1 Examples

Package \texttt{bnumexpr} provides \texttt{\themexpr\langle\textit{expression}\rangle\relax} which is analogous to \texttt{\themexpr\langle\textit{expression}\rangle\relax}, with these extensions:

- it allows arbitrarily big integers,
- it computes powers (with either \texttt{**} or \texttt{^} as infix operator),
- it computes factorials (with \texttt{!} as postfix operator),
- it has an operator \texttt{//} for floored division and \texttt{/:} for the associated modulo,
- the space character can be used to separate in the source blocks of digits for better readability of long numbers,
- since 1.2b the underscore \_ may also be used as visual digit separator,
- comma separated expressions are allowed.

It expands completely, but in two steps (whereas \texttt{\themexpr\langle\textit{expression}\rangle\relax} expands in one step only, and \texttt{\numexpr} by itself is not expandable).

1.2d adds a \texttt{\bnumeval\{\langle\textit{expression}\rangle\}} interface. The \texttt{\bnumeval} macro also expands completely in two steps.
2 Differences from $\numexpr$

Examples:
\begin{verbatim}
\thebnumexpr 1 208 637 867 * (2 187 917 891 + 3 109 197 072)\relax
6402293730134103921
\bnumeval {(13_8089_1090-300_1890_2902)*(1083_1908_3901-109_8290_3890)}
-278751467289976289932
\thebnumexpr (92_874_927_979**5-31_9792_7979**6)/30!\relax
-400624073656543944035189
\bnumeval {30!/20!/21/22/24/25/(26*27*28*29)}
30
\thebnumexpr 13^50//12^50, 13^50/:12^50\relax
54, 650556287901099025745221048683760161794567947140168553
\bnumeval {13^50/12^50, 12^50}
55, 910043815000214977332758527534256632492715260325658624
\thebnumexpr (1^10+2^10+3^10+4^10+5^10+6^10+7^10+8^10+9^10)^3\relax
118685075462698981700620828125
\bnumeval {100!/36^100}
219
\end{verbatim}

2 Differences from $\numexpr$

Apart from the extension to big integers (i.e. exceeding the \TeX{} limit at 2147483647), and the added operators, there are a number of important differences between $\bnumexpr$ and $\numexpr$:

1. one must use either $\thebnumexpr$ or $\bneth\bnumexpr$ to get a printable result, as $\bnumexpr \ldots \relax$ expands to a private format (using $\number$ as prefix to $\bnumexpr$ would only serve to trigger the expansion to the $\bnumexpr$ private format, hence will raise a \TeX{} error),

2. one may embed directly (without $\bneth$) a $\bnumexpr \ldots \relax$ in another one (or in a $\xintexpr \ldots \relax$), but not in a $\numexpr \ldots \relax$; on the other hand a $\numexpr \ldots \relax$ does not need to be prefixed by $\the$ or $\number$ inside $\bneth \bnumexpr$ or $\thebnumexpr$,

3. contrarily to $\numexpr$, the $\bnumexpr$ parser stops only after having found (and swallowed) a mandatory ending $\relax$ token,
4. in particular spaces between digits do not stop \texttt{\numexpr}, in contrast with \texttt{\bnumexpr}:
\begin{verbatim}
the \numexpr 3 5+79\relax expands (in one step) to 35+79\relax
thebnumexpr 3 5+79\relax expands (in two steps) to 114
\end{verbatim}

5. one may do \texttt{\edef \tmp \{\bnumexpr 1+2\relax \}}, and then either use \texttt{\tmp} in another \texttt{\bnumexpr ...\relax}, or print it via \texttt{\bnethe \tmp}. The computation is done at the time of the \texttt{\edef} (and two expansion steps suffice). This is again in contrast with \texttt{\numexpr ...\relax} which, without \texttt{the} (or \texttt{number} or \texttt{romannumeral}) as prefix would not expand inside an \texttt{edef}.

6. expressions may be comma separated. On input, spaces are ignored, naturally, and on output the values are comma separated with a space after each comma,

7. \texttt{\bnumexpr -(1+1)\relax} is legal contrarily to \texttt{\numexpr -(1+1)\relax} which raises an error,

8. \texttt{\numexpr 2\cnta \relax} is illegal (with \texttt{\cnta} a \texttt{count}-variable.) But \texttt{\bnumexpr 2\cnta \relax} is perfectly legal and will do the tacit multiplication,

9. more generally, tacit multiplication applies in front of parenthesized sub-expressions, or sub \texttt{\numexpr ...\relax} (or \texttt{\numexpr ...\relax}),

10. the underscore \_ is accepted within the digits composing a number and is silently ignored by \texttt{\bnumexpr},

11. \texttt{\numexpr} accepts \texttt{\TeX} syntax for hexadecimal input "\texttt{EF}" (or octal input '377), but currently \texttt{\bnumexpr} does not. This could possibly be added in future\footnote{It is only a matter of copying pasting relevant code from \texttt{xintexpr}, so depends upon user demands addressed to the author ;-)}. (hexadecimal prefix " is part of the recognized \texttt{xintiiexpr} or syntax from \texttt{xintexpr} if \texttt{xintbinhex} is loaded.) In the meantime, one may encapsulate such inputs (obeying \TeX's bound on numbers) into a sub \texttt{\numexpr ...\relax}.

An important thing to keep in mind is that if one has a calculation whose result is a small integer, acceptable by \TeX in \texttt{ifnum} or count assignments, this integer produced by \texttt{\bnumexpr} is not self-delimiting, contrarily to a \texttt{\numexpr ...\relax} construct: the situation is exactly as with a \texttt{\the \numexpr ...\relax}, thus one may need to terminate the number to avoid premature expansion of following tokens; for example with the \texttt{space} token.

The parser \texttt{\bnumexpr} is a scaled-down version of parser \texttt{\xintiiexpr} from package \texttt{xintexpr}. It lacks in particular boolean operators, square roots and other functions, variables, hexadecimal inputs, etc... it may be slightly
3 Printing big numbers

...faster when handling complicated expressions as it does not have to check so many things.

The documentation of \texttt{xintexpr} explains that there is an impact on the memory of \TeX{} (the string pool, the hash table) as each intermediate number is stored as a dummy control sequence name during processing. After thousands of evaluations with numbers having hundreds of digits, parts of the \TeX{} memory can become saturated and end the \texttt{latex|pdflatex} run, but the problem can be avoided via enlarged memory parameters for \texttt{pdftex}, as made possible by modern \TeX{} installations. Anyhow, computations with thousands of digits take time, and this is probably a more stringent constraint.

If the same operations need to be repeated again and again tens of thousands of times on varying (big) numbers, the memory problem mentioned above may be avoided by using nested macros rather than \texttt{\bnumexpr} or \texttt{xintexpr} expressions. Utility \texttt{\xintNewIIExpr} from package \texttt{xintexpr} can be used to construct the possibly very complicated nested macro from a given expression with the needed operators and usual \texttt{#1, #2, #3, ...} placeholders.

The \texttt{\epsilon-\TeX} extensions are required (this is the default on all modern installations for \texttt{latex|pdflatex} and also for \texttt{xelatex|lualatex}).

3 Printing big numbers

\LaTeX{} will not split long numbers at the end of lines. I personally often use helper macros (not in the package) of the following type:

\begin{verbatim}
\def\allowsplits #1{\ifx #1\relax \else #1\hskip 0pt plus 1pt\relax \expandafter\allowsplits\fi}%
\def\printnumber #1{\expandafter\allowsplits \romannumeral-`0#1\relax }% %\printnumber thus first ``fully'' expands its argument.
\end{verbatim}

\begin{verbatim}
\thebnumexpr 1000! =  402387260077093773543702433923003985719374864
21071463254379991042993851239862902059205442028486969404800479988610197196
0586316668729940855890113238296699445490997424504087073759918283627727188
732519797505905527612087497546249704360614182780946446496291056393887437
8864873377119181045825783647849977012476632889835955735432513185323958463
07555740911426241747439437553428646576611667797396668820291207379143853
719588249808126867838374559731746136085379534524221586593201928090872927
308431392844403281231558611036976801357304216168747609675871348312025478
589320767169132448426236131412508780208000261683151027314827977074874635
868170164365024153691398281264810213092761244896359928705114964975419909
34222156683255208201233138611681553561583654698404670897560290950537616
47584772842188967946244945160765353408198901385442487984959953319101723
35556602139453997362807501378376153071277619268490345325625000015888535
14733117012053618759215109077880193931780141945452572238655441461062892
187960223838971476085062768629671466746975629112340824939208160153780889
8939645182632432637161676217916890977991190375403127462228998800519544414
2820121873617459926249565817466280295557029902432451318167210465832036
7869061172601587835207515162842255402651704833042262143974286933061690897
\end{verbatim}
4 Expression syntax

It is the expected one with infix operators and parentheses, the recognized operators being +, -, *, / (rounded division), \(^\wedge\) (power), \(^\star\star\) (power), // (by default floored division), \(/:\) (the associated modulo) and ! (factorial).\(^2\)

The modulo \(/:\) is by default associated with the floored division //, but using \texttt{\textbackslash bnumpsetup \{mod=...\}} it can, like the other operators, be remapped to any macro of one's choice.

Different computations may be separated by commas. The whole expression is handled token by token, any component (digit, operator, parenthesis... even the ending \texttt{\textbackslash relax}) may arise on the spot from macro expansions.

The precedence rules are the expected ones.

There is currently no user interface to change precedence levels. The three operators /, //, \(/:\) are at the same level of precedence as the multiplication \(*\). The factorial postfix \texttt{!} has highest precedence. The minus signs inherit the precedence level of the previously encountered infix operators.

In case of equal precedence the operations are left-associative, hence:

\texttt{\textbackslash bnumeval \{2^3^4, (2^3)^4, 2^(3^4)\}}

4096, 4096, 2417851639229258349412352

The underscore \_\_\_ can be used to separate digits in long numbers, for readability of the input.

\(^2\)Releases 1.2b and earlier associated in the default configuration truncated division to //\textbackslash /\textbackslash. This was changed at 1.2c to stay in sync with \texttt{xintcore 1.2p}. For backwards compatibility, one may add to existing document \texttt{\textbackslash bnumpsetup\{div=\texttt{xintiiDivTrunc}, mod=\texttt{xintiiModTrunc}\}.}
5 Options

The sole package option is custom: it tells bnumexpr not to load package xint-core.

6 \bnumexprsetup

Package bnumexpr needs that some big integer engine provides the macros doing the actual computations. By default, it loads package xintcore (a subset of xint; version 1.3d is required) and uses \bnumexprsetup in the following way:

\usepackage{xintcore}
\bnumexprsetup{add=\xintiiAdd, sub=\xintiiSub, mul=\xintiiMul, 
divround=\xintiiDivRound, div=\xintiiDivFloor, 
mod=\xintiiMod, pow=\xintiiPow, fac=\xintiiFac}

If using \bnumexprsetup, it is not necessary to specify all keys, for example one can do \bnumexprsetup {mul=\MyFasterMul }, and only multiplication will be changed.

Naturally it is up to the user to load the appropriate package for the alternative macros.

As per the macros which are the key values, they must have the following properties:

1. they must be completely expandable (in the sense of an \edef or a \csname me ...\endcsname.)

2. they must fully expand their arguments first (in the sense of \romannumeral -`0.)

3. they must output a number with no leading zeros, at most one minus sign and no plus sign.

The first two items are truly mandatory, the last one may be not obeyed if the extra key opp is used with \bnumexprsetup to specify a suitable macro for the opposite of a number. This macro will be presented not with a braced argument but directly with a sequence of digits (either as gathered by the parser which skips leading zeros, or as produced by the other arithmetic macros and then there could be a minus, or even a plus if macros others than the ones from xintcore have been used). Thus, opp could identify a plus sign + upfront and then act adequately. ³

Macro \bnumexprsetup can be used multiple times in the same document, thus allowing to switch math engines or to remap operators to some other arithmetic macros of the same math engine. Its effect obeys the local scope.

³see \BNE_Op_opp in the code for the default.
7 Readme

7 Readme

| Source:  bnumexpr.dtx
| Version: v1.2e, 2019/01/08 (doc: 2019/01/08)
| Author:  Jean-Francois Burnol
| Info:    Expressions with big integers
| License: LPPL 1.3c

README: [Usage], [Installation], [License]
===========================================================================

Usage
-----

The package `bnumexpr` allows _expandable_ computations with big integers and the four infix operators `+`, `-`, `*`, `/` familiar from the `\numexpr` e-TeX parser.

Besides extending the scope to arbitrarily big numbers (and having a more complete syntax, for example `-(1)` is legal input), it adds the (by default) floored division operator `/\`, and its associated modulo `/:\`, the power operator `^` (or equivalently `**`), and the factorial post-fix operator `!`. The space character as well as the underscore character `_` both may serve to optionally separate digits in long numbers, for better readability of the input.

For example:

```
\bnumeval{( 92_874_927_979^5 - 31_9792_7979^6 ) / 30!}
```

The above expands (in two steps) to `-4006240736596543944035189` (the `/` does rounded division to match the `\numexpr` behaviour).

The expression parser is scaled-down from the `\xinttheiiexpr...\relax` parser as provided by package xintexpr[^1]: it does not handle hexadecimal input, boolean operators, dummy or user defined variables, functions, etc...

By default, the package loads xintcore[^1] (release 1.3d is then required) but it is possible via option _custom_ and macro `\bnumexprsetup` to map the operators to macros of one's own choice. It is the responsability of the user to load the packages providing these custom macros.

Notice that the possibility not to use the xintcore macros might be removed in the future: perhaps a future release will maintain during computations a private internal representation (especially tailored either for the xintcore macros or new ones which would be included within `bnumexpr.sty` itself) and the constraints this implies may render optional use of other macros impossible.

[^1]: <http://www.ctan.org/pkg/xint>

Installation
------------

Obtain `bnumexpr.dtx` (and possibly, `bnumexpr.ins` and the `README`) from CTAN:

> <http://www.ctan.org/pkg/bnumexpr>

Both `"tex bnumexpr.ins"` and `"tex bnumexpr.dtx"` extract from

7
`bnumexpr.dtx` the following files:

`bnumexpr.sty`
: this is the style file.

`README.md`
: reconstitutes this README.

`bnumexprchanges.tex`
: lists changes from the initial version.

`bnumexpr.tex`
: can be used to generate the documentation:
  - with latex+dvipdfmx: `"latex bnumexpr.tex"` (thrice) then
    `"dvipdfmx bnumexpr.dvi"`.
  - with pdflatex: `"pdflatex bnumexpr.tex"` (thrice).
: Ignore dvipdfmx warnings, but if the pdf file has problems with
  fonts (possibly from an old dvipdfmx), use then rather pdflatex.
: In both cases files `README.md` and `bnumexprchanges.tex` must
  be present in the same repertory.

without `bnumexpr.tex`:
  : `"pdflatex bnumexpr.dtx"` (thrice) extracts all files and
    simultaneously generates the pdf documentation.

Finishing the installation:

```
 bnumexpr.sty      --> TDS:tex/latex/bnumexpr/
bnumexpr.dtx      --> TDS:source/latex/bnumexpr/
bnumexpr.ins      --> TDS:source/latex/bnumexpr/
bnumexpr.pdf      --> TDS:doc/latex/bnumexpr/
 README            --> TDS:doc/latex/bnumexpr/
```

Files `bnumexpr.tex`, `bnumexprchanges.tex`, `README.md` may be
discarded.

License
-------

Copyright (C) 2014-2019 by Jean-Francois Burnol

> This Work may be distributed and/or modified under the
> conditions of the LaTeX Project Public License 1.3c.
> This version of this license is in
> <http://www.latex-project.org/lppl/lppl-1-3c.txt>
> and version 1.3 or later is part of all distributions of
> LaTeX version 2005/12/01 or later.

This Work has the LPPL maintenance status "author-maintained".

The Author and Maintainer of this Work is Jean-Francois Burnol.

This Work consists of the main source file `bnumexpr.dtx`
and the derived files
8 Changes

bnumexpr.sty, bnumexpr.pdf, bnumexpr.ins, bnumexpr.tex,
bnumexprchanges.tex, README.md

8 Changes

1.2e (2019/01/08) Fixes a documentation glitch (extra braces when mention-
ing \the \numexpr or \thebnumexpr).

1.2d (2019/01/07) • requires xintcore 1.3d or later (if not using option custom).
  • adds \bnumeval\{\langle expression\rangle\} user interface.

1.2c (2017/12/05) Breaking changes:
  • requires xintcore 1.2p or later (if not using option custom).
  • divtrunc key of \bnumexprsetup is renamed to div.
  • the // and \:/ operators are now by default associated to the floored division. This is to keep in sync with the change of xintcore at 1.2p.
  • for backwards compatibility, one may add to existing document: \bnumexprsetup{div=\xintiiDivTrunc, mod=\xintiiModTrunc}

1.2b (2017/07/09) • the _ may be used to separate visually blocks of dig-
its in long numbers.

1.2a (2015/10/14) • requires xintcore 1.2 or later (if not using option custom).
  • additions to the syntax: factorial !, truncated division //, its associated modulo \:/ and ** as alternative to ^.
  • all options removed except custom.
  • new command \bnumexprsetup which replaces the commands such as \bnex
  umexprusesbigintcalc.
  • the parser is no more limited to numbers with at most 5000 digits.

1.1b (2014/10/28) • README converted to markdown/pandoc syntax,
  • the package now loads only xintcore, which belongs to xint bundle version 1.1 and extracts from the earlier xint package the core arithmetic operations as used by bnumexpr.

1.1a (2014/09/22) • added l3bigint option to use experimental \LaTeX3 pack-
age of the same name,
  • added Changes and Readme sections to the documentation,
  • better \BNE_protect mechanism for use of \bnumexpr ...\relax in-
side an \edef (without \bnethe). Previous one, inherited from xinte\xpr.sty 1.09n, assumed that the \=\langle digits\rangle dummy control sequence
8 Changes

encapsulating the computation result had \relax meaning. But removing this assumption was only a matter of letting \BNE_protect protect two, not one, tokens. This will be backported to next version of xintexpr, naturally (done with xintexpr.sty 1.1).

1.1 (2014/09/21) First release. This is down-scaled from the (development version of) xintexpr. Motivation came the previous day from a chat with Joseph Wright over big int status in \LaTeX. The \bnumexpr ...\relax parser can be used on top of big int macros of one's choice. Functionalities limited to the basic operations. I leave the power operator ^ as an option.
9 Package \texttt{bnumexpr} implementation

Contents

9.1 Package identification and catcode setup ........................................... 11
9.2 Some helper macros and constants from xint ..................................... 11
9.3 \texttt{\textbackslash bnumexprsetup} ..................................................... 12
9.4 Package options .................................................................................. 12
9.5 \texttt{\textbackslash bnumexpr}, \texttt{\textbackslash thebnumexpr}, \texttt{\textbackslash bnethe}, \texttt{\textbackslash bnumeval} .................................................... 13
9.6 \texttt{\textbackslash BNE\_getnext} ............................................................... 13
9.7 Parsing an integer ............................................................................... 14
9.8 \texttt{\textbackslash BNE\_getop} ................................................................. 16
9.9 Until macros for global expression and parenthesized sub-ones .......... 17
9.10 The arithmetic operators. .................................................................. 17
9.11 \texttt{!} as postfix factorial operator ................................................. 18
9.12 The minus as prefix operator of variable precedence level ........... 19
9.13 The comma may separate expressions. ............................................ 19
9.14 Cleanup .............................................................................................. 20

Comments are sparse. Error handling by the parser is kept to a minimum; if something goes wrong, the offensive token gets discarded, and some undefined control sequence attempts to trigger writing to the log of some sort of informative message. It is recommended to set \texttt{\errorcontextlines} to at least \texttt{2} for more meaningful context.

9.1 Package identification and catcode setup

\texttt{v1.2c} forgot to identify itself as such :(. Fixed (of course) at \texttt{v1.2d}.

\begin{verbatim}
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{bnumexpr}[2019/01/08 v1.2e Expressions with big integers (JFB)]
\edef\BNErestorecatcodes {
\catcode\noexpand\!\the\catcode\!
\catcode\noexpand\?\the\catcode\?
\catcode\noexpand\_\the\catcode\_
\catcode\noexpand\:\the\catcode\:\relax }% 
\catcode\! 11
\catcode\? 11
\catcode\_ 11
\catcode\:\ 12
\catcode\* 12
\catcode\( 12
\catcode\) 12
\end{verbatim}

9.2 Some helper macros and constants from xint

These macros from xint should not change, hence overwriting them here should not be cause for alarm. I opted against renaming everything with \texttt{\textbackslash BNE\_} prefix rather than \texttt{\xi}
The \xint_dothis/\xint_orthat thing is a new style I have adopted for expandably forking. The least probable branches should be specified first, for better efficiency. See examples of uses in the present code.

18 \chardef\xint_c_ 0
19 \chardef\xint_c_i 1
20 \chardef\xint_c_ii 2
21 \chardef\xint_c_vi 6
22 \chardef\xint_c_vii 7
23 \chardef\xint_c_viii 8
24 \chardef\xint_c_ix 9
25 \chardef\xint_c_x 10
26 \long\def\xint_gobble_i #1{}%
27 \long\def\xint_gobble_iii #1#2#3{}%
28 \long\def\xint_firstofone #1{#1}%
29 \long\def\xint_firstoftwo #1#2{#1}%
30 \long\def\xint_secondoftwo #1#2{#2}%
31 \long\def\xint_firstofthree #1#2#3{#1}%
32 \long\def\xint_secondofthree #1#2#3{#2}%
33 \long\def\xint_thirdofthree #1#2#3{#3}%
34 \def\xint_gob_til_! #1!{}% this ! has catcode 11
35 \long\def\xint_UDsignfork #1-#2#3\krof {#2}%
36 \long\def\xint_afterfi #1#2\fi {i #1}%
37 \let\xint_orthat\xint_firstofone
38 \long\def\xint_zapspaces #1 #2{#1#2\xint_zapspaces }%
39 \long\def\xint_zapspaces #1 #2{#1#2\xint_zapspaces }%

9.3 \bnumexprsetup

New with v1.2a. Replaces removed \bnumexprUsesbigintcalc etc...
40 \catcode`! 3
41 \def\bnumexprsetup #1{%\BNE_parsekeys #1,=!,}%
42 \def\BNE_parsekeys #1=#2#3,{\ifx!#2\expandafter\BNE_parsedone\fi
43 \expandafter
44 \let\csname BNE_Op_\xint_zapspaces #1 \xint_gobble_i\endcsname=#2\BNE_parsekeys
45 }%
46 \catcode`! 11
47 \def\BNE_parsedone #1\BNE_parsekeys {}%

9.4 Package options

v1.2c replaces former key \divtrunc by \div. v1.2d requires (by default) xintcore 1.3d.
48 \def\BNE_tmpa {0}%
49 \DeclareOption {custom}{{\def\BNE_tmpa {1}}%}
50 \ProcessOptions\relax
51 \if0\BNE_tmpa % Default is to load xintcore.sty
52 \RequirePackage[xintcore][2019/01/06]\xintcore 1.3d
53 \bnumexprsetup\add=new\xintiiAdd, sub=new\xintiiSub, mul=new\xintiiMul,
54 \divround=new\xintiiDivRound, div=new\xintiiDivFloor,
55 \mod=new\xintiiMod, pow=new\xintiiPow, fac=new\xintiiFac}%
56 \fi
9.5 \bnumexpr, \thebnumexpr, \bnethe, \bnumeval

In the full \xintexpr, the final unlocking may involve post-treatment of the comma separated values, hence there are _print macros to handle the possibly comma separated values. Here we may just identify _print with _unlock.

With v1.2a the gathering of numbers happens directly inside \csname ...\endcsname. There is no more a `locking' macro.

Attention v1.2d gives new meaning to \bnumeval, the former meaning is now named \bnumexpr. The leading space in \BNE_wrap is removed at it served to nothing.

72 \def\BNE_getnext #1% 73 { 74 \expandafter\BNE_getnext_a\romannumeral-`0#1% 75 }% 76 \def\BNE_getnext_a #1% 77 { 78 \xint_gob_til_! #1\BNE_gn_foundexpr !% this ! has catcode 11 79 \ifcat\relax#1% \count or \numexpr etc... token or count, dimen, skip cs 80 \expandafter\BNE_gn_countetc 81 \else 82 \expandafter\expandafter\expandafter\BNE_gn_fork\expandafter\string 83 \fi

9.6 \BNE_getnext

The getnext scans forward to find a number: after expansion of what comes next, an opening parenthesis signals a parenthesized sub-expression, a ! with catcode 11 signals there was there a sub \bnumexpr ...\relax (now evaluated), a minus sign is treated as a prefix operator inheriting its precedence level from the previous operator, a plus sign is swallowed, a \count or \dimen will get fetched to \number (in case of a count variable, this provides a full locked number but \count 0 1 for example is like 1231 if \count 0’s value is 123); a digit triggers the number scanner. With v1.2a the gathering of digits happens directly inside \csname ...\endcsname. Leading zeroes are trimmed directly. The flow then proceeds with \BNE_getop which looks for the next operator or possibly the end of the expression. Note: \bnumexpr \relax is illegal.

Extended in v1.2a to recognize \ht, etc...
This is quite simplified here compared to \xintexpr, for various reasons: we have dropped the \xintNewExpr thing, and we can treat the ( directly as we don’t have to get back to check if we are in an \xintexpr, \xintfloatexpr, etc..

9.7 Parsing an integer

We gather a string of digits, plus and minus prefixes have already been swallowed. There might be some leading string of zeros which will have to be removed. In the full \xintexpr the situation is more involved as it has to recognize and accept decimal numbers, numbers in scientific notation, also hexadecimal numbers, function names, variable names...
Contents

9.8 \BNE_getop

This finds the next infix operator or closing parenthesis or expression end. It then leaves in the token flow <precedence> <operator> <locked number>. The <precedence> stops expansion and ultimately gives back control to a \BNE_until_<op> command. The code here is derived from more involved context where the actual macro associated to the operator may vary, depending if we are in \xintexpr, \xintfloatexpr or \xintiiexpr. Here things are simpler but I have kept the general scheme, thus the actual macro to be used for the <operator> is not decided immediately.

\bne: \texttt{v1.2a} adds a technique for allowing two-letters operators, for //, /: and **.

\begin{verbatim}
167 \def\BNE_getop #1#2% this #1 is the current locked computed value
168 \{% 169 \expandafter\BNE_getop_a\expandafter #1\romannumeral-`0#2% 170 \}\%
171 \catcode`* 11
172 \def\BNE_getop_a #1#2% 173 {% if a control sequence is found, must be \relax, or possibly register or
174 % variable if tacit multiplication is allowed 175 \ifx \relax #2\xint_dothis\xint_firstofthree\fi 176 \% tacit multiplications:
177 \ifcat \relax #2\xint_dothis\xint_secondofthree\fi 178 \if (#2\xint_dothis \xint_secondofthree\fi 179 \ifx !#2\xint_dothis \xint_secondofthree\fi 180 \xint_orthat \xint_thirdofthree
181 {\BNE_foundend #1}%
182 {\BNE_precedence_* *#1#2}% tacit multiplication
183 {\BNE_scanop_a #2#1}%
184 \}%
185 \catcode`* 12
186 \def\BNE_foundend \{\xint_c_ \relax \}% \relax is only a place-holder here.
187 \def\BNE_scanop_a #1#2#3% 188 \{%expandafter\BNE_scanop_b\expandafter #1\expandafter #2\romannumeral-`0#3%
189 \def\BNE_scanop_b #1#2#3% 190 \{% 191 \ifcat#3\relax\xint_dothis{\BNE_foundop #1#2#3}\fi 192 \ifcsname BNE_itself_#1#3\endcsname\endsname 193 \xint_dothis
194 \{%expandafter\BNE_foundop\csname BNE_itself_#1#3\endsname #2\}\fi
195 \xint_orthat \{\BNE_foundend #1#2#3\}%
196 \}%
197 \def\BNE_foundend \% 198 \{% 199 \ifcsname BNE_precedence_#1\endsname 200 \\csname BNE_precedence_#1\expandafter\endsname 201 \expandafter #1% 202 \else
203 \BNE_notanoperator \{#1\}endafter\BNE_getop 204 \fi
205 \}%
206 \def\BNE_notanoperator \{#1\}endafter\BNE_notanoperator \{#1\}$%
\end{verbatim}
9.9 Until macros for global expression and parenthesized sub-ones

The minus sign as prefix is treated here.

\begin{verbatim}
\catcode`\=11
\def\BNE_tmpa #1{%nn
  \def\BNE_until_end_a ##1{%nn
    \xint_UDsignfork
    ##1{\expandafter\BNE_until_end_a\romannumeral-`0#1}%nn
    -{\BNE_until_end_b ##1}%
    \krof
  }%
}expandafter\BNE_tmpa\csname BNE_op_-vi\endcsname
\def\BNE_until_end_b #1#2{%nn
  \ifcase #1\expandafter\BNE_done
  \or
  \xint_afterfi{\BNE:extra_)_?\expandafter
      \BNE_until_end_a\romannumeral-`0\BNE_getop }%
  \else
  \xint_afterfi{\expandafter\BNE_until_end_a
      \romannumeral-`0\csname BNE_op_#2\endcsname }%
  \fi
}%
\catcode`) 11
\def\BNE_op_( {\expandafter\BNE_until_)_a\romannumeral-`0\BNE_getnext }%
\let\BNE_oparen\BNE_op_(
\catcode`) 12
\def\BNE_tmpa #1{%nn
  \def\BNE_until_)_a ##1{\xint_UDsignfork
    ##1{\expandafter \BNE_until_)_a\romannumeral-`0#1}%nn
    -{\BNE_until_)_b ##1}%
  \krof }%
}expandafter\BNE_tmpa\csname BNE_op_-vi\endcsname
\def \BNE_until_)_b #1#2{%nn
  \ifcase #1\expandafter \BNE_missing_)_? % missing ) ?
  \or\expandafter \BNE_getop % found closing
  \else \xint_afterfi
  {\expandafter \BNE_until_)_a\romannumeral-`0\csname BNE_op_#2\endcsname }%
  \fi
}%
\catcode`) 11
\def\BNE_op_( {\expandafter\BNE_until_)_a\romannumeral-`0\BNE_getnext }%
\let\BNE_oparen\BNE_op_(
\catcode`) 12
\def\BNE_tmpa #1{%nn
  \def\BNE_until_)_a ##1{\xint_UDsignfork
    ##1{\expandafter \BNE_until_)_a\romannumeral-`0#1}%nn
    -{\BNE_until_)_b ##1}%
  \krof }%
}expandafter\BNE_tmpa\csname BNE_op_-vi\endcsname
\def \BNE_until_)_b #1#2{%nn
  \ifcase #1\expandafter \BNE_missing_)_? % missing ) ?
  \or\expandafter \BNE_getop % found closing
  \else \xint_afterfi
  {\expandafter \BNE_until_)_a\romannumeral-`0\csname BNE_op_#2\endcsname }%
  \fi
}%
\catcode`) 11
\def\BNE_op_( {\expandafter\BNE_until_)_a\romannumeral-`0\BNE_getnext }%
\let\BNE_oparen\BNE_op_(
\catcode`) 12
\end{verbatim}

9.10 The arithmetic operators.

This is where the infix operators are mapped to actual macros. These macros must `\texttt{\textasciitilde f-}
expand' their arguments, and know how to handle then big integers having no leading zeros and at most a minus sign.
v1.2a adds // for truncated division, /: for modulo operations and ** for powers (synonym to ^).

v1.2c has // and /: per default associated to floored division.

\begin{verbatim}
\def\BNE_tmpc #1#2#3#4#5#6#7% 
{\def #1##1% \BNE_op_<op>
{\expandafter #2\expandafter ##1\romannumeral-`0\expandafter\BNE_getnext}
\def #2##1##2% \BNE_until_<op>_a
{\xint_UDsignfork##2{\expandafter #2\expandafter ##1\romannumeral-`0##4}-{#3##1##2}\krof}
\def #3##1##2##3##4% \BNE_until_<op>_b
{\either execute next operation now, or first do next (possibly unary)
\ifnum##2>##5\xint_afterfi{\expandafter #2\expandafter ##1\romannumeral-`0\csname BNE_op_##3\endcsname{##4}}\else\xint_afterfi{\expandafter##2\expandafter##3\csname .=#6{\BNE_unlock##1}{\BNE_unlock##4}\endcsname}\fi}
\let #7##5}
\def\BNE_tmpb #1#2#3% 
{\expandafter\BNE_tmpc\csname BNE_op_#1\expandafter\endcsname\csname BNE_until_#1_a\expandafter\endcsname\csname BNE_until_#1_b\expandafter\endcsname\csname BNE_op_-#2\expandafter\endcsname\csname xint_c_#2\expandafter\endcsname\csname #3\expandafter\endcsname\csname BNE_precedence_#1\endcsname}
\BNE_tmpb +{vi}{BNE_Op_add}
\BNE_tmpb -{vi}{BNE_Op_sub}
\BNE_tmpb *{vii}{BNE_Op_mul}
\BNE_tmpb /{vii}{BNE_Op_divround}
\BNE_tmpb ^{viii}{BNE_Op_pow}
\expandafter\def\csname BNE_itself_**\endcsname{^}
\expandafter\def\csname BNE_itself_//\endcsname{/}
\expandafter\def\csname BNE_itself_/:\endcsname{:}
\BNE_tmpb {//}{vii}{BNE_Op_div}
\BNE_tmpb {/:}{vii}{BNE_Op_mod}
\end{verbatim}

9.11 ! as postfix factorial operator

New with v1.2a.

\let\BNE_precedence_! \xint_c_x
\def\BNE_op_! #1% 
{\expandafter\BNE_getop\csname .=\BNE_Op_fac\{\BNE_unlock #1\}\endcsname}
\end{verbatim}

9.11 ! as postfix factorial operator

New with v1.2a.

\let\BNE_precedence_! \xint_c_x
\def\BNE_op_! #1% 
{\expandafter\BNE_getop\csname .=\BNE_Op_fac\{\BNE_unlock #1\}\endcsname}
\end{verbatim}

9.11 ! as postfix factorial operator

New with v1.2a.

\let\BNE_precedence_! \xint_c_x
\def\BNE_op_! #1% 
{\expandafter\BNE_getop\csname .=\BNE_Op_fac\{\BNE_unlock #1\}\endcsname}
\end{verbatim}

9.11 ! as postfix factorial operator

New with v1.2a.
9.12 The minus as prefix operator of variable precedence level

It inherits the level of precedence of the previous operator.

9.13 The comma may separate expressions.

It suffices to treat the comma as a binary operator of precedence \textit{ii}. We insert a space after the comma. The current code in \texttt{\xintexpr} does not do it at this stage, but only later during the final unlocking, as there is anyhow need for some processing for final formatting and was considered to be as well the opportunity to insert the space. Here, let’s do it immediately. These spaces are not an issue when \texttt{\bnumexpr} is identified as a sub-expression in \texttt{\xintexpr}, for example in: \texttt{\xinttheiiexpr \lcm(\bnumexpr 175-12,1\relax 23+34,56\relax )}\relax (this example requires package \texttt{xintgcd}).
9.14 Cleanup

\let\BNE_tmpa\relax \let\BNE_tmpb\relax \let\BNE_tmpc\relax
\BNErestorecatcodes