_tfdef c{0_4}{4}{4}{3}{4}{3.4.4.4.4.4.4.4.}%
\calx_tfdef c{0_3}{4}{3}{4}{4}{4.4.4.4.4.4.}%
\calx_tfdef c{0_2}{4}{2}{4}{4}{4.4.4.4.}%
\calx_tfsel c{0}{7}{6.2817.5.963.4.227.3.28.2.1.}%
\or
```

P = 12.

```
\calx_tfdef s{0}{4}{7}{1}{4}{1.2.2.3.3.3.3.4.4.4.4.4.4.4.}%
\calx_tfdef s{0_6}{4}{6}{2}{4}{2.3.3.3.3.4.4.4.4.4.4.4.}%
\calx_tfdef s{0_5}{4}{5}{3}{4}{3.3.3.4.4.4.4.4.4.4.}%
\calx_tfdef s{0_4}{4}{4}{3}{4}{3.4.4.4.4.4.4.4.}%
\calx_tfdef s{0_3}{4}{3}{4}{4}{4.4.4.4.4.4.}%
\calx_tfdef s{0_2}{4}{2}{4}{4}{4.4.4.4.}%
\calx_tfsel s{0}{7}{6.3298.5.1163.4.278.3.35.2.1.}%
\calx_tfdef c{0}{4}{8}{1}{4}{1.2.2.2.2.3.3.3.3.4.4.4.4.4.4.4.}%
\calx_tfdef c{0_7}{4}{7}{2}{4}{2.2.2.3.3.3.3.4.4.4.4.4.4.4.}%
\calx_tfdef c{0_6}{4}{6}{2}{4}{2.3.3.3.3.4.4.4.4.4.4.4.}%
\calx_tfdef c{0_5}{4}{5}{3}{4}{3.3.3.4.4.4.4.4.4.4.}%
\calx_tfdef c{0_4}{4}{4}{3}{4}{3.4.4.4.4.4.4.4.}%
\calx_tfdef c{0_3}{4}{3}{4}{4}{4.4.4.4.4.4.}%
\calx_tfsel c{0}{8}{7.4912.6.2217.5.740.4.166.3.19.}%
\or
```

P = 16.

```
\calx_tfdef s{0}{5}{9}{1}{5}{1.2.2.2.2.3.3.4.4.4.4.5.5.5.5.5.5.5.}%
\calx_tfdef s{0_8}{5}{8}{2}{5}{2.2.2.3.3.4.4.4.4.5.5.5.5.5.5.5.}%
\calx_tfdef s{0_7}{5}{7}{2}{5}{2.3.3.4.4.4.4.5.5.5.5.5.5.5.}%
\calx_tfdef s{0_6}{5}{6}{3}{5}{3.4.4.4.4.5.5.5.5.5.5.5.}%
\calx_tfdef s{0_5}{5}{5}{4}{5}{4.4.4.5.5.5.5.5.5.5.}%
\calx_tfdef s{0_4}{5}{4}{4}{5}{4.5.5.5.5.5.5.5.}%
\calx_tfdef s{0_3}{5}{3}{5}{5}{5.5.5.5.5.5.}%
\calx_tfsel s{0}{9}{8.4990.7.2387.6.890.5.245.4.42.3.3.}%
\calx_tfdef c{0}{5}{9}{2}{5}{2.2.2.3.3.3.3.4.4.4.4.5.5.5.5.5.5.5.}%
\calx_tfdef c{0_8}{5}{8}{2}{5}{2.3.3.3.3.4.4.4.4.5.5.5.5.5.5.5.}%
\calx_tfdef c{0_7}{5}{7}{3}{5}{3.3.3.4.4.4.4.5.5.5.5.5.5.5.}%
\calx_tfdef c{0_6}{5}{6}{3}{5}{3.4.4.4.4.5.5.5.5.5.5.5.}%
\calx_tfdef c{0_5}{5}{5}{4}{5}{4.4.4.5.5.5.5.5.5.5.}%
\calx_tfdef c{0_4}{5}{4}{4}{5}{4.5.5.5.5.5.5.5.}%
\calx_tfdef c{0_3}{5}{3}{5}{5}{5.5.5.5.5.5.}%
\calx_tfsel c{0}{9}{8.3836.7.1720.6.618.5.164.4.27.3.2.}%
\or
```

P = 20.

```
\calx_tfdef s{0}{6}{10}{2}{6}{2.2.2.3.3.3.3.4.4.5.5.5.5.6.6.6.6.6.6.6.}%
\calx_tfdef s{0_9}{6}{9}{2}{6}{2.3.3.3.3.4.4.5.5.5.5.6.6.6.6.6.6.6.}%
\calx_tfdef s{0_8}{6}{8}{3}{6}{3.3.3.4.4.5.5.5.5.6.6.6.6.6.6.6.}%
\calx_tfdef s{0_7}{6}{7}{3}{6}{3.4.4.5.5.5.5.6.6.6.6.6.6.6.}%
\calx_tfdef s{0_6}{6}{6}{4}{6}{4.5.5.5.5.6.6.6.6.6.6.6.}%
\calx_tfdef s{0_5}{6}{5}{5}{6}{5.5.5.6.6.6.6.6.6.6.}%
```

```
\calx_tfdef s{0_4}{6}{4}{5}{6}{5.6.6.6.6.6.6.6.}%
\calx_tfsel s{0}{10}{9.4078.8.1953.7.776.6.243.5.53.4.6.}%
\calx_tfdef c{0}{6}{11}{1}{6}{1.2.2.3.3.3.3.4.4.4.4.5.5.5.5.6.6.6.6.6.6.6.}%
\calx_tfdef c{0_10}{6}{10}{2}{6}{2.3.3.3.3.4.4.4.4.5.5.5.5.6.6.6.6.6.6.6.}%
\calx_tfdef c{0_9}{6}{9}{3}{6}{3.3.3.4.4.4.4.5.5.5.5.6.6.6.6.6.6.6.}%
\calx_tfdef c{0_8}{6}{8}{3}{6}{3.4.4.4.4.5.5.5.5.6.6.6.6.6.6.6.}%
\calx_tfdef c{0_7}{6}{7}{4}{6}{4.4.4.5.5.5.5.6.6.6.6.6.6.6.}%
\calx_tfdef c{0_6}{6}{6}{4}{6}{4.5.5.5.5.6.6.6.6.6.6.6.}%
\calx_tfdef c{0_5}{6}{5}{5}{6}{5.5.5.6.6.6.6.6.6.6.}%
\calx_tfdef c{0_4}{6}{4}{5}{6}{5.6.6.6.6.6.6.6.}%
\calx_tfsel c{0}{11}{10.5473.9.3041.8.1405.7.542.6.163.5.34.4.4.}%
\or
```

P = 24.

```
\calx_tfdef s{0}{7}{11}{2}{7}{2.3.3.3.3.4.4.4.4.5.5.6.6.6.6.7.7.7.7.7.7.7.}%
\calx_tfdef s{0_10}{7}{10}{3}{7}{3.3.3.4.4.4.4.5.5.6.6.6.6.7.7.7.7.7.7.7.}%
\calx_tfdef s{0_9}{7}{9}{3}{7}{3.4.4.4.4.5.5.6.6.6.6.7.7.7.7.7.7.7.}%
\calx_tfdef s{0_8}{7}{8}{4}{7}{4.4.4.5.5.6.6.6.6.7.7.7.7.7.7.7.}%
\calx_tfdef s{0_7}{7}{7}{4}{7}{4.5.5.6.6.6.6.7.7.7.7.7.7.7.}%
\calx_tfdef s{0_6}{7}{6}{5}{7}{5.6.6.6.6.7.7.7.7.7.7.7.}%
\calx_tfdef s{0_5}{7}{5}{6}{7}{6.6.6.7.7.7.7.7.7.7.}%
\calx_tfdef s{0_4}{7}{4}{6}{7}{6.7.7.7.7.7.7.7.}%
\calx_tfsel s{0}{11}{10.3626.9.1779.8.754.7.263.6.70.5.12.4.1.}%
\calx_tfdef c{0}{7}{12}{2}{7}{2.2.2.3.3.4.4.4.4.5.5.5.5.6.6.6.6.7.7.7.7.7.7.
7.}%
\calx_tfdef c{0_11}{7}{11}{2}{7}{2.3.3.4.4.4.4.5.5.5.5.6.6.6.6.7.7.7.7.7.7.7.
}%
\calx_tfdef c{0_10}{7}{10}{3}{7}{3.4.4.4.4.5.5.5.5.6.6.6.6.7.7.7.7.7.7.7.}%
\calx_tfdef c{0_9}{7}{9}{4}{7}{4.4.4.5.5.5.5.6.6.6.6.7.7.7.7.7.7.7.}%
\calx_tfdef c{0_8}{7}{8}{4}{7}{4.5.5.5.5.6.6.6.6.7.7.7.7.7.7.7.}%
\calx_tfdef c{0_7}{7}{7}{5}{7}{5.5.5.6.6.6.6.7.7.7.7.7.7.7.}%
\calx_tfdef c{0_6}{7}{6}{5}{7}{5.6.6.6.6.7.7.7.7.7.7.7.}%
\calx_tfdef c{0_5}{7}{5}{6}{7}{6.6.6.7.7.7.7.7.7.7.}%
\calx_tfsel c{0}{12}{11.4708.10.2583.9.1237.8.510.7.172.6.44.5.7.}%
\or
```

P = 28.

```
\calx_tfdef s{0}{8}{13}{1}{8}{1.2.2.3.3.4.4.4.4.5.5.5.5.6.6.7.7.7.7.8.8.8.8.
8.8.8.}%
\calx_tfdef s{0_12}{8}{12}{2}{8}{2.3.3.4.4.4.4.5.5.5.5.6.6.7.7.7.7.8.8.8.8.8.
8.8.}%
\calx_tfdef s{0_11}{8}{11}{3}{8}{3.4.4.4.4.5.5.5.5.6.6.7.7.7.7.8.8.8.8.8.8.8.
}%
\calx_tfdef s{0_10}{8}{10}{4}{8}{4.4.4.5.5.5.5.6.6.7.7.7.7.8.8.8.8.8.8.8.}%
\calx_tfdef s{0_9}{8}{9}{4}{8}{4.5.5.5.5.6.6.7.7.7.7.8.8.8.8.8.8.8.}%
\calx_tfdef s{0_8}{8}{8}{5}{8}{5.5.5.6.6.7.7.7.7.8.8.8.8.8.8.8.}%
\calx_tfdef s{0_7}{8}{7}{5}{8}{5.6.6.7.7.7.7.8.8.8.8.8.8.8.}%
\calx_tfdef s{0_6}{8}{6}{6}{8}{6.7.7.7.7.8.8.8.8.8.8.8.}%
\calx_tfdef s{0_5}{8}{5}{7}{8}{7.7.7.8.8.8.8.8.8.8.}%
\calx_tfsel s{0}{13}{12.5121.11.2952.10.1498.9.669.8.254.7.77.6.17.5.2.}%
\calx_tfdef c{0}{8}{13}{2}{8}{2.3.3.3.3.4.4.5.5.5.5.6.6.6.6.7.7.7.7.8.8.8.8.
8.8.8.}%
\calx_tfdef c{0_12}{8}{12}{3}{8}{3.3.3.4.4.5.5.5.5.6.6.6.6.7.7.7.7.8.8.8.8.8.
8.8.}%
```

```
\calx_tfdef c{0_11}{8}{11}{3}{8}{3.4.4.5.5.5.5.6.6.6.6.7.7.7.7.8.8.8.8.8.8.%
}%
\calx_tfdef c{0_10}{8}{10}{4}{8}{4.5.5.5.5.6.6.6.6.7.7.7.7.8.8.8.8.8.8.}%
\calx_tfdef c{0_9}{8}{9}{5}{8}{5.5.5.6.6.6.6.7.7.7.7.8.8.8.8.8.8.8.}%
\calx_tfdef c{0_8}{8}{8}{5}{8}{5.6.6.6.6.7.7.7.7.8.8.8.8.8.8.8.}%
\calx_tfdef c{0_7}{8}{7}{6}{8}{6.6.6.7.7.7.7.8.8.8.8.8.8.8.}%
\calx_tfdef c{0_6}{8}{6}{6}{8}{6.7.7.7.7.8.8.8.8.8.8.8.}%
\calx_tfdef c{0_5}{8}{5}{7}{8}{7.7.7.8.8.8.8.8.8.8.}%
\calx_tfsel c{0}{13}{12.4168.11.2309.10.1148.9.502.8.186.7.55.6.12.5.1.}%
\or
```

P = 32.

```
\calx_tfdef s{0}{9}{14}{2}{9}{2.2.2.3.3.4.4.5.5.5.5.6.6.6.6.7.7.8.8.8.8.9.9.%
9.9.9.9.9.}%
\calx_tfdef s{0_13}{9}{13}{2}{9}{2.3.3.4.4.5.5.5.5.6.6.6.6.7.7.8.8.8.8.9.9.9.%
9.9.9.9.}%
\calx_tfdef s{0_12}{9}{12}{3}{9}{3.4.4.5.5.5.5.6.6.6.6.7.7.8.8.8.8.9.9.9.9.9.%
9.9.}%
\calx_tfdef s{0_11}{9}{11}{4}{9}{4.5.5.5.5.6.6.6.6.7.7.8.8.8.8.9.9.9.9.9.9.9.%
}%
\calx_tfdef s{0_10}{9}{10}{5}{9}{5.5.5.6.6.6.6.7.7.8.8.8.8.9.9.9.9.9.9.}%
\calx_tfdef s{0_9}{9}{9}{5}{9}{5.6.6.6.6.7.7.8.8.8.8.9.9.9.9.9.9.9.}%
\calx_tfdef s{0_8}{9}{8}{6}{9}{6.6.6.7.7.8.8.8.8.9.9.9.9.9.9.9.}%
\calx_tfdef s{0_7}{9}{7}{6}{9}{6.7.7.8.8.8.8.9.9.9.9.9.9.9.}%
\calx_tfdef s{0_6}{9}{6}{7}{9}{7.8.8.8.8.9.9.9.9.9.9.9.}%
\calx_tfsel s{0}{14}{13.4589.12.2660.11.1394.10.654.9.268.8.91.7.24.6.4.}%
\calx_tfdef c{0}{9}{15}{1}{9}{1.2.2.3.3.4.4.4.4.5.5.6.6.6.6.7.7.7.7.8.8.8.8.%
9.9.9.9.9.9.9.9.}%
\calx_tfdef c{0_14}{9}{14}{2}{9}{2.3.3.4.4.4.4.5.5.6.6.6.6.7.7.7.7.8.8.8.8.9.%
9.9.9.9.9.9.9.}%
\calx_tfdef c{0_13}{9}{13}{3}{9}{3.4.4.4.4.5.5.6.6.6.6.7.7.7.7.8.8.8.8.9.9.9.%
9.9.9.9.}%
\calx_tfdef c{0_12}{9}{12}{4}{9}{4.4.4.5.5.6.6.6.6.7.7.7.7.8.8.8.8.9.9.9.9.9.%
9.9.}%
\calx_tfdef c{0_11}{9}{11}{4}{9}{4.5.5.6.6.6.6.7.7.7.7.8.8.8.8.9.9.9.9.9.9.9.%
}%
\calx_tfdef c{0_10}{9}{10}{5}{9}{5.6.6.6.6.7.7.7.7.8.8.8.8.9.9.9.9.9.9.9.}%
\calx_tfdef c{0_9}{9}{9}{6}{9}{6.6.6.7.7.7.7.8.8.8.8.9.9.9.9.9.9.9.}%
\calx_tfdef c{0_8}{9}{8}{6}{9}{6.7.7.7.7.8.8.8.8.9.9.9.9.9.9.9.}%
\calx_tfdef c{0_7}{9}{7}{7}{9}{7.7.7.8.8.8.8.9.9.9.9.9.9.9.}%
\calx_tfdef c{0_6}{9}{6}{7}{9}{7.8.8.8.8.9.9.9.9.9.9.9.}%
\calx_tfsel c{0}{15}{14.5737.13.3657.12.2069.11.1065.10.491.9.197.8.65.7.17.%
6.3.}%
\or
```

P = 36.

```
\calx_tfdef s{0}{10}{15}{2}{10}{2.3.3.3.3.4.4.5.5.6.6.6.6.7.7.7.7.8.8.9.9.9.%
9.10.10.10.10.10.10.10.10.}%
\calx_tfdef c{0}{10}{16}{2}{10}{2.2.2.3.3.4.4.5.5.5.5.6.6.7.7.7.7.8.8.8.8.9.%
9.9.9.10.10.10.10.10.10.10.}%
\calx_ttdef s{0}{10}{15}{2}{10}{2.3.3.3.3.4.4.5.5.6.6.6.6.7.7.7.7.8.8.9.9.9.%
9.10.10.10.10.10.10.10.10.}%
\calx_ttdef c{0}{10}{16}{2}{10}{2.2.2.3.3.4.4.5.5.5.5.6.6.7.7.7.7.8.8.8.8.9.%
9.9.9.10.10.10.10.10.10.10.}%
```

```
\or
  P = 40.
\calx_tfdef s{0}{11}{16}{2}{11}{2.3.3.4.4.4.4.5.5.6.6.7.7.7.7.8.8.8.8.9.9.10.%
10.10.10.11.11.11.11.11.11.11.}%
\calx_tfdef c{0}{11}{17}{2}{11}{2.3.3.3.3.4.4.5.5.6.6.6.6.7.7.8.8.8.8.9.9.9.%
9.10.10.10.10.11.11.11.11.11.11.11.}%
\calx_ttdef s{0}{11}{16}{2}{11}{2.3.3.4.4.4.4.5.5.6.6.7.7.7.7.8.8.8.8.9.9.10.%
10.10.10.11.11.11.11.11.11.11.}%
\calx_ttdef c{0}{11}{17}{2}{11}{2.3.3.3.3.4.4.5.5.6.6.6.6.7.7.8.8.8.8.9.9.9.%
9.10.10.10.10.11.11.11.11.11.11.11.}%
\or
  P = 44.
\calx_ttdef s{0}{12}{18}{2}{12}{2.2.2.3.3.4.4.5.5.5.5.6.6.7.7.8.8.8.8.9.9.9.%
9.10.10.11.11.11.11.12.12.12.12.12.12.12.}%
\calx_ttdef c{0}{12}{18}{2}{12}{2.3.3.4.4.4.4.5.5.6.6.7.7.7.7.8.8.9.9.9.9.10.%
10.10.10.11.11.11.11.12.12.12.12.12.12.12.}%
\or
  P = 48.
\calx_ttdef s{0}{13}{19}{2}{13}{2.3.3.3.3.4.4.5.5.6.6.6.6.7.7.8.8.9.9.9.9.10.%
10.10.10.11.11.12.12.12.12.13.13.13.13.13.13.13.}%
\calx_ttdef c{0}{13}{19}{2}{13}{2.3.3.4.4.5.5.5.5.6.6.7.7.8.8.8.8.9.9.10.10.%
10.10.11.11.11.11.12.12.12.12.13.13.13.13.13.13.13.}%
\or
  P = 52.
\calx_ttdef s{0}{14}{20}{2}{14}{2.3.3.4.4.4.4.5.5.6.6.7.7.7.7.8.8.9.9.10.10.%
10.10.11.11.11.11.12.12.13.13.13.13.14.14.14.14.14.14.14.}%
\calx_ttdef c{0}{14}{21}{2}{14}{2.3.3.3.3.4.4.5.5.6.6.6.6.7.7.8.8.9.9.9.9.10.%
10.11.11.11.11.12.12.12.12.13.13.13.13.14.14.14.14.14.14.14.}%
\or
  P = 56.
\calx_ttdef s{0}{15}{21}{2}{15}{2.3.3.4.4.5.5.5.5.6.6.7.7.8.8.8.8.9.9.10.10.%
11.11.11.11.12.12.12.12.13.13.14.14.14.14.15.15.15.15.15.15.15.}%
\calx_ttdef c{0}{15}{22}{2}{15}{2.3.3.4.4.4.4.5.5.6.6.7.7.7.7.8.8.9.9.10.10.%
10.10.11.11.12.12.12.12.13.13.13.13.14.14.14.14.15.15.15.15.15.15.15.}%
\or
  P = 60.
\calx_ttdef s{0}{16}{22}{2}{16}{2.3.3.4.4.5.5.6.6.6.6.7.7.8.8.9.9.9.9.10.10.%
11.11.12.12.12.12.13.13.13.13.14.14.15.15.15.15.16.16.16.16.16.16.16.}%
\calx_ttdef c{0}{16}{23}{2}{16}{2.3.3.4.4.5.5.5.5.6.6.7.7.8.8.8.8.9.9.10.10.%
11.11.11.11.12.12.13.13.13.13.14.14.14.14.15.15.15.15.16.16.16.16.16.16.16.}%
\or
  P = 64.
\calx_ttdef s{0}{17}{23}{3}{17}{3.3.3.4.4.5.5.6.6.7.7.7.7.8.8.9.9.10.10.10.%
10.11.11.12.12.13.13.13.13.14.14.14.14.15.15.16.16.16.16.17.17.17.17.17.17.%
17.}%
\calx_ttdef c{0}{17}{24}{2}{17}{2.3.3.4.4.5.5.6.6.6.6.7.7.8.8.9.9.9.9.10.10.%
11.11.12.12.12.12.13.13.14.14.14.14.15.15.15.15.16.16.16.16.17.17.17.17.17.%
17.17.}%
\or
```

P = 68.

```
\calx_ttdef s{0}{18}{25}{2}{18}{2.3.3.4.4.4.4.5.5.6.6.7.7.8.8.8.8.9.9.10.10.%
11.11.11.11.12.12.13.13.14.14.14.14.15.15.15.15.16.16.17.17.17.17.18.18.18.%
18.18.18.18.}%
\calx_ttdef c{0}{18}{25}{2}{18}{2.3.3.4.4.5.5.6.6.7.7.7.7.8.8.9.9.10.10.10.%
10.11.11.12.12.13.13.13.13.14.14.15.15.15.15.16.16.16.16.17.17.17.17.18.18.%
18.18.18.18.18.}%
\or
```

P = 72.

```
\calx_ttdef s{0}{19}{26}{2}{19}{2.3.3.4.4.5.5.5.5.6.6.7.7.8.8.9.9.9.9.10.10.%
11.11.12.12.12.12.13.13.14.14.15.15.15.15.16.16.16.16.17.17.18.18.18.18.19.%
19.19.19.19.19.19.}%
\calx_ttdef c{0}{19}{26}{3}{19}{3.3.3.4.4.5.5.6.6.7.7.8.8.8.8.9.9.10.10.11.%
11.11.11.12.12.13.13.14.14.14.14.15.15.16.16.16.16.17.17.17.17.18.18.18.18.%
19.19.19.19.19.19.19.}%
\or
```

P = 76.

```
\calx_ttdef s{0}{20}{27}{2}{20}{2.3.3.4.4.5.5.6.6.6.6.7.7.8.8.9.9.10.10.10.%
10.11.11.12.12.13.13.13.13.14.14.15.15.16.16.16.16.17.17.17.17.18.18.19.19.%
19.19.20.20.20.20.20.20.20.}%
\calx_ttdef c{0}{20}{27}{3}{20}{3.4.4.4.4.5.5.6.6.7.7.8.8.9.9.9.9.10.10.11.%
11.12.12.12.12.13.13.14.14.15.15.15.15.16.16.17.17.17.17.18.18.18.18.19.19.%
19.19.20.20.20.20.20.20.20.}%
\or
```

P = 80.

```
\calx_ttdef s{0}{21}{28}{2}{21}{2.3.3.4.4.5.5.6.6.7.7.7.7.8.8.9.9.10.10.11.%
11.11.11.12.12.13.13.14.14.14.14.15.15.16.16.17.17.17.17.18.18.18.18.19.19.%
20.20.20.20.21.21.21.21.21.21.21.}%
\calx_ttdef c{0}{21}{28}{3}{21}{3.4.4.5.5.5.5.6.6.7.7.8.8.9.9.10.10.10.10.11.%
11.12.12.13.13.13.13.14.14.15.15.16.16.16.16.17.17.18.18.18.18.19.19.19.19.%
20.20.20.20.21.21.21.21.21.21.21.}%
\or
```

P = 84.

```
\calx_ttdef s{0}{22}{29}{2}{22}{2.3.3.4.4.5.5.6.6.7.7.8.8.8.8.9.9.10.10.11.%
11.12.12.12.12.13.13.14.14.15.15.15.15.16.16.17.17.18.18.18.18.19.19.19.19.%
20.20.21.21.21.21.22.22.22.22.22.22.22.}%
\calx_ttdef c{0}{22}{29}{3}{22}{3.4.4.5.5.6.6.6.6.7.7.8.8.9.9.10.10.11.11.11.%
11.12.12.13.13.14.14.14.14.15.15.16.16.17.17.17.17.18.18.19.19.19.19.20.20.%
20.20.21.21.21.21.22.22.22.22.22.22.22.}%
\or
```

P = 88.

```
\calx_ttdef s{0}{23}{30}{3}{23}{3.3.3.4.4.5.5.6.6.7.7.8.8.9.9.9.9.10.10.11.%
11.12.12.13.13.13.13.14.14.15.15.16.16.16.16.17.17.18.18.19.19.19.19.20.20.%
20.20.21.21.22.22.22.22.23.23.23.23.23.23.23.}%
\calx_ttdef c{0}{23}{30}{3}{23}{3.4.4.5.5.6.6.7.7.7.7.8.8.9.9.10.10.11.11.12.%
12.12.12.13.13.14.14.15.15.15.15.16.16.17.17.18.18.18.18.19.19.20.20.20.20.%
21.21.21.21.22.22.22.22.23.23.23.23.23.23.23.}%
\or
```

P = 92.

```
\calx_ttdef s{0}{24}{31}{3}{24}{3.4.4.4.4.5.5.6.6.7.7.8.8.9.9.10.10.10.10.11.%
```

```
11.12.12.13.13.14.14.14.14.15.15.16.16.17.17.17.17.18.18.19.19.20.20.20.20.%
21.21.21.21.22.22.23.23.23.23.24.24.24.24.24.24.24.}%
\calx_ttdef c{0}{24}{32}{2}{24}{2.3.3.4.4.5.5.6.6.7.7.8.8.8.8.9.9.10.10.11.%
11.12.12.13.13.13.13.14.14.15.15.16.16.16.16.17.17.18.18.19.19.19.19.20.20.%
21.21.21.21.22.22.22.22.23.23.23.23.24.24.24.24.24.24.24.}%
\or
```

P = 96.

```
\calx_ttdef s{0}{25}{32}{3}{25}{3.4.4.5.5.5.5.6.6.7.7.8.8.9.9.10.10.11.11.11.%
11.12.12.13.13.14.14.15.15.15.15.16.16.17.17.18.18.18.18.19.19.20.20.21.21.%
21.21.22.22.22.22.23.23.24.24.24.24.25.25.25.25.25.25.25.}%
\calx_ttdef c{0}{25}{33}{2}{25}{2.3.3.4.4.5.5.6.6.7.7.8.8.9.9.9.9.10.10.11.%
11.12.12.13.13.14.14.14.14.15.15.16.16.17.17.17.17.18.18.19.19.20.20.20.20.%
21.21.22.22.22.22.23.23.23.23.24.24.24.24.25.25.25.25.25.25.25.}%
\or
```

P = 100.

```
\calx_ttdef s{0}{26}{33}{3}{26}{3.4.4.5.5.6.6.6.6.7.7.8.8.9.9.10.10.11.11.12.%
12.12.12.13.13.14.14.15.15.16.16.16.16.17.17.18.18.19.19.19.19.20.20.21.21.%
22.22.22.22.23.23.23.23.24.24.25.25.25.25.26.26.26.26.26.26.26.}%
\calx_ttdef c{0}{26}{34}{2}{26}{2.3.3.4.4.5.5.6.6.7.7.8.8.9.9.10.10.10.10.%
11.11.12.12.13.13.14.14.15.15.15.15.16.16.17.17.18.18.18.18.19.19.20.20.21.%
21.21.21.22.22.23.23.23.23.24.24.24.24.25.25.25.25.26.26.26.26.26.26.26.}%
\fi%
\fi%
```

37. Cumulative line and macro count

module	lines	macros
xintkernel	715	(162)
xinttools	1625	(378)
xintcore	443	(153)
xint	772	(249)
xintbinhex	828	(164)
xintgcd	366	(63)
xintfrac	2807	(782)
xintseries	384	(66)
xintcfrac	1038	(257)
xintexpr	669	(153)
calxcore	2275	(607)
calxint	2521	(619)
calxfp	2884	(695)
calx	5439	(1590)
calxlog	3434	(625)
calxlogdata	1496	(37)
calxtrig	3739	(675)
calxtrigdata	761	(165)

Total number of code lines: 32196. Each package or library file starts with circa 50 lines dealing with catcodes, package identification and reloading management, also for Plain TEX.

Total number of def'ed (or let'ed) macros: 7440. This is an approximation as some macros are def'ed in a way escaping the automated detection, in particular this applies to xintexpr macros associated to infix operators and syntax elements, whose construction uses \csname-based definitions with a template and auxiliary macros. Their number has been evaluated manually (last updated in June 2022, probably outdated) at being at least about 452. This number is probably now wrong but is incorporated into the calx count shown left, and the total above.

Version 2.0a2 of 2026/11/26.