The bnumexpr package

JEAN-FRANÇOIS BURNOL
jfbu (at) free (dot) fr

Package version: 1.2a (2015/10/14); documentation date: 2015/10/14.
From source file bnumexpr.dtx. Time-stamp: <14-10-2015 at 16:24:00 CEST>.

Contents

1 Examples

Package bnumexpr provides \thebnumexpr ...\relax which is analogous to \th⤶
e \numexpr ...\relax, while allowing arbitrarily big integers, powers, factorials, truncated division, modulo, and comma separated expressions. Examples:

```
\thebnumexpr 1208637867168*(2187917891279+3109197072870)\relax
```

6402293732412744144160032

```
\thebnumexpr (1380891090-30018902902)*(108319083901-10982903890)\relax
```

-2787514672889976289932

```
\thebnumexpr 30!/20!/21/22/23/24/25/(26*27*28*29)\relax
```

30

```
\thebnumexpr 13^50//12^50, 13^50/:12^50\relax
```

54, 6505562879010990257452210486837601617945679471401685553

```
\thebnumexpr 13^50/12^50, 12^50\relax
```

55, 910043815000214977332758527534256632492715260325658624

```
\thebnumexpr (1^10+2^10+3^10+4^10+5^10+6^10+7^10+8^10+9^10)^3\relax
```

11868507546269898170062082812 5

```
\thebnumexpr 100!/36^100\relax
```

219

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 `\bnethe \bnumexpr` to get a printable result, as `\bnumexpr ...\relax` expands to a private format,
2. one may embed directly (without `\bnethe`) a `\bnumexpr ...\relax` in another one (or in a `\xintexpr ...\relax`), but not in a `\numexpr ...\relax`; on the other hand a `\numexpr ...\relax` does not need to be prefixed by `\the` or `\number` inside `\bnethe \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 `\bnumexpr`, in contrast with `\numexpr`:

   `\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`
5. one may do `\edef \tmp {\bnumexpr 1+2\relax }`, and then either use `\tmp` in another `\bnumexpr ...\relax`, or print it via `\bnethe \tmp`. The computation is done at the time of the `\edef` (and two expansion steps suffice). This is again in contrast with `\numexpr ...\relax` which, without `\the` (or `\number` or `\romannumeral`) as prefix would not expand inside an `\edef`,
6. tacit multiplication applies in front of parenthesized sub-expressions, or sub `\bnumexpr ...\relax` (or `\numexpr ...\relax`), or in front of a `\count` or `\dimen` register.
7. 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.
8. `\bnumexpr -(1+1)\relax` is legal contrarily to `\numexpr -(1+1)\relax` which raises an error.

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

The parser \bnumexpr is a scaled-down version of parser \xintiiexpr from package xintexpr. It lacks in particular boolean operators, square roots and other functions, variables, hexadecimal inputs, etc... it may be slightly faster when handling complicated expressions as it does not have to check so many things.

I recall from the documentation of xintexpr that there is a potential impact on the memory of TeX (the string pool, the hash table) because 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 will become saturated and end the latex|pdflatex run (but the problem can in theory be avoided through the use of a ``bigger'' pdfetex compiled with enlarged memory parameters). Anyhow, computations with thousands of digits take time, and this is probably a more stringent constraint.

If the same expression needs to be evaluated again and again tens of thousands of times, it may be necessary to drop use of bnumexpr and either use directly the macros from package xintcore, or apply \xintNewIIExpr from package xintexpr to first construct the possibly very complicated nested macro.

The ε-TeX extensions are required (this is the default on all modern installations for latex|pdflatex and also for 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:

```
\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.
```

\thebnumexpr 1000!\relax = 402387260077093773543702433923003985719374864
210714632543799910429938512398629020592044208486969404800479988610197196
058631666872994808558901323829669944590997424504087073759918823627727188
732519779505950995276120874975462497043601418278094646496291056393887437
886487337119181045825783647849977012476632889835955735432513185323958463
075557409114262417474349347553428646576611667797396668820291207379143853
719588249808126867838374559731746136085379534524221586593201928090878297
308431392844403281231558611036976801357304216168747609675871348312025478
589320767169132448426236131412508780208000261683151027341827977704784635
868170164365024153691398281264810213092761244896359928705114964975419909
342221566832572080821333186116811553615836546984046708975602900950537616

```
475847728421889679646244945160765353408198901385442487984959953319101723
355556602139450399736280750137837615307127761926849034352625200015888535
147331611702103968175921510907788019393178114194545257223865541461062892
187960223838971476088506276862967146674697562911234082439208160153780889
893964518263243671616762179168909779911903754031274622289988005195444414
282012187361745992642956581746628302955570299024324153181617210465832036
786906117260158783520751516284225540265170483304226143974286933061690897
968482590125458327168226458066526769958652682272807075781391858178889652
208164348344825993266043367660176999612831860788386150279465955131156552
036093988180612138558600301435694527224206344631797460594682573103790084
024432438465657245014402821885252470935190620929023136493273497565513958
720559654228749774011413346962715422845862377387538230483865688976461927
383814900140767310446640259899490222221765904339901886018566526485061799
702356193897017860040811889729918311021171229845901641921068884387121855
646124960798722908519296819372388642614839657382291123125024186649353143
970137428531926649875337218940694281434118520158014123344828015051399694
290153483077644569099073152433278288269864602789864321139083506217095002
597389863554277196742822248757586765752344220207573630569498825087968928
162753848863396909959826280956121450994871701244516461260379029309120889
086942028510640182154399457156805941872748998094254742173582401063677404
595741785160829230135358081840096996372524230560855903700624271243416909
004153690105933983835777939410970027753472000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000
000
```

4 Expression syntax

It is the expected one with infix operators and parentheses, the recognized operators being `+`, `-`, `*`, `/` (rounded division), `^` (power), `**` (power), `//` (truncated division), `/:` (modulo) and `!` (factorial).

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

The precedence rules are the expected ones. Notice though that in case of equal precedence the operations are left-associative, hence:

```
\thebnumexpr 2^3^4, (2^3)^4, 2^(3^4)\relax
```

```
4096, 4096, 2417851639229258349412352
```

The three operators `/`, `//`, `/:` are at the same level of precedence as the multiplication `*`. The modulo `/:` is associated with truncated division `//`.

The factorial postfix `!` has highest precedence. The minus signs inherit the precedence level of the previously encountered infix operators.

5 Option custom, \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 1.2 is required) and uses \bnumexprsetup in the following way:

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

The keys given to \bnumexprsetup must be lowercased. 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 ...\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.[1]

The sole package option is custom: it tells bnumexpr not to load xintcore.

6 Readme

```
| Source:  bnumexpr.dtx
| Version: v1.2a, 2015/10/14 (doc: 2015/10/14)
| Author:  Jean-Francois Burnol
| Info:    Expressions with big integers
| License: LPPL 1.3c or later

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

[1]see \BNE_Op_opp in the code for the default.

````
Usage
-----

The package `bnumexpr` allows _expandable_ computations with big
integers, the four infix operators `+`, `-`, `*`, `/` (which does
rounded division), the power operators `^` or `**`, the factorial
`!`, the truncated division `//`, and its associated modulo `/:`.

For example:

    \thebnumexpr (92874927979^5-3197927979^6)/30!\relax

outputs `-400624073659654394403518 9`.

The `\relax` ending token is mandatory and will be removed as a
result of the evaluation.

The expression parser is scaled-down from `\xinttheiiexpr...\relax`
from package xintexpr[^1], it does not handle boolean operators,
conditional branching, variables and recognizes no functions.

By default the underlying arithmetic macros are the ones provided
by package xintcore[^1] (its release 1.2 is required).

bnumexpr has only one option _custom_ which says to not load
xintcore, and a command `\bnumexprsetup` to inform the package
which macros to use if not those from xintcore.

This is a breaking release: some options and commands from `v1.1`
are not defined anymore (as `\bnumexprsetup` provides a new
interface), and documents which used them will need updating.

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 taylored
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
`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"`.

  :    Ignore dvipdfmx warnings, but if the pdf file has problems with
       fonts (possibly from an old dvipdfmx), use then rather pdflatex.

  :  - with pdflatex: `"pdflatex bnumexpr.tex"` (thrice).

  : 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) 2014-2015 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

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

7 Changes

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 \bnumexprusesbigintcalc.
- 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 package of the same name,
- added Changes and Readme sections to the documentation,
- better \BNE_protect mechanism for use of \bnumexpr ...\relax inside an \edef (without \bnethe). Previous one, inherited from xintexpr.sty 1.09n, assumed that the \.=<digits> dummy control sequence 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 LaTeX3. 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.

8 Package bnumexpr implementation

Contents

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 \errorcontextlines to at least 2 for more meaningful context.

8.1 Package identification and catcode setup

```
\NeedsTeXFormat{LaTeX2e}%
\ProvidesPackage{bnumexpr}[2015/10/14 v1.2a Expressions with big integers (jfB)]%
\edef\BNErestorecatcodes {\catcode`\noexpand\!\the\catcode`\!
                    \catcode`\noexpand\?\the\catcode`\?
                    \catcode`\noexpand\_\the\catcode`\_
                    \catcode`\noexpand\:\the\catcode`\:
                    \catcode`\noexpand\(\the\catcode`\(
                    \catcode`\noexpand\)\the\catcode`\)
                    \catcode`\noexpand\*\the\catcode`\*
                    \catcode`\noexpand\,\the\catcode`\,\relax }%
\catcode`\! 11
\catcode`\? 11
\catcode`\_ 11
\catcode`\: 11
\catcode`\, 12
\catcode`\* 12
\catcode`\( 12
```

8.2 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 \BNE_ prefix rather than \xint_. 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.

```
\chardef\xint_c_     0
\chardef\xint_c_i    1
\chardef\xint_c_ii   2
\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
\long\def\xint_gobble_i      #1{}%
\long\def\xint_gobble_iii    #1#2#3{}%
\long\def\xint_firstofone    #1{#1}%
\long\def\xint_firstoftwo    #1#2{#1}%
\long\def\xint_secondoftwo   #1#2{#2}%
\long\def\xint_firstofthree  #1#2#3{#1}%
\long\def\xint_secondofthree #1#2#3{#2}%
\long\def\xint_thirdofthree  #1#2#3{#3}%
\def\xint_gob_til_!          #1!{}% this ! has catcode 11
\def\xint_UDsignfork         #1-#2#3\krof {#2}%
\long\def\xint_afterfi       #1#2\fi {\fi #1}%
\long\def\xint_dothis        #1#2\xint_orthat #3{\fi #1}%
\let\xint_orthat             \xint_firstofone
\def\xint_zapspaces          #1 #2{#1#2\xint_zapspaces }%
```

8.3 \bnumexprsetup

New with v1.2a. Replaces removed \bnumexprUsesbigintcalc etc...

```
\catcode`! 3
\def\bnumexprsetup #1{\BNE_parsekeys #1,=!,}%
\def\BNE_parsekeys #1=#2#3,{\ifx!#2\expandafter\BNE_parsedone\fi
    \expandafter
\let\csname BNE_Op_\xint_zapspaces #1 \xint_gobble_i\endcsname=#2\BNE_parsekeys
}%
\catcode`! 11
\def\BNE_parsedone #1\BNE_parsekeys {}%
```

8.4 Package options

```
\def\BNE_tmpa {0}%
\DeclareOption {custom}{\def\BNE_tmpa {1}}%
\ProcessOptions\relax
\if0\BNE_tmpa % Default is to load xintcore.sty
    \RequirePackage{xintcore}[2015/10/10]%
    \bnumexprsetup{add=\xintiiAdd, sub=\xintiiSub, mul=\xintiiMul,
                   divround=\xintiiDivRound, divtrunc=\xintiiDivTrunc,
                   mod=\xintiiMod, pow=\xintiiPow, fac=\xintiiFac}%
\fi
```

8.5 \bnumexpr, \bnethe, \thebnumexpr, ...

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.

```
\def\bnumexpr {\romannumeral0\bnumeval }%
\def\bnumeval {\expandafter\BNE_wrap\romannumeral0\BNE_eval }%
\def\BNE_eval {\expandafter\BNE_until_end_a\romannumeral-`0\BNE_getnext }%
\def\BNE_wrap { !\BNE_usethe\BNE_protect\BNE_unlock }%
\protected\def\BNE_usethe\BNE_protect {\BNE:missing_bnethe!}%
\def\BNE_protect\BNE_unlock {\noexpand\BNE_protect\noexpand\BNE_unlock\noexpand }%
\let\BNE_done\space
\def\thebnumexpr
              {\romannumeral-`0\expandafter\BNE_unlock\romannumeral0\BNE_eval }%
\def\bnethe #1{\romannumeral-`0\expandafter\xint_gobble_iii\romannumeral-`0#1}%
\def\BNE_unlock   {\expandafter\BNE_unlock_a\string }%
\def\BNE_unlock_a #1.={}%
```

8.6 \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...

```
\def\BNE_getnext #1%
{%
    \expandafter\BNE_getnext_a\romannumeral-`0#1%
}%
\def\BNE_getnext_a #1%
{%
    \xint_gob_til_! #1\BNE_gn_foundexpr !% this ! has catcode 11
    \ifcat\relax#1% \count or \numexpr etc... token or count, dimen, skip cs
       \expandafter\BNE_gn_countetc
    \else
       \expandafter\expandafter\expandafter\BNE_gn_fork\expandafter\string
    \fi
    #1%
}%
\def\BNE_gn_foundexpr !#1\fi !{\expandafter\BNE_getop\xint_gobble_iii }%
\def\BNE_gn_countetc #1%
{%
```

```
    \ifx\count#1\else\ifx\dimen#1\else\ifx\numexpr#1\else\ifx\dimexpr#1\else
    \ifx\skip#1\else\ifx\glueexpr#1\else\ifx\fontdimen#1\else\ifx\ht#1\else
    \ifx\dp#1\else\ifx\wd#1\else\ifx\fontcharht#1\else\ifx\fontcharwd#1\else
    \ifx\fontchardp#1\else\ifx\fontcharic#1\else
      \BNE_gn_unpackvar
    \fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi
    \expandafter\BNE_getnext\number #1%
}%
\def\BNE_gn_unpackvar\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi
     \expandafter\BNE_getnext\number #1%
{%
  \fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi
  \expandafter\BNE_getop\csname .=\number#1\endcsname
}%
```

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..

```
\def\BNE_gn_fork #1{%
    \if#1+\xint_dothis \BNE_getnext\fi
    \if#1-\xint_dothis -\fi
    \if#1(\xint_dothis \BNE_oparen \fi
    \xint_orthat       {\BNE_scan_number #1}%
}%
```

8.7 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. . .

```
\def\BNE_scan_number #1% this #1 has necessarily here catcode 12
{%
    \ifnum \xint_c_ix<1#1 \else\expandafter\BNE_notadigit\fi
    \BNE_scan_nbr #1%
}%
\def\BNE_notadigit\BNE_scan_nbr #1{\BNE:not_a_digit!\BNE_getnext }%
```

Scanning for a number. We gather it directly inside csname. earlier version did a chain of romannumeral. No limit on number of digits anymore from the maximal expansion depth. We only have to be careful about leading zeros.

If we hit against some catcode eleven !, this means there was a sub \bnumexpr ..\relax. We then apply tacit multiplication.

```
\def\BNE_scan_nbr #1%
{% the #1 here is a catcode 12 digit
    \if#10\expandafter\BNE_scan_nbr_gobzeroes
    \else
          \expandafter\BNE_scan_nbr_start
    \fi #1%
}%
\def\BNE_scan_nbr_start #1#2%
```

```
{%
    \expandafter\BNE_getop\csname.=#1%
    \expandafter\BNE_scanint_b\romannumeral-`0#2%
}%
\def\BNE_scan_nbr_gobzeroes #1%
{%
    \expandafter\BNE_getop\csname.=%
    \expandafter\BNE_gobz_scanint_b\romannumeral-`0#1%
}%
\def\BNE_scanint_b #1%
{%
    \ifcat \relax #1\expandafter\BNE_scanint_endbycs\expandafter #1\fi
    \ifnum\xint_c_ix<1\string#1 \else\expandafter\BNE_scanint_c\fi
    \string#1\BNE_scanint_d
}%
\def\BNE_scanint_endbycs#1#2\BNE_scanint_d{\endcsname #1}%
\def\BNE_scanint_c\string #1\BNE_scanint_d
{%
    \ifcat a#1\xint_dothis{\endcsname*}\fi % tacit multiplication
    \xint_orthat {\expandafter\endcsname \string}#1%
}%
\def\BNE_scanint_d #1%
{%
    \expandafter\BNE_scanint_b\romannumeral-`0#1%
}%
\def\BNE_gobz_scanint_b #1%
{%
    \ifcat \relax #1\expandafter\BNE_gobz_scanint_endbycs\expandafter #1\fi
    \ifnum\xint_c_x<1\string#1 \else\expandafter\BNE_gobz_scanint_c\fi
    \string#1\BNE_scanint_d
}%
\def\BNE_gobz_scanint_endbycs#1#2\BNE_scanint_d{0\endcsname #1}%
\def\BNE_gobz_scanint_c\string #1\BNE_scanint_d
{%
    \ifcat a#1\xint_dothis{0\endcsname*#1}\fi % tacit multiplication
    \if    0#1\xint_dothis\BNE_gobz_scanint_d\fi
    \xint_orthat {0\expandafter\endcsname \string#1}%
}%
\def\BNE_gobz_scanint_d #1%
{%
    \expandafter\BNE_gobz_scanint_b\romannumeral-`0#1%
}%
```

8.8 \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.

v1.2a adds a technique for allowing two-letters operators, for //, /: and **.

```
\def\BNE_getop #1#2% this #1 is the current locked computed value
{%
    \expandafter\BNE_getop_a\expandafter #1\romannumeral-`0#2%
}%
\catcode`* 11
\def\BNE_getop_a #1#2%
{%  if a control sequence is found, must be \relax, or possibly register or
 %  variable if tacit multiplication is allowed
    \ifx \relax #2\xint_dothis\xint_firstofthree\fi
    % tacit multiplications:
    \ifcat \relax #2\xint_dothis\xint_secondofthree\fi
    \if    (#2\xint_dothis      \xint_secondofthree\fi
    \ifx   !#2\xint_dothis      \xint_secondofthree\fi
    \xint_orthat \xint_thirdofthree
    {\BNE_foundend #1}%
    {\BNE_precedence_* *#1#2}% tacit multiplication
    {\BNE_scanop_a #2#1}%
}%
\catcode`* 12
\def\BNE_foundend {\xint_c_ \relax }% \relax is only a place-holder here.
\def\BNE_scanop_a #1#2#3%
    {\expandafter\BNE_scanop_b\expandafter #1\expandafter #2\romannumeral-`0#3}%
\def\BNE_scanop_b #1#2#3%
{%
  \ifcat#3\relax\xint_dothis{\BNE_foundop #1#2#3}\fi
  \ifcsname BNE_itself_#1#3\endcsname
  \xint_dothis
        {\expandafter\BNE_foundop\csname BNE_itself_#1#3\endcsname #2}\fi
  \xint_orthat {\BNE_foundop #1#2#3}%
}%
\def\BNE_foundop #1%
{%
    \ifcsname BNE_precedence_#1\endcsname
        \csname BNE_precedence_#1\expandafter\endcsname
        \expandafter #1%
    \else
        \BNE_notanoperator {#1}\expandafter\BNE_getop
    \fi
}%
\def\BNE_notanoperator #1{\BNE:not_an_operator! \xint_gobble_i {#1}}%
```

8.9 Until macros for global expression and parenthesized sub-ones

The minus sign as prefix is treated here.

```
\catcode`) 11
\def\BNE_tmpa #1{%
    \def\BNE_until_end_a ##1%
    {%
        \xint_UDsignfork
            ##1{\expandafter\BNE_until_end_a\romannumeral-`0#1}%
            -{\BNE_until_end_b ##1}%
      \krof
    }%
}\expandafter\BNE_tmpa\csname BNE_op_-vi\endcsname
\def\BNE_until_end_b #1#2%
    {%
      \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{%
    \def\BNE_until_)_a ##1{\xint_UDsignfork
                         ##1{\expandafter \BNE_until_)_a\romannumeral-`0#1}%
                           -{\BNE_until_)_b ##1}%
                    \krof }%
}\expandafter\BNE_tmpa\csname BNE_op_-vi\endcsname
\def \BNE_until_)_b #1#2%
    {%
     \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
    }%
\def\BNE_missing_)_? {\BNE:missing_)_inserted \xint_c_ \BNE_done }%
\let\BNE_precedence_) \xint_c_i
\let\BNE_op_)   \BNE_getop
\catcode`) 12
```

8.10 The arithmetic operators.

This is where the infix operators are mapped to actual macros. These macros must ``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 ^).

```
\def\BNE_tmpc #1#2#3#4#5#6#7%
{%
  \def #1##1% \BNE_op_<op>
  {% keep value, get next number and operator, then do until
    \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 {^}% shortcut for alias
\expandafter\def\csname BNE_itself_//\endcsname {//}%
\expandafter\def\csname BNE_itself_/:\endcsname {/:}%
\BNE_tmpb  {//}{vii}{BNE_Op_divtrunc}%
\BNE_tmpb  {/:}{vii}{BNE_Op_mod}%
```

8.11 ! 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 }%
```

8.12 The minus as prefix operator of variable precedence level

It inherits the level of precedence of the previous operator.

```
\def\BNE_tmpa #1%
{%
\expandafter\BNE_tmpb
    \csname BNE_op_-#1\expandafter\endcsname
    \csname BNE_until_-#1_a\expandafter\endcsname
    \csname BNE_until_-#1_b\expandafter\endcsname
    \csname xint_c_#1\endcsname
}%
\def\BNE_tmpb #1#2#3#4%
{%
    \def #1% \BNE_op_-<level>
    {%  get next number+operator then switch to _until macro
        \expandafter #2\romannumeral-`0\BNE_getnext
    }%
    \def #2##1% \BNE_until_-<level>_a
    {\xint_UDsignfork
        ##1{\expandafter #2\romannumeral-`0#1}%
          -{#3##1}%
     \krof }%
    \def #3##1##2##3% \BNE_until_-<level>_b
    {%
        \ifnum ##1>#4%
         \xint_afterfi {\expandafter #2\romannumeral-`0%
                        \csname BNE_op_##2\endcsname {##3}}%
        \else
         \xint_afterfi {\expandafter ##1\expandafter ##2%
                        \csname .=\expandafter\BNE_Op_opp
                                  \romannumeral-`0\BNE_unlock ##3\endcsname }%
        \fi
    }%
}%
\BNE_tmpa {vi}%
\BNE_tmpa {vii}%
\BNE_tmpa {viii}%
\def\BNE_Op_opp #1{\if-#1\else\if0#10\else-#1\fi\fi }%
```

8.13 The comma may separate expressions.

It suffices to treat the comma as a binary operator of precedence ii. We insert a space after the comma. The current code in \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 \bnumexpr is identified as a sub-expression in \xintexpr, for example in: \xinttheiiexpr lcm(\bnumexpr 175-12,123+34,56*31\relax)\relax (this example requires package xintgcd).

```
\catcode`, 11
\def\BNE_op_, #1%
{%
    \expandafter \BNE_until_,_a\expandafter #1\romannumeral-`0\BNE_getnext
}%
\def\BNE_tmpa #1{% #1 = \BNE_op_-vi
  \def\BNE_until_,_a ##1##2%
  {%
    \xint_UDsignfork
        ##2{\expandafter \BNE_until_,_a\expandafter ##1\romannumeral-`0#1}%
          -{\BNE_until_,_b ##1##2}%
     \krof }%
}\expandafter\BNE_tmpa\csname BNE_op_-vi\endcsname
```

```
\def\BNE_until_,_b #1#2#3#4%
{%
    \ifnum #2>\xint_c_ii
        \xint_afterfi {\expandafter \BNE_until_,_a
                   \expandafter #1\romannumeral-`0%
                   \csname BNE_op_#3\endcsname {#4}}%
    \else
        \xint_afterfi {\expandafter #2\expandafter #3%
                       \csname .=\BNE_unlock #1, \BNE_unlock #4\endcsname }%
    \fi
}%
\let \BNE_precedence_, \xint_c_ii
```

8.14 Cleanup

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