The `calc` package
Infix notation arithmetic in LaTeX*

Kresten Krab Thorup, Frank Jensen (and Chris Rowley)

1998/07/07

Abstract

The `calc` package reimplements the LaTeX commands `\setcounter`, `\addtocounter`, `\setlength`, and `\addtolength`. Instead of a simple value, these commands now accept an infix notation expression.

1 Introduction

Arithmetic in TeX is done using low-level operations such as `\advance` and `\multiply`. This may be acceptable when developing a macro package, but it is not an acceptable interface for the end-user.

This package introduces proper infix notation arithmetic which is much more familiar to most people. The infix notation is more readable and easier to modify than the alternative: a sequence of assignment and arithmetic instructions. One of the arithmetic instructions (`\divide`) does not even have an equivalent in standard LaTeX.

The infix expressions can be used in arguments to macros (the `calc` package doesn't employ category code changes to achieve its goals)[1].

2 Informal description

Standard LaTeX provides the following set of commands to manipulate counters and lengths [2, pages 194 and 216].

`\setcounter{`*ctr*`}{`*num*`}` sets the value of the counter *ctr* equal to (the value of) *num*. (Fragile)

`\addtocounter{`*ctr*`}{`*num*`}` increments the value of the counter *ctr* by (the value of) *num*. (Fragile)

`\setlength{`*cmd*`}{`*len*`}` sets the value of the length command *cmd* equal to (the value of) *len*. (Robust)

`\addtolength{`*cmd*`}{`*len*`}` sets the value of the length command *cmd* equal to its current value plus (the value of) *len*. (Robust)

(The `\setcounter` and `\addtocounter` commands have global effect, while the `\setlength` and `\addtolength` commands obey the normal scoping rules.) In standard LaTeX, the arguments to these commands must be simple values. The `calc` package

*We thank Frank Mittelbach for his valuable comments and suggestions which have greatly improved this package.

[1] However, it therefore assumes that the category codes of the special characters, such as `(*/)` in its syntax do not change.

extends these commands to accept infix notation expressions, denoting values of appropriate types. Using the `calc` package, *num* is replaced by ⟨integer expression⟩, and *len* is replaced by ⟨glue expression⟩. The formal syntax of ⟨integer expression⟩ and ⟨glue expression⟩ is given below.

In addition to these commands to explicitly set a length, many LaTeX commands take a length argument. After loading this package, most of these commands will accept a ⟨glue expression⟩. This includes the optional width argument of `\makebox`, the width argument of `\parbox`, `minipage`, and a `tabluar` p-column, and many similar constructions. (This package does not redefine any of these commands, but they are defined by default to read their arguments by `\setlength` and so automatically benefit from the enhanced `\setlength` command provided by this package.)

In the following, we shall use standard TeX terminology. The correspondence between TeX and LaTeX terminology is as follows: LaTeX counters correspond to TeX's count registers; they hold quantities of type ⟨number⟩. LaTeX length commands correspond to TeX's dimen (for rigid lengths) and skip (for rubber lengths) registers; they hold quantities of types ⟨dimen⟩ and ⟨glue⟩, respectively.

TeX gives us primitive operations to perform arithmetic on registers as follows:

- addition and subtraction on all types of quantities without restrictions;
- multiplication and division by an *integer* can be performed on a register of any type;
- multiplication by a *real* number (i.e., a number with a fractional part) can be performed on a register of any type, but the stretch and shrink components of a glue quantity are discarded.

The `calc` package uses these TeX primitives but provides a more user-friendly notation for expressing the arithmetic.

An expression is formed of numerical quantities (such as explicit constants and LaTeX counters and length commands) and binary operators (the tokens '`+`', '`-`', '`*`', and '`/`' with their usual meaning) using the familiar infix notation; parentheses may be used to override the usual precedences (that multiplication/division have higher precedence than addition/subtraction).

Expressions must be properly typed. This means, e.g., that a dimen expression must be a sum of dimen terms: i.e., you cannot say '`2cm+4`' but '`2cm+4pt`' is valid.

In a dimen term, the dimension part must come first; the same holds for glue terms. Also, multiplication and division by non-integer quantities require a special syntax; see below.

Evaluation of subexpressions at the same level of precedence proceeds from left to right. Consider a dimen term such as "`4cm*3*4`". First, the value of the factor `4cm` is assigned to a dimen register, then this register is multiplied by 3 (using `\multiply`), and, finally, the register is multiplied by 4 (again using `\multiply`). This also explains why the dimension part (i.e., the part with the unit designation) must come first; TeX simply doesn't allow untyped constants to be assigned to a dimen register.

The `calc` package also allows multiplication and division by real numbers. However, a special syntax is required: you must use `\real{`⟨decimal constant⟩`}`[2] or `\ratio{`⟨dimen expression⟩`}{`⟨dimen expression⟩`}` to denote a real value to be used for multiplication/division. The first form has the obvious meaning, and the second form denotes the number obtained by dividing the value of the first expression by the value of the second expression.

A later addition to the package (in June 1998) allows an additional method of specifying a factor of type dimen by setting some text (in LR-mode) and measuring its dimensions: these are denoted as follows.

[2] Actually, instead of ⟨decimal constant⟩, the more general ⟨optional signs⟩⟨factor⟩ can be used. However, that doesn't add any extra expressive power to the language of infix expressions.

`\widthof{`⟨text⟩`}` `\heightof{`⟨text⟩`}` `\depthof{`⟨text⟩`}`

These calculate the natural sizes of the ⟨text⟩ in exactly the same way as is done for the commands `\settowidth` etc. on Page 216 of the manual [2].

Note that there is a small difference in the usage of these two methods of accessing text dimensions. After `\settowidth{\txtwd}{Some text}` you can use:

```
\setlength{\parskip}{0.68\textwd}
```

whereas using the more direct access to the width of the text requires the longer form for multiplication, thus:

```
\setlength{\parskip}{\widthof{Some text} * \real{0.68}}
```

TEX discards the stretch and shrink components of glue when glue is multiplied by a real number. So, for example,

```
\setlength{\parskip}{3pt plus 3pt * \real{1.5}}
```

will set the paragraph separation to 4.5pt with no stretch or shrink. (Incidentally, note how spaces can be used to enhance readability.)

When TEX performs arithmetic on integers, any fractional part of the results are discarded. For example,

```
\setcounter{x}{7/2}
\setcounter{y}{3*\real{1.6}}
\setcounter{z}{3*\real{1.7}}
```

will assign the value 3 to the counter `x`, the value 4 to `y`, and the value 5 to `z`. This truncation also applies to *intermediate* results in the sequential computation of a composite expression; thus, the following command

```
\setcounter{x}{3 * \real{1.6} * \real{1.7}}
```

will assign 6 to `x`.

As an example of the use of `\ratio`, consider the problem of scaling a figure to occupy the full width (i.e., `\textwidth`) of the body of a page. Assume that the original dimensions of the figure are given by the dimen (length) variables, `\Xsize` and `\Ysize`. The height of the scaled figure can then be expressed by

```
\setlength{\newYsize}{\Ysize*\ratio{\textwidth}{\Xsize}}
```

3 Formal syntax

The syntax is described by the following set of rules. Note that the definitions of ⟨number⟩, ⟨dimen⟩, ⟨glue⟩, ⟨decimal constant⟩, and ⟨plus or minus⟩ are as in Chapter 24 of The TEXbook [1]; and ⟨text⟩ is LR-mode material, as in the manual [2]. We use *type* as a meta-variable, standing for 'integer', 'dimen', and 'glue'.[3]

⟨*type* expression⟩ ⟶ ⟨*type* term⟩ | ⟨*type* expression⟩⟨plus or minus⟩⟨*type* term⟩

⟨*type* term⟩ ⟶ ⟨*type* factor⟩ | ⟨*type* term⟩⟨multiply or divide⟩⟨integer factor⟩
| ⟨*type* term⟩⟨multiply or divide⟩⟨real number⟩

⟨*type* factor⟩ ⟶ ⟨*type*⟩ | ⟨text dimen factor⟩ | `(`$_{12}$⟨*type* expression⟩`)`$_{12}$

⟨integer⟩ ⟶ ⟨number⟩

⟨text dimen factor⟩ ⟶ ⟨text dimen command⟩`{`⟨text⟩`}`

⟨text dimen command⟩ ⟶ `\widthof` | `\heightof` | `\depthof`

[3] This version of the `calc` package doesn't support evaluation of muglue expressions.

⟨multiply or divide⟩ $\longrightarrow$ `*`$_{12}$ | `/`$_{12}$

⟨real number⟩ $\longrightarrow$ `\ratio{`⟨dimen expression⟩`}{`⟨dimen expression⟩`}`
| `\real{`⟨decimal constant⟩`}`

Note that during most of the parsing of calc expressions, no expansion happens; thus the above syntax must be explicit[4].

References

[1] D. E. KNUTH. *The TEXbook* (Computers & Typesetting Volume A). Addison-Wesley, Reading, Massachusetts, 1986.

[2] L. LAMPORT. *LATEX, A Document Preparation System.* Addison-Wesley, Reading, Massachusetts, Second edition 1994/1985.

[4] Two exceptions to this are: the first token is expanded one-level (thus the whole expression can be put into a macro); wherever a ⟨decimal constant⟩ or ⟨`type`⟩ is expected.