The `basicarith` Package, v1.1

Donald P. Goodman III

October 27, 2014

Abstract

The `basicarith` package provides means for typesetting arithmetic problems, of whatever operations, in a clean and open fashion, suitable for educational texts rather than scholarly works. Digits are spaced out, work (such as carrying, borrowing, and dropping) can be shown visibly, and individual digits can be styled independently.

Contents

1 Introduction

TEX and LATEX are, of course, justly celebrated for their ability to beautifully typeset mathematics. However, there are few utilities available for typesetting the sort of mathematics that we find in primary school textbooks. We can easily typeset incredibly complex equations, and TEX will put them together perfectly; but to typeset an example addition problem, showing our work, is quite difficult. `basicarith` attempts to fill this need by providing macros for typesetting and organizing such problems.

It is notable that basic arithmetic is done in several different styles in various places, though of course the algorithms are typically the same. The author being an American, `basicarith` typesets the problems according to the American custom. Most of these are shared with other English-speaking countries; however, some, particularly long division, will look quite odd to those from other countries. No option for using these other styles is currently available.

2 Basic Macros

`basicarith` offers a few basic macros for formulating problems. These must be divided into two main groups: long division, and other operations. Since the latter are simpler, we'll start with those.

2.1 Non-Division Operations

`\opline`
`\probline`
`\nextpline`
`\soluline`

The fundamental macros for this are `\opline`, `\probline`, `\nextpline`, and `\soluline`. The syntax for each is simple; we'll take them in their order of use.

`\probline {`⟨*width*⟩`} {`⟨*number*⟩`}`

Both these arguments are mandatory. Simply put, `probline` takes first the total width of the problem you are typesetting; this is equivalent to the longest line of the problem. So if your longest line is, say, eight digits long, you put "8" here. The second is the number of the first line of the equation.

`\nextpline {`⟨*number*⟩`}`

`\nextpline` just takes the next number of the equation.

`\opline {`⟨*operator*⟩`} {`⟨*number*⟩`}`

`\opline` takes two mandatory arguments; first, the operator to be typeset, and second, the number of the final line. It also prints a "solution line" underneath itself; this makes it the last line in the problem before we start figuring the answer. Note that, if you want a mathematical symbol for the operator (say, "$\times$"), you have to include the dollar signs yourself. This means that *any* character can be your operator, not only math characters.

`\soluline {`⟨*number*⟩`}`

`\soluline` is designed for the solution of the problem. Functionally this is equivalent to `\nextpline`, but semantically it is clearer. Furthermore, it resets the line counter, so that line styles will be correctly applied in future equations.

There remains only one more basic macro: `\noopline`.

`\noopline {`⟨*number*⟩`}`

`\noopline`

`\noopline` is identical to `\opline` except that it does not print the operator. This is useful for typesetting operations that require one or more intermediate solutions; for example, multiplication by more than one digit. Typically, in the American style the operator is printed only on the final line of the first solution;

subsequent intermediate solutions require a rule above them but no operator on that rule. `\noopline` is what produces this.

These are then combined in the way one might expect. For example:

```
\probline{9}{58193}
\nextpline{54}
\nextpline{4397}
\opline{$+$}{38291374}
\nextpline{38354018}
\noopline{54}
\soluline{38354072}
```

```
        5 8 1 9 3
              5 4
          4 3 9 7
+ 3 8 2 9 1 3 7 4
-----------------
  3 8 3 5 4 0 1 8
              5 4
-----------------
  3 8 3 5 4 0 7 2
```

And that's the essentials of using `basicarith` for non-long division.

2.2 Long Division

Long division is typically more difficult to typeset than other functions; so `basicarith` offers different macros for handling it. The examples below are meaningless (unlike those above, which are actually correct); they are merely meant to show the macros which are available.

`\longdiv` {⟨*length*⟩} {⟨*dividend*⟩} {⟨*divisor*⟩}

`\longdiv` The `\longdiv` macro is, obviously, the most important; it typesets the all-important first line of a long division problem. `\longdiv` takes three arguments, all mandatory; the first is the length of the dividend in number of digits (it uses this to calculate the appropriate length of the line over of the dividend); the second is the dividend itself; and the third is the divisor. It uses a simple close-parenthesis between the dividend and divisor, scaled to 1.2 times the current font size.

`\ldsoluline` {⟨*solution*⟩} {⟨*remainder*⟩}

`\ldsoluline` `\ldsoluline` typesets the answer. This is a bit tricky, because as mentioned before, in American-style long division, the answer is typeset *above* the question, not below it; but we don't know how much space we will need for that answer until we've typeset the first line, containing the dividend, divisor, and close-parenthesis. So `\ldsoluline` should be input *after* the `\longdiv` command but *before* any `\nextldline`s; otherwise, `basicarith` won't be able to place it correctly.

The second argument to `\ldsoluline` is the remainder; if you don't want to typeset a remainder, you still must include this argument, but it can be left blank.

`\nextldline` {⟨*cutoff*⟩} {⟨*number*⟩}

`\nextldline` In the American style of long division, intermediate values are placed *underneath* the dividend, while the solution itself is placed *above* it. `\nextldline` typesets these intermediate values. It takes two arguments, both mandatory. The first is the number of digits of the dividend that are encompassed by this intermediate result (that is, in purely visual terms, the number of digits in the dividend which will not have a number underneath them); the second is the number itself.

Notice that `basicarith` does not keep track of how much it must indent subsequent lines when you're showing your work. This has the benefit that you can skip steps as you see fit; it has the drawback that you have to tell `basicarith` how much each `\nextldline` must skip ahead in order to get the number properly placed.

```
\longdiv{6}{430;932}{983}
\ldsoluline{837;61}{4}
\nextldline{3}{43}
\nextldline{2}{4389}
```

```
          8 3 7;6 1    R 4
       ____________
983 ) 4 3 0;9 3 2
          4 3
        4 3 8 9
```

And this suffices for basic usage of `basicarith`. However, there are many other settings available with the package, which we will address in the next section.

3 Advanced Usage

`basicarith` allows each line of a mathematical problem to be specially styled. For example, say you are trying to draw particular attention to the subtrahend of a given problem:

$$\begin{array}{rr} & 5\;4\;8 \\ - & 2\;7 \\ \hline & 5\;2\;1 \end{array}$$

Or for some reason you want to highlight each line of a subtraction problem differently, perhaps to visually demonstrate which part is which:

$$\begin{array}{rr} & 5\;4\;8 \\ - & 2\;7 \\ \hline & 5\;2\;1 \end{array}$$

`\linestyle` This is accomplished using the `\linestyle` command, which takes two arguments: the number of the line of the problem to be styled (starting with 1 at the top), and the style to be applied to that line.

`\linestyle` {⟨*line number*⟩} {⟨*style*⟩}

The style commands can be anything at all; colors, weights, shapes, or any combination of the above. These commands will typically be confined to the innermost box (particularly, they will not span `\problemboxes`), but if you do need to manually clear all these values back to default (which is no styling at all), issue the command `\clearlinestyles`. `\clearlinestyles`

```
\linestyle{1}{\color{blue}}%
\linestyle{2}{\color{red}}%
\linestyle{3}{\color{green}}%
\probline{4}{548}%
\opline{$-$}{27}%
\soluline{521}%
```

$$\begin{array}{cccc} & 5 & 4 & 8 \\ - & & 2 & 7 \\ \hline & 5 & 2 & 1 \end{array}$$

`\digstyle` Digits can also be individually styled by means of the `\digstyle` command.

`\digstyle` {⟨*column number*⟩} {⟨*style*⟩}

Note that these are really styling *columns*, not merely digits; the style will be applied to every number in that column until either another `\digstyle` for that column number is issued or a `\cleardigitstyles` command is encountered. (This latter is like `\clearlinestyles`, and is pretty self-explanatory.)

`\cleardigitstyles`

```
\linestyle{1}{\color{blue}}%
\linestyle{2}{\color{red}}%
\linestyle{3}{\color{green}}%
\digstyle{3}{\color{black}\itshape}%
\probline{4}{548}%
\opline{$-$}{27}%
\soluline{521}%
```

$$\begin{array}{cccccc} & 2 & 1;\mathcal{T} & 4 & 8 \\ - & & & 2 & 7 \\ \hline & & & \mathit{5};2 & 1 \end{array}$$

Notice also that `\digstyle` commands always take precedence over `\linestyle` commands, no matter what order they are issued in.

Oftentimes we want to show our work explicitly, including our carrying (in the case of addition or multiplication) and our borrowing (in the case of subtraction). We can show this work by using the `\carryline` macro. `\carryline` takes two arguments, the number of digits you'll be putting carries over, and the line with the carries or borrows that you wish to display.

`\carryline`

`\carryline` {⟨*number of digits*⟩} {⟨*carries*⟩}

`\carryline` will respect all `\linestyle` and `\digstyle` commands, and it *does* count as a line for `\linestyle` purposes.

```
\carryline{4}{{3}{\strike{18}17}}
\probline{4}{548}%
\opline{$-$}{29}%
\soluline{519}%
```

$$\begin{array}{cccc} & & {\scriptstyle 3} & {\scriptstyle \cancel{18}17} \\ & 5 & 4 & 8 \\ - & & 2 & 9 \\ \hline & 5 & 1 & 9 \end{array}$$

The first argument must be the same as that in the `\probline` that the carries will be a part of. This is a clunkiness in the interface that I haven't been able to resolve.

`\strike`

Multiple digits can be put in a single carry place by enclosing them in brackets. Further, styling can be applied in a limited way within those brackets; most basic styling, such as italics and boldface, will work, but more complex styling will not. Experimentally, at the very least `soul` and `ulem` styling will not work here. For this reason, `basicarith` provides the `\strike` command, which will strike out a borrow when another borrow is required. It is used as shown in the example above (quite erroneously applied there for the sake of example). `\strike` takes a single argument, the number to be struck out. It is implemented quite naïvly, and will consequently only strike out one or two digit numbers.

`\problembox`

Oftentimes we want to arrange our equations in the page, and `basicarith`'s setup, with each line being a separate `\hbox`, can sometimes make this difficult. We therefore have `\problembox`, which can enclose any number of `basicarith` statements and make it easier to position it on the page. For example, a `basicarith` equation in a `center` environment will not be centered, due to some deep TEX magic that we don't need to go into here. Wrap the equation up in `\problembox`, however, with the `basicarith` equation as the only element, and it will work just fine:

$$\begin{array}{cccc} & & {\scriptstyle 3} & {\scriptstyle \cancel{18}17} \\ & 5 & 4 & 8 \\ - & & 2 & 9 \\ \hline & 5 & 1 & 9 \end{array}$$

4 Configuration Commands

`\b@solverulewidth`

There are a variety of other bits and pieces of `basicarith` that can be customized. The width of solution line is held in the macro `\b@solverulewidth`. As implied by the in its name, this is considered an "internal" command, but with `\makeatletter` and `\makeatother` it can still be reset by a user, to values reasonable or ridiculous. For example, `\b@solverulewidth=2pt` (or `\setlength{\b@solverulewidth}{2pt}`) gives:

$$\begin{array}{ccc} & 4 & 4 \\ + & 4 & 4 \\ \hline \hline \end{array}$$

`\b@longdivlinewidth`

The longdivision equivalent is `\b@longdivlinewidth`; setting it equal to 2pt gives:

$$2\,4\,\big)\overline{3\,2\,9\,8}$$

Both of these values are the TEX-default 0.4pt unless changed by the user.

`\b@maxcols`
`\b@maxrows`

For adminstrative reasons, the sizes of problems are limited both in number of *rows* and in number of *columns*. The parameters controlling these things are, unsurprisingly, `\b@maxcols` and `\b@maxrows`. Both of these are set at twenty to begin with, which ought to be more than enough; but they can be changed if needed.

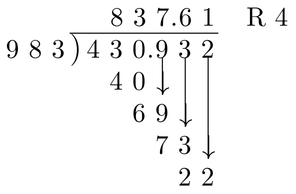
By default, when a remainder is displayed in a long division problem, it is prefixed by "R," in the American custom. If you'd like something different, you can redefine `\b@remaindertext`; its contents will be printed instead.

`\b@remaindertext`

When doing long division, sometimes work is shown in a more explicit way. In addition to writing intermediate solutions beneath the appropriate digits of the dividend, we often show visibly how digits are "dropped" from the dividend into those intermediate problems, by drawing an arrow down from those digits to the appropriate place. With `basicarith`, we do this by issuing the `\showdivwork`, and turn it off with `\noshowdivwork`. Showing work is turned *on* by default.

`\showdivwork`
`\noshowdivwork`

```
\showdivwork
\longdiv{6}{430.932}{983}
\ldsoluline{837.61}{4}
\nextldline{3}{43}
\nextldline{2}{4389}
```



These settings can also be set globally for the package at loading time, with the package options `noshowdivwork` and `showdivwork`. The latter is, of course, the default.

`noshowdivwork`
`showdivwork`

Incidentally, the last of these examples demonstrates another configuration option: `\fractionsymbol`. The author is a dozenalist, and dozenalists customarily use a semicolon as a fractional point; but most people are decimalists, and usually use either a dot or a comma. With `basicarith`, you can use any symbol you want; the default is ";", but by redefining `\fractionsymbol`, you can get anything else. The above was typeset with:

`\fractionsymbol`

```
\def\fractionsymbol{.}
```

The width of the box which encloses each digit is controlled by a macro called `\b@widthofdigits`; redefining this will result in different size digit boxes. The following two lines show this macro set at 3ex and 1ex, respectively.

`\b@widthofdigit`

2 4 ; 5 6

24;56

The default for this setting is 2ex. Note that this is a macro, not a dimen; change it using `\def` or `\renewcommand`, not with `\setlength`.

5 Implementation

We begin by defining the necessary conditional for determining whether long division work should be shown (that is, whether to draw drop arrows), and then defining and processing the package options.

```
\newif\ifshowdivisionwork % switch for drop arrows
\showdivisionworktrue
\DeclareOption{noshowdivwork}{\showdivisionworkfalse}
\DeclareOption{showdivwork}{\showdivisionworktrue}
\ProcessOptions
```

We move on by defining the many dimensions and counters that are required to typeset basic arithmetic problems. Most of these are described afterward by a comment. First, we define those which rarely change; think limitations on the numbers of rows and columns; in the documentation we call these "configuration options."

```
\newdimen\b@solverulewidth % rule under the operator line
\b@solverulewidth=0.4pt
\newcount\b@maxcols % maximum length of a problem
\b@maxcols=20
\newcount\b@maxrows % maximum lines of a problem
\b@maxrows=20
\newdimen\b@longdivlinewidth % width of above
\def\fractionsymbol{;}
```

Next, we define those variables which change frequently; these are the counters that keep track of which row and column we're on and things of that nature.

```
\newdimen\b@topdivline % length of the above
\newdimen\b@totalprobwid % width of widest line of problem
\newdimen\b@digitwid % width of a digit
\def\b@widthofdigit{2ex}
\newcount\b@colnum % row number of problem
\b@colnum=0%
\def\specialdigitstyle{} % style for a given digit
\def\speciallinestyle{} % style for a given digit
\b@longdivlinewidth=0.4pt
\newdimen\b@parenfontsize % size of parenthesis in longdiv
\newcount\b@linenum % row number of problem
\newdimen\b@divisorlen % length of divisor
\newdimen\b@divparenlen % width of the paren in ld
\newdimen\b@ldrowlen % length to add to b@divisorlen
\newdimen\b@fulldivlen % length of divisor + dividend
\b@fulldivlen=0pt
\newcount\b@charcount % number of chars in an argument
\b@charcount=0
\newcount\b@loopi % generic loop counter
\b@loopi=0
\def\b@remaindertext{R} % text for the remainder
\newdimen\b@droparrowlen % drop arrow length
\b@droparrowlen=0pt
```

Now we define the macros that are used for counting the number of characters in various strings; these are adapted from macros by "Florent" at `tex-and-stuff.blogspot.com`.

```
\def\gobblechar{\let\char= }
```

```
\def\assignthencheck{\afterassignment\checknil\gobblechar}
\def\countunlessnil{%
\ifx\char\nil \let\next=\relax%
\else%
\let\next=\auxcountchar%
\advance\b@charcount by1%
\fi%
\ifx\char;\advance\b@charcount by-1\fi%
\next%
}%
\def\auxcountchar{%
\afterassignment\countunlessnil\gobblechar%
}%
\def\countchar#1{\def\xx{#1}\b@charcount=0 \expandafter\auxcountchar\xx\nil}
```

Now we define some macros for splitting a string into individual characters and putting them in boxes; these are adapted from David Carlisle's answer on `tex.stackexchange.com`, question 57598.

```
\def\b@expandloop#1{%
\hbox{%
\b@xloop#1\relax
}%
}
\def\b@xloop#1{%
\if#1\fractionsymbol\else\advance\b@colnum by-1\fi%
\ifx\relax#1%
\else%
\if#1\fractionsymbol%
\rlap{\hbox to0pt{\hss#1\hss}}%
\else%
\hbox to\b@digitwid{\hfil{%
\csname speciallinestyle\romannumeral\b@linenum\endcsname%
\csname specialdigitstyle\romannumeral\b@colnum\endcsname%
#1%
}\hfil}%
\fi%
\expandafter\b@xloop%
\fi%
}
\def\b@spaceout#1{%
\countchar{#1}%
\b@colnum=\b@charcount%
\advance\b@colnum by1%
\b@expandloop{#1}%
}%
```

Now we define our macros for non-long-division problems.

```
\def\probline#1#2{%
\advance\b@linenum by1%
\b@digitwid=\b@widthofdigit%
```

```
\b@totalprobwid=\b@digitwid%
\multiply\b@totalprobwid by#1%
\hbox to\b@totalprobwid{%
\hfil\b@spaceout{#2}%
}%
}%
\def\opline#1#2{%
\advance\b@linenum by1%
\hbox{%
\hbox to\b@digitwid{\hfil#1}%
\advance\b@totalprobwid by-\b@digitwid%
\hbox to\b@totalprobwid{%
\hfil\b@spaceout{#2}%
}%
}%
\vskip0.5ex%
\hrule width\b@totalprobwid height\b@solverulewidth%
\vskip0.5ex%
}%
\def\noopline#1{%
\opline{}{#1}%
}%
\def\nextpline#1{%
\advance\b@linenum by1%
\hbox{%
\hbox to\b@totalprobwid{%
\hfil\b@spaceout{#1}%
}%
}%
}%
\def\soluline#1{%
\advance\b@linenum by1%
\hbox{%
\hbox to\b@totalprobwid{%
\hfil\b@spaceout{#1}%
}%
}%
\b@linenum=0%
}%
\def\carryline#1#2{%
{%
\advance\b@linenum by1%
\b@digitwid=\b@widthofdigit%
\b@totalprobwid=\b@digitwid%
\multiply\b@totalprobwid by#1%
\footnotesize%
\hbox to\b@totalprobwid{%
\hfil\b@spaceout{#2}%
}%
\hrule width\b@totalprobwid height0pt%
\vskip0.4em%
}%
}
```

Now we proceed to define the macros for long division.

```
\def\longdiv#1#2#3{%
\advance\b@linenum by1%
\vskip\baselineskip%
\b@digitwid=\b@widthofdigit%
\b@topdivline=\b@digitwid%
\settowidth{\b@divisorlen}{\b@spaceout{#3}}
\b@parenfontsize=\f@size pt%
\multiply\b@parenfontsize by12%
\divide\b@parenfontsize by10%
\settowidth{\b@divparenlen}{%
\fontsize{\b@parenfontsize}{\b@parenfontsize}\selectfont)}%
\multiply\b@topdivline by#1%
\advance\b@topdivline by0.5\b@digitwid%
\vskip0.5ex%
\vbox{%
\hbox{%
\hskip\b@divisorlen%
\vrule width\b@topdivline height\b@longdivlinewidth%
}%
\nointerlineskip%
\hbox{%
\b@spaceout{#3}%
\hfil{\fontsize{\b@parenfontsize}{\b@parenfontsize}\selectfont)}%
\b@spaceout{#2}%
}%
}%
\advance\b@divisorlen by\b@divparenlen%
}%
\def\ldsoluline#1#2{%
\advance\b@fulldivlen by\b@divisorlen%
\advance\b@fulldivlen by\b@topdivline%
\advance\b@fulldivlen by-\b@digitwid%
\advance\b@fulldivlen by\b@divparenlen%
\vskip-2\baselineskip%
\hbox to\b@fulldivlen{%
\hfil%
\b@spaceout{#1}%
\if#2\relax\else\rlap{\hskip1em \b@remaindertext{ }#2}\fi%
}%
\vskip\baselineskip%
}%
```

This is an interesting little trick which gets the width of a box; we use this to determine the placement of the drop arrows when we're showing our long division work.

```
\newdimen\b@droparrowwidth
\def\getdroparrowwidth{%
\setbox\@tempboxa\hbox{$\downarrow$}%
\b@droparrowwidth=\wd\@tempboxa%
}%
```

Now we get back to long division macros (yes, these do take a rather long time). We start with the macro for subsequent long division lines.

```
\def\nextldline#1#2{%
\advance\b@linenum by1%
\b@ldrowlen=\b@digitwid%
\multiply\b@ldrowlen by#1%
\hbox{%
\hskip\b@divisorlen%
\hskip\b@ldrowlen%
\b@spaceout{#2}%
\b@droparrowlen=\baselineskip%
\ifshowdivisionwork%
\ifnum\b@linenum>2%
\getdroparrowwidth%
\multiply\b@droparrowlen by\b@linenum%
\advance\b@droparrowlen by-2\baselineskip%
\hskip-0.5\b@digitwid%
\vtop{\vskip-\baselineskip\vskip-\b@droparrowlen%
\rlap{%
\vrule width0.4pt height\b@droparrowlen%
\hskip-0.5\b@droparrowwidth{$\downarrow$}%
}
}%
\fi%
\fi%
}%
}%
```

Now we move on to the styling macros; this requires a lot of looping and other weirdities (weirdities for TEX programming, anyway). First we style lines; then we style columns.

```
\def\linestyle#1#2{%
\b@loopi=0%
\loop\ifnum\the\b@loopi<\the\b@maxrows%
\advance\b@loopi by1%
\ifnum#1=\the\b@loopi
\expandafter\def\csname speciallinestyle\romannumeral\b@loopi\endcsname{#2}%
\fi
\repeat
}%
\def\digstyle#1#2{%
\b@loopi=0%
\loop\ifnum\the\b@loopi<\the\b@maxcols%
\advance\b@loopi by1%
\ifnum#1=\the\b@loopi
\expandafter\def\csname specialdigitstyle\romannumeral\b@loopi\endcsname{#2}%
\fi
\repeat
}%
```

Now we get our commands to clear the styling; again, first lines, then columns.

```
\def\clearlinestyles{%
\b@loopi=0%
\loop\ifnum\the\b@loopi<\the\b@maxrows%
\advance\b@loopi by1%
\expandafter\def\csname speciallinestyle\romannumeral\b@loopi\endcsname{}%
\repeat
}%
\def\cleardigitstyles{%
\b@loopi=0%
\loop\ifnum\the\b@loopi<\the\b@maxcols%
\advance\b@loopi by1%
\expandafter\def\csname specialdigitstyle\romannumeral\b@loopi\endcsname{}%
\repeat
}%
```

A few miscellanies; a box for binding together problems, so that they can be positioned on the page more easily; a macro for doing strikethroughts, quite useful in the carries; and macros for showing or not showing our long division work.

```
\def\problembox#1{%
\leavevmode\vbox{#1}%
}%
\def\strike#1{%
{\rlap{\bf---}#1}%
}
\def\showdivwork{%
\showdivisionworktrue%
}
\def\noshowdivwork{%
\showdivisionworkfalse%
}
```

Now, finally, we clear all the digit styles (so that they at least exist), and then we're done. Thanks for reading; happy TEXing!

```
\cleardigitstyles
```

Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used.