# The bnumexpr package

Jean-François Burnol
jfbu (at) free (dot) fr

Package version: 1.4 (2021/05/12); documentation date: 2021/05/12.
From source file bnumexpr.dtx. Time-stamp: <12-05-2021 22:05:26 CEST>.

## Contents



## 1 \thebnumexpr, \bnumeval, \evaltohex

LaTeX Package bnumexpr provides \thebnumexpr⟨*expression*⟩\relax:[1] it is analogous to \the \numexpr⟨*expression*⟩\relax, with these extensions:

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

[1] Since 1.4, one can use \bnumexpr ...\relax directly in typesetting context, it is not mandatory to prefix it with \bnethe or to use \thebnumexpr.

There is also an alternative interface \bnumeval{⟨*expression*⟩}, where the expression is fetched as braced argument. And there is \evaltohex{⟨*expression*⟩} which does the same as \bnumeval{⟨*expression*⟩} but with a conversion to hexadecimal notation of the (possibly comma separated) output. Hexadecimal input uses the " prefix.

This package parser is a scaled-down variant of \xintiiexpr from package xintexpr, dropping support for nested structures, functions, variables, booleans, etc..., but incorporating by default support for hexadecimal input as xintbinhex will be automatically loaded.

The ε-TEX extensions are required, this is the default on all modern installations for latex|pdflatex and also for xelatex|lualatex.

Further, at 1.4 (2021/05/12) the \expanded primitive is required. It is available in all engines since TEXLive 2019.

By default the arithmetic operations are executed via the xintcore macros, but via option custom and usage of \bnumexprsetup it is possible to replace them by expandable macros of a custom origin.

## 2 Examples

```
\thebnumexpr ---1 208 637 867 * (2 187 917 891 - 3 109 197 072)\relax
```

```
1113492904235346927
```

```
\bnumeval {(13_8089_1090-300_1890_2902)*(1083_1908_3901-109_8290_3890)}
```

```
-2787514672889976289932
```

```
\thebnumexpr (92_874_927_979**5-31_9792_7979**6)/30!\relax
```

```
-4006240736596543944035189
```

```
\bnumeval {30!/20!/21/22/23/24/25/(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, 91004381500021497733275852753425663249271526032565862 4
```

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

```
11868507546269898170062082 8125
```

```
\bnumeval {100!/36^100}
```

```
219
```

```
\bnumeval {"10*"100*"1000*"A0000, 16^(1+2+3+4)*10}

                    10995116277760,10995116277760

\evaltohex {"7FFFFFFF+1, "400^3, "ABCDEF*"FEDCBA}

                    80000000,40000000,AB0A74EF03A6
```

## 3 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 may embed directly \bnumexpr ...\relax in another one (or in a \xintexpr ...\relax), but not in a \numexpr ...\relax: it must then be using \thebnumexpr ... or \bnethe \bnumexpr ... syntax; on the other hand a \numexpr ...\relax does not need to be prefixed by \the or \number inside \bnumexpr ...\relax.

2. contrarily to \numexpr, the \bnumexpr parser stops only after having found (and swallowed) a mandatory ending \relax token,

3. 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

4. one may do \edef \variable {\bnumexpr 1+2\relax }, and then either use \variable in another \bnumexpr ...\relax, or print it via \bnethe \variable (or directly since 1.4). 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,

5. 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,

6. \bnumexpr -(1+1)\relax is legal contrarily to \numexpr -(1+1)\relax which raises an error,

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

8. more generally, tacit multiplication applies in front of parenthesized sub-expressions, or sub \bnumexpr ...\relax (or \numexpr ...\relax), or also after parentheses in front of numbers,

9. the underscore _ is accepted within the digits composing a number and is silently ignored by \bnumexpr.

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.

# 4 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
21071463254379991042993851239862902059204420848696940480047998861019719 6
05863166687299480855890132382966994459099742450408707375991882362772718 8
73251977950595099527612087497546249704360141827809464649629105639388743 7
88648733711918104582578364784997701247663288983595573543251318532395846 3
07555740911426241747434934755342864657661166779739666882029120737914385 3
71958824980812686783837455973174613608537953452422158659320192809087829 7
30843139284440328123155861103697680135730421616874760967587134831202547 8
58932076716913244842623613141250878020800026168315102734182797770478463 5
86817016436502415369139828126481021309276124489635992870511496497541990 9
34222156683257208082133318611681155361583654698404670897560290095053761 6
47584772842188967964624494516076535340819890138544248798495995331910172 3
35555660213945039973628075013783761530712776192684903435262520001588853 5
14733161170210396817592151090778801939317811419454525722386554146106289 2
18796022383897147608850627686296714667469756291123408243920816015378088 9
89396451826324367161676217916890977991190375403127462228998800519544441 4
28201218736174599264295658174662830295557029902432415318161721046583203 6
78690611726015878352075151628422554026517048330422614397428693306169089 7
96848259012545832716822645806652676995865268227280707578139185817888965 2
20816434834482599326604336766017699961283186078838615027946595513115655 2
03609398818061213855860030143569452722420634463179746059468257310379008 4
02443243846565724501440282188525247093519062092902313649327349756551395 8
72055965422874977401141334696271542284586237738753823048386568897646192 7
38381490014076731044664025989949022222176590433990188601856652648506179 9
```

```
70235619389701786004081188972991831102117122984590164192106888438712185 5
64612496079872290851929681937238864261483965738229112312502418664935314 3
97013742853192664987533721894069428143411852015801412334482801505139969 4
29015348307764456909907315243327828826986460278986432113908350621709500 2
59738986355427719674282224875758676575234422020757363056949882508796892 8
16275384886339690995982628095612145099487170124451646126037902930912088 9
08694202851064018215439945715680594187274899809425474217358240106367740 4
59574178516082923013535808184009699637252423056085590370062427124341690 9
00415369010593398383577793941097002775347200000000000000000000000000000 0
00000000000000000000000000000000000000000000000000000000000000000000000 0
00000000000000000000000000000000000000000000000000000000000000000000000 0
00000000000000000000000000000000000000000000000000000000000000000000000 0
000
```

# 5 Expression syntax

It is the expected one with infix operators and parentheses, the recognized operators being `+`, `-`, `*`, `/` (rounded division), `^` (power), `**` (power), `//` (by default floored division), `/:` (the associated modulo) and `!` (factorial). One can input hexadecimal numbers as in TEX syntax for number assignments, i.e. using a `"` prefix and only uppercase letters `ABCDEF`.

The modulo `/:` is by default associated with the floored division `//`, but using `\bnumexprsetup {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 `\relax`) may arise on the spot from macro expansions.

The precedence rules are the expected ones. Tacit multiplication applies in front of parentheses, and after them, and it has an elevated precedence compared to multiplication explicitly induced by `*`.

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 `!` 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,[2] hence:

```
\bnumeval {2^3^4, (2^3)^4, 2^(3^4)}
```

4096, 4096, 241785163922925834941235 2

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

[2] But it has been announced at xintexpr 1.4 that probably in future power operator will become right-associative, and if this is done, this will be transferred to bnumexpr also.

# 6 Options

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

# 7 \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) 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 ...\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.

At 1.4 release there is no option to customize how the hexadecimal input and output is handled, it goes necessarily via the xintbinhex macros.

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

# 8 Readme

```
| Source:  bnumexpr.dtx
| Version: v1.4, 2021/05/12 (doc: 2021/05/12)
| Author:  Jean-Francois Burnol
| Info:    Expressions with big integers
| License: LPPL 1.3c

bnumexpr usage
==============

The LaTeX package `bnumexpr` allows expandable computations with
integers and the four infix operators `+`, `-`, `*`, `/` using the
expression syntax familiar from the `\numexpr` e-TeX parser, with
these extensions:

- arbitrarily big integers,
- floored division `//`,
- associated modulo `/:`,
- power operators `^` and `**`,
- factorial post-fix operator `!`,
- comma separated expressions,
- the space character as well as the underscore may serve
  to separate groups of digits,
- optional conversion of output to hexadecimal.

The expression parser is a scaled-down variant from the
`\xintiiexpr...\relax`
parser from package [xintexpr](http://ctan.org/pkg/xintexpr).

To support hexadecimal input and output, the package
[xintbinhex](http://ctan.org/pkg/xint) is loaded automatically.

The package loads by default [xintcore](http://ctan.org/pkg/xint)
but the option _custom_ together with macro `\bnumexprsetup` allow to map
the syntax elements to macros from an alternative big integer
expandable engine of the user own choosing,
and then [xintcore](http://ctan.org/pkg/xint) is not loaded.

Note; 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.

Installation
============

Use your installation manager to install or update `bnumexpr`.

Else, obtain `bnumexpr.dtx`, from CTAN:

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

Run `"etex bnumexpr.dtx"` to extract these files:

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

`README.md`

`bnumexprchanges.tex`
```

```
  : change history.

`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).

  : In both cases files `README.md` and `bnumexprchanges.tex` must
    be located in the same repertory as `bnumexpr.tex` and `bnumexpr.dtx`.

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.pdf   --> TDS:doc/latex/bnumexpr/
               README.me   --> TDS:doc/latex/bnumexpr/

License
=======

Copyright (C) 2014-2021 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.tex, bnumexprchanges.tex,
    and README.md
```

# 9 Changes

**1.4 (2021/05/12)**

- technology transfer from xintexpr 1.4 of 2020/01/31. The \expanded primitive is now required (TeXLive 2019).
- addition to the syntax of the " prefix for hexadecimal input.
- addition of \evaltohex which is like \bnumeval with an extra conversion step to hexadecimal notation.

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

**1.2d (2019/01/07)**
- requires xintcore 1.3d or later (if not using option custom).
- adds \bnumeval{⟨*expression*⟩} 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 digits 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 \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.

# 10 Package bnumexpr implementation

## Contents



Comments are sparse. Actually at 1.4, there are simply no comments. I transferred from xintexpr its \expanded based infra-structure from its own 1.4 release of January 2020. It is a possibility that, even though I did remove very large chunks of unneeded macros, in the end, some more simplifications could have been considered here.

Error handling by the parser is kept to a minimum; if something goes wrong, the offensive token gets discarded, and it is not even always the case that some expandable error message is issued.

### 10.1 Package identification and catcode setup

```
\NeedsTeXFormat{LaTeX2e}%
\ProvidesPackage{bnumexpr}[2021/05/12 v1.4 Expressions with big integers (JFB)]%
```

### 10.2 Load unconditionally xintbinhex

Newly done at 1.4. Formerly, bnumexpr had no dependency if loaded with option custom. But for 1.4 release I have decided to add unconditional support for hexadecimal notation.

Let's require the most recent xint date at time of writing. We should check for availability of \expanded but well.

```
\RequirePackage{xintbinhex}[2021/05/10]%
```

### 10.3 \bnumexprsetup

```
{\catcode`! 3 \catcode`_ 11 %
  \gdef\bnumexprsetup #1{\BNE_parsekeys #1,=!,}%
  \gdef\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
  }%
  \gdef\BNE_parsedone #1\BNE_parsekeys {}%
}%
```

## 10.4 Package options

```
\def\BNEtmpa {0}%
\DeclareOption {custom}{\def\BNEtmpa {1}}%
\ProcessOptions\relax
\edef\BNErestorecatcodes{\XINTrestorecatcodes}%
\XINTsetcatcodes%
\if0\BNEtmpa\expandafter\xint_secondoftwo\fi
\xint_gobble_i{%
    \RequirePackage{xintcore}[2021/05/10]%
    \bnumexprsetup{add=\xintiiAdd, sub=\xintiiSub, mul=\xintiiMul,
                   divround=\xintiiDivRound, div=\xintiiDivFloor,
                   mod=\xintiiMod, pow=\xintiiPow, fac=\xintiiFac}%
}%
```

Strangely those three are not defined in xintkernel.sty, but only in xint.sty

```
\long\def\xint_firstofthree  #1#2#3{#1}%
\long\def\xint_secondofthree #1#2#3{#2}%
\long\def\xint_thirdofthree  #1#2#3{#3}%
```

## 10.5 \bnumexpr, \thebnumexpr, \bnethe, \bnumeval

```
\def\XINTfstop {\noexpand\XINTfstop}%
\def\bnumexpr  {\romannumeral0\bnumexpro}%
\def\bnumexpro {\expandafter\BNE_wrap\romannumeral0\bnebareeval }%
\def\BNE_wrap  {\XINTfstop\BNEprint.}%
\def\bnumeval #1%
   {\expanded\expandafter\BNEprint\expandafter.\romannumeral0\bnebareeval#1\relax}%
\def\evaltohex #1%
   {\expanded\expandafter\BNEprinthex\expandafter.\romannumeral0\bnebareeval#1\relax}%
\def\thebnumexpr
   {\expanded\expandafter\BNEprint\expandafter.\romannumeral0\bnebareeval}%
\def\bnebareeval{\BNE_start}%
\def\bnethe#1{\expanded\expandafter\xint_gobble_i\romannumeral`&&@#1}%
\protected\def\BNEprint.#1{{\BNE_print#1.}}%
\def\BNE_print#1{#1\expandafter\BNE_print_a\string}%
\def\BNE_print_a#1{\unless\if#1.\expandafter\BNE_print_b\fi}%
\def\BNE_print_b
   {\expandafter\BNE_print_c\expandafter{\expandafter\xint_gobble_i\string}}%
\def\BNE_print_c#1{, #1\expandafter\BNE_print_a\string}%
\protected\def\BNEprinthex.#1{{\BNE_printhex#1.}}%
\def\BNE_printhex#1{\xintDecToHex{#1}\expandafter\BNE_printhex_a\string}%
\def\BNE_printhex_a#1{\unless\if#1.\expandafter\BNE_printhex_b\fi}%
\def\BNE_printhex_b
   {\expandafter\BNE_printhex_c\expandafter{\expandafter\xint_gobble_i\string}}%
\def\BNE_printhex_c#1{, \xintDecToHex{#1}\expandafter\BNE_printhex_a\string}%
```

## 10.6 \BNE_getnext

```
\def\BNE_getnext #1%
{%
    \expandafter\BNE_put_op_first\romannumeral`&&@%
    \expandafter\BNE_getnext_a\romannumeral`&&@#1%
}%
\def\BNE_put_op_first #1#2#3{\expandafter#2\expandafter#3\expandafter{#1}}%
\def\BNE_getnext_a #1%
{%
    \ifx\relax #1\xint_dothis\BNE_foundprematureend\fi
    \ifx\XINTfstop#1\xint_dothis\BNE_subexpr\fi
    \ifcat\relax#1\xint_dothis\BNE_countetc\fi
    \xint_orthat{}\BNE_getnextfork #1%
}%
\def\BNE_foundprematureend\BNE_getnextfork #1{{}\xint_c_\relax}%
\def\BNE_subexpr #1.#2%
{%
    \expanded{\unexpanded{{#2}}\expandafter}\romannumeral`&&@\BNE_getop
}%
\def\BNE_countetc\BNE_getnextfork#1%
{%
    \if0\ifx\count#11\fi
        \ifx\dimen#11\fi
        \ifx\numexpr#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 0\expandafter\BNE_fetch_as_number\fi
   \expandafter\BNE_getnext_a\number #1%
}%
\def\BNE_fetch_as_number
    \expandafter\BNE_getnext_a\number #1%
{%
    \expanded{{{\number#1}}\expandafter}\romannumeral`&&@\BNE_getop
}%
```

In the case of hitting a `(`, previous release inserted directly a `\BNE_oparen`. But the expansion architecture imported from upstream `\xintiiexpr` has been refactored, and the `..._oparen` meaning and usage evolved. We stick with `{}\xint_c_ii ^v (` from upstream, which works (I am 15 months away from `xintexpr 1.4`).

```
\def\BNE_getnextfork #1{%
    \if#1+\xint_dothis \BNE_getnext_a \fi
    \if#1-\xint_dothis {{}{}-}\fi
    \if#1(\xint_dothis {{}\xint_c_ii^v (}\fi
    \xint_orthat {\BNE_scan_number #1}%
```

```
}%
```

## 10.7 Parsing an integer in decimal or hexadecimal notation

```
\def\BNE_scan_number #1%
{%
    \if "#1\xint_dothis \BNE_scanhex\fi
    \ifnum \xint_c_ix<1\string#1 \xint_dothis \BNE_startint\fi
    \xint_orthat \BNE_notadigit #1%
}%
\def\BNE_notadigit#1{\BNE_getnext }%
\def\BNE_startint #1%
{%
    \if #10\expandafter\BNE_gobz_a\else\expandafter\BNE_scanint_a\fi #1%
}%
\def\BNE_scanint_a #1#2%
    {\expanded\bgroup{{\iffalse}}\fi #1%
     \expandafter\BNE_scanint_main\romannumeral`&&@#2}%
\def\BNE_gobz_a #1#2%
    {\expanded\bgroup{{\iffalse}}\fi
     \expandafter\BNE_gobz_scanint_main\romannumeral`&&@#2}%
\def\BNE_scanint_main #1%
{%
    \ifcat \relax #1\expandafter\BNE_scanint_hit_cs \fi
    \ifnum\xint_c_ix<1\string#1 \else\expandafter\BNE_scanint_next\fi
    #1\BNE_scanint_again
}%
\def\BNE_scanint_again #1%
{%
    \expandafter\BNE_scanint_main\romannumeral`&&@#1%
}%
\def\BNE_scanint_hit_cs \ifnum#1\fi#2\BNE_scanint_again
{%
    \iffalse{{{\fi}}\expandafter}\romannumeral`&&@\BNE_getop#2%
}%
\def\BNE_scanint_next #1\BNE_scanint_again
{%
    \if    _#1\xint_dothis\BNE_scanint_again\fi
    \xint_orthat
    {\iffalse{{{\fi}}\expandafter}\romannumeral`&&@\BNE_getop#1}%
}%
\def\BNE_gobz_scanint_main #1%
{%
    \ifcat \relax #1\expandafter\BNE_gobz_scanint_hit_cs\fi
    \ifnum\xint_c_x<1\string#1 \else\expandafter\BNE_gobz_scanint_next\fi
    #1\BNE_scanint_again
}%
\def\BNE_gobz_scanint_again #1%
{%
    \expandafter\BNE_gobz_scanint_main\romannumeral`&&@#1%
}%
\def\BNE_gobz_scanint_hit_cs\ifnum#1\fi#2\BNE_scanint_again
{%
```

```
    0\iffalse{{{\fi}}\expandafter}\romannumeral`&&@\BNE_getop#2%
}%
\def\BNE_gobz_scanint_next #1\BNE_scanint_again
{%
    \if    _#1\xint_dothis\BNE_gobz_scanint_again\fi
    \if    0#1\xint_dothis\BNE_gobz_scanint_again\fi
    \xint_orthat
    {0\iffalse{{{\fi}}\expandafter}\romannumeral`&&@\BNE_getop#1}%
}%
\def\BNE_hex_in #1.%
{%
    \expanded{{{\xintHexToDec{#1}}}\expandafter}\romannumeral`&&@\BNE_getop
}%
\def\BNE_scanhex #1% #1="
{%
    \expandafter\BNE_hex_in\expanded\bgroup\BNE_scanhex_a
}%
\def\BNE_scanhex_a #1%
{%
    \ifcat #1\relax\xint_dothis{.\iffalse{\fi}#1}\fi
    \xint_orthat {\BNE_scanhex_aa #1}%
}%
\def\BNE_scanhex_aa #1%
{%
    \if\ifnum`#1>`/
       \ifnum`#1>`9
       \ifnum`#1>`@
       \ifnum`#1>`F
       0\else1\fi\else0\fi\else1\fi\else0\fi 1%
       \expandafter\BNE_scanhex_b
    \else
       \if _#1\xint_dothis{\expandafter\BNE_scanhex_bgob}\fi
       \xint_orthat {\xint_afterfi {.\iffalse{\fi}}}%
    \fi
    #1%
}%
\def\BNE_scanhex_b #1#2%
{%
    #1\expandafter\BNE_scanhex_a\romannumeral`&&@#2%
}%
\def\BNE_scanhex_bgob #1#2%
{%
    \expandafter\BNE_scanhex_a\romannumeral`&&@#2%
}%
```

## 10.8 \BNE_getop

```
\def\BNE_getop #1%
{%
    \expandafter\BNE_getop_a\romannumeral`&&@#1%
}%
\catcode`* 11
\def\BNE_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_secondofthree\fi %)
    \xint_orthat \xint_thirdofthree
    {\BNE_foundend}%
    {\BNE_precedence_*** *#1}%
    {\expandafter\BNE_scanop_a \string#1}%
}%
\catcode`* 12
\def\BNE_foundend {\xint_c_ \relax}%
\def\BNE_scanop_a #1#2%
{%
    \expandafter\BNE_scanop_b\expandafter#1\romannumeral`&&@#2%
}%
\def\BNE_scanop_b #1#2%
{%
    \ifcat#2\relax\xint_dothis{\BNE_foundop_a #1#2}\fi
    \ifcsname BNE_itself_#1#2\endcsname
    \xint_dothis
        {\expandafter\BNE_scanop_c\csname BNE_itself_#1#2\endcsname}\fi
    \xint_orthat {\BNE_foundop_a #1#2}%
}%
\def\BNE_scanop_c #1#2%
{%
    \expandafter\BNE_scanop_d\expandafter#1\romannumeral`&&@#2%
}%
\def\BNE_scanop_d #1#2%
{%
  \ifcat#2\relax \xint_dothis{\BNE_foundop #1#2}\fi
  \ifcsname BNE_itself_#1#2\endcsname
  \xint_dothis
        {\expandafter\BNE_scanop_c\csname BNE_itself_#1#2\endcsname }\fi
  \xint_orthat {\csname BNE_precedence_#1\endcsname #1#2}%
}%
\def\BNE_foundop_a #1%
{%
    \ifcsname BNE_precedence_#1\endcsname
        \csname BNE_precedence_#1\expandafter\endcsname
        \expandafter #1%
    \else
        \xint_afterfi{\BNE_getop\romannumeral0%
        \XINT_expandableerror
        {"#1" is unknown as operator. (I)nsert one:} }%<<deliberate space
    \fi
}%
\def\BNE_foundop #1{\csname BNE_precedence_#1\endcsname #1}%
```

## 10.9 Expansion spanning; opening and closing parentheses

```
\def\BNE_tmpa #1#2#3#4#5%
{%
```

```
    \def#1% start
    {%
        \expandafter#2\romannumeral`&&@\BNE_getnext
    }%
    \def#2##1% check
    {%
        \xint_UDsignfork
          ##1{\expandafter#3\romannumeral`&&@#4}%
            -{#3##1}%
        \krof
    }%
    \def#3##1##2% checkp
    {%
        \ifcase ##1%
           \expandafter\BNE_done
        \or\expandafter#5%
        \else
           \expandafter#3\romannumeral`&&@\csname BNE_op_##2\expandafter\endcsname
        \fi
    }%
    \def#5%
    {%
        \XINT_expandableerror
        {An extra ) has been removed. Hit Return, fingers crossed.}%
        \expandafter#2\romannumeral`&&@\expandafter\BNE_put_op_first
        \romannumeral`&&@\BNE_getop_legacy
    }%
}%
\let\BNE_done\space
\expandafter\BNE_tmpa
    \csname BNE_start\expandafter\endcsname
    \csname BNE_check\expandafter\endcsname
    \csname BNE_checkp\expandafter\endcsname
    \csname BNE_op_-xii\expandafter\endcsname
    \csname BNE_extra_)\endcsname
\def\BNE_tmpa #1#2#3#4#5#6%
{%
    \def #1##1% op_(
    {%
        \expandafter #4\romannumeral`&&@\BNE_getnext
    }%
    \def #2##1% op_)
    {%
        \expanded{\unexpanded{\BNE_put_op_first{##1}}\expandafter}\romannumeral`&&@\BNE_getop
    }%
    \def #3% oparen
    {%
        \expandafter #4\romannumeral`&&@\BNE_getnext
    }%
    \def #4##1% check-
    {%
        \xint_UDsignfork
```

```
            ##1{\expandafter#5\romannumeral`&&@#6}%
              -{#5##1}%
        \krof
    }%
    \def #5##1##2% checkp
    {%
        \ifcase ##1\expandafter\BNE_missing_)
        \or \csname BNE_op_##2\expandafter\endcsname
        \else
          \expandafter #5\romannumeral`&&@\csname BNE_op_##2\expandafter\endcsname
        \fi
    }%
}%
\expandafter\BNE_tmpa
    \csname BNE_op_(\expandafter\endcsname
    \csname BNE_op_)\expandafter\endcsname
    \csname BNE_oparen\expandafter\endcsname
    \csname BNE_check-_)\expandafter\endcsname
    \csname BNE_checkp_)\expandafter\endcsname
    \csname BNE_op_-xii\endcsname
\catcode`) 11
\let\BNE_precedence_)\xint_c_i
\def\BNE_missing_)
   {\XINT_expandableerror{Sorry to report a missing ) at the end of this journey.}%
    \xint_c_ \BNE_done }%
\catcode`) 12
```

## 10.10 The comma as binary operator

```
\def\BNE_tmpa #1#2#3#4#5%
{%
    \def #1##1% \BNE_op_,
    {%
      \expanded{\unexpanded{#2{##1}}\expandafter}%
      \romannumeral`&&@\expandafter#3\romannumeral`&&@\BNE_getnext
    }%
    \def #2##1##2##3##4{##2##3{##1##4}}% \BNE_exec_,
    \def #3##1% \BNE_check-_,
    {%
      \xint_UDsignfork
        ##1{\expandafter#4\romannumeral`&&@#5}%
          -{#4##1}%
      \krof
    }%
    \def #4##1##2% \BNE_checkp_,
    {%
      \ifnum ##1>\xint_c_iii
        \expandafter#4%
           \romannumeral`&&@\csname BNE_op_##2\expandafter\endcsname
      \else
        \expandafter##1\expandafter##2%
      \fi
    }%
```

```
}%
\expandafter\BNE_tmpa
    \csname BNE_op_,\expandafter\endcsname
    \csname BNE_exec_,\expandafter\endcsname
    \csname BNE_check-_,\expandafter\endcsname
    \csname BNE_checkp_,\expandafter\endcsname
    \csname BNE_op_-xii\endcsname
\expandafter\let\csname BNE_precedence_,\endcsname\xint_c_iii
```

## 10.11 The minus as prefix operator of variable precedence level

```
\def\BNE_tmpb #1#2#3#4#5%
{%
    \def #1% \BNE_op_-<level>
    {%
      \expandafter #2\romannumeral`&&@\expandafter#3%
      \romannumeral`&&@\BNE_getnext
    }%
    \def #2##1##2##3% \BNE_exec_-<level>
    {%
      \expandafter ##1\expandafter ##2\expandafter
       {\expandafter{\romannumeral`&&@\expandafter\BNE_Op_opp\xint_firstofone##3}}%
    }%
    \def #3##1% \BNE_check-_-<level>
    {%
      \xint_UDsignfork
        ##1{\expandafter #4\romannumeral`&&@#1}%
          -{#4##1}%
      \krof
    }%
    \def #4##1##2% \BNE_checkp_-<level>
    {%
      \ifnum ##1>#5%
        \expandafter #4%
        \romannumeral`&&@\csname BNE_op_##2\expandafter\endcsname
      \else
        \expandafter ##1\expandafter ##2%
      \fi
    }%
}%
\def\BNE_tmpa #1%
{%
\expandafter\BNE_tmpb
    \csname BNE_op_-#1\expandafter\endcsname
    \csname BNE_exec_-#1\expandafter\endcsname
    \csname BNE_check-_-#1\expandafter\endcsname
    \csname BNE_checkp_-#1\expandafter\endcsname
    \csname xint_c_#1\endcsname
}%
\BNE_tmpa {xii}%
\BNE_tmpa {xiv}%
\BNE_tmpa {xvi}%
\BNE_tmpa {xviii}%
```

```
\def\BNE_Op_opp #1{\if-#1\else\if0#10\else-#1\fi\fi }%
```

## 10.12 The infix operators.

```
\def\BNE_defbin_c #1#2#3#4#5#6#7%
{%
  \def #1##1% \BNE_op_<op>
  {%
    \expanded{\unexpanded{#2{##1}}\expandafter}%
    \romannumeral`&&@\expandafter#3\romannumeral`&&@\BNE_getnext
  }%
  \def #2##1##2##3##4% \BNE_exec_<op>
  {%
    \expandafter##2\expandafter##3\expandafter
      {\expandafter{\romannumeral`&&@#6##1##4}}%
  }%
  \def #3##1% \BNE_check-_<op>
  {%
    \xint_UDsignfork
      ##1{\expandafter#4\romannumeral`&&@#5}%
        -{#4##1}%
    \krof
  }%
  \def #4##1##2% \BNE_checkp_<op>
  {%
    \ifnum ##1>#7%
      \expandafter#4%
      \romannumeral`&&@\csname BNE_op_##2\expandafter\endcsname
    \else
      \expandafter ##1\expandafter ##2%
    \fi
  }%
}%
\def\BNE_defbin_b #1#2#3#4%
{%
  \expandafter\BNE_defbin_c
  \csname BNE_op_#1\expandafter\endcsname
  \csname BNE_exec_#1\expandafter\endcsname
  \csname BNE_check-_#1\expandafter\endcsname
  \csname BNE_checkp_#1\expandafter\endcsname
  \csname BNE_op_-#3\expandafter\endcsname
  \csname #4\expandafter\endcsname
  \csname BNE_precedence_#1\endcsname
  \expandafter
  \let\csname BNE_precedence_#1\expandafter\endcsname
      \csname xint_c_#2\endcsname
}%
\BNE_defbin_b {//} {xiv}{xiv}{BNE_Op_div}%
\BNE_defbin_b {/:} {xiv}{xiv}{BNE_Op_mod}%
\BNE_defbin_b  +   {xii}{xii}{BNE_Op_add}%
\BNE_defbin_b  -   {xii}{xii}{BNE_Op_sub}%
\BNE_defbin_b  *   {xiv}{xiv}{BNE_Op_mul}%
\BNE_defbin_b  /   {xiv}{xiv}{BNE_Op_divround}%
```

```
\BNE_defbin_b  ^   {xviii}{xviii}{BNE_Op_pow}%
\expandafter\def\csname BNE_itself_**\endcsname {^}%
\expandafter\def\csname BNE_itself_//\endcsname {//}%
\expandafter\def\csname BNE_itself_/:\endcsname {/:}%
\expandafter\let\csname BNE_precedence_***\endcsname \xint_c_xvi
```

## 10.13 ! as postfix factorial operator

```
\catcode`! 11
\let\BNE_precedence_! \xint_c_xx
\def\BNE_op_! #1%
{%
    \expandafter\BNE_put_op_first
    \expanded{{{\BNE_Op_fac#1}}\expandafter}\romannumeral`&&@\BNE_getop
}%
```

## 10.14 Cleanup

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