"Hey — it works!"

Jeremy Gibbons
University of Auckland
`jeremy@cs.aukuni.ac.nz`

Welcome again to *"Hey — it works!"*. This column is dedicated to elegant, interesting, or just plain useful techniques in TeX and LaTeX—in the style of Jon Bentley's *Programming Pearls* or Richard Bird's *Functional Pearls*, for the 'computists' among you. If you have any such items, no matter how trivial, please pass them on to me at the above address.

In this issue we have three pieces: one by Donald Arseneau on interleaving under- and overbraces; one of my own on constructing new symbols from existing ones; and one by Navindra Gambhir on constructing tables, illustrating the use of `\afterassignment`.

Non-nested braces

Donald Arseneau
Tri-University Meson Facility
`asnd@erich.triumf.ca`

Sometimes one would like to interleave braces labelling parts of an equation:

$$a+b+\overbrace{c+d+e+f}^{x}+\overbrace{g+h+i+j}^{y}+k+l+m=\pi r^2 \qquad \text{with } \underbrace{e+f+g+h}_{z}$$

This is something that `\overbrace` and `\underbrace` cannot do. The solution I have chosen (for both plain TeX and LaTeX) is based on visual formatting into columns, like an array or table.

First, (on paper or imagination) write the formula and divide it into segments at the tip of each brace. In our example there are seven segments:

$$\begin{array}{|c|c|c|c|c|c|c|l}
a+b+ & c+d+ & e+f & + & g+h & +i+j & +k+l+m & =\pi r^2 \\
1 & 2 & 3 & 4 & 5 & 6 & 7 &
\end{array}$$

although segments 1 and 7 could be omitted. We will type the main line of the formula with `&` inserted between each segment:

```
a + b +&c + d +&e + f&+&g + h&+ i + j&+ k + l + m
```

Each segment then becomes a column in an alignment, and each brace may span multiple columns. The idea is best illustrated by a LaTeX array:

```
\begin{array}[c]{ccccccc}
  & \multicolumn{2}{c}{x}& & \multicolumn{2}{c}{y}\\
  & \multicolumn{2}{c}{\downbracefill}&
                            & \multicolumn{2}{c}{\downbracefill}\\
  a + b +&c + d +&e + f&+&g + h&+ i + j&+ k + l + m\\
  & & \multicolumn{3}{c}{\upbracefill}\\
  & & \multicolumn{3}{c}{z}
\end{array} = \pi r^2
```

This doesn't work well, having bad braces with bad vertical and horizontal spacing. We should instead use TeX's primitive `\halign` command, and measure the size of boxes to get the correct vertical positioning of the formula's main line. Rather than type the very messy TeX commands directly with each use, it is better to make a new command:

`\overunderbraces{`*upper_braces*`}{`*main_formula*`}{`*lower_braces*`}`

We already know how to write the main line of the formula (by segmenting the formula and inserting `&` between segments), and we will also use `&` between braces. Each labelled brace will be specified in the format:

`\br{`*n*`}{`*label*`}`

where n is the number of segments spanned by the brace. Each top brace of our example spans two segments (with an empty segment between them) and the bottom brace spans three segments, so they are specified by:

top braces:	`&\br{2}{x}&  &\br{2}{y}`
bottom braces:	`& &\br{3}{z}`

Having determined all three parameters to use with `\overunderbraces`, the complete command is

```
\overunderbraces{&\br{2}{x}& &\br{2}{y}}%
   {a + b +&c + d +&e + f&+&g + h&+ i + j&+ k + l + m}%
   {& &\br{3}{z}}
 = \pi r^2
```

which produces the equation at the start of this article.

Here is the definition of `\overunderbraces`. It may look long, but it has some repetition in order to measure the height of the top half of the formula. There are also the auxiliary macros `\brAup`, `\brAdown`, and `\brAlabel` that allow the same `\br` command to produce up-braces, down-braces, and centered labels at the appropriate places.

```
% \overunderbraces{upper_braces}{main_formula}{lower_braces}
%
\def\overunderbraces #1#2#3{{%
 \baselineskip=0pt \lineskip=4pt \lineskiplimit=4pt
 \displaystyle % deliberately generate error if not in math mode!
% first, do the top half of the alignment in a save-box
 \setbox0=\vbox{\ialign{&\hfil${}##{}$\hfil\cr
   \global\let\br\brAlabel #1\cr % upper labels
   \global\let\br\brAdown #1\cr  % upper braces
   #2\cr % main line of the formula
 }}% finished partial alignment and \vbox.
 \dimen0=-\ht0 %   Measure height of partial alignment --
% ... it is the height we want for the whole.
% Now do the whole alignment (notice the repetition from above)
 \setbox0=\vbox{\ialign{&\hfil${}##{}$\hfil\cr
   \global\let\br\brAlabel #1\cr % upper labels
   \global\let\br\brAdown #1\cr  % upper braces
   #2\cr % main line of the formula
   \global\let\br\brAup #3\cr    % lower braces
   \global\let\br\brAlabel #3\cr % lower labels
 }}% finished whole alignment and \vbox.
 \advance\dimen0 by\ht0 % calc. the necessary lowering
 \lower\dimen0\hbox{\box0}% shift to get the desired height
}}

% Three aliases for \br.
% #1=number of spanned columns,  #2=label
\def\brAup#1#2{\multispan{#1}\upbracefill}
\def\brAdown#1#2{\multispan{#1}\downbracefill}
\def\brAlabel#1#2{\multispan{#1}\hidewidth $#2$\hidewidth}
```

If you check the electronic version of this newsletter, you will find the text of `oubraces.sty` which contains these definitions, except for some streamlining with 'internal' commands such as `\dimen@` used for `\dimen0`.

Overlaying symbols

Jeremy Gibbons

My particular branch of computer science often requires me to construct new mathematical symbols. Sometimes I do this with METAFONT, but if I want to be able to distribute my TEX or `dvi` files, I need to construct the new symbols out of existing ones. One technique for doing this is simply to 'overlay' two existing symbols; for this purpose, I wrote the style file `overlay.sty`.

The basic usage is to overlay two symbols, centre-aligning them with respect to each other and producing a symbol as wide as the wider component. Thus, `$\overlay{\triangleleft}{-\!\!-}$` yields ‘$\triangleleft\!\!\!-$’. Two variations on this theme involve left- and right-alignment instead of centre-alignment; using `\loverlay` or `\roverlay` (great names!) yields ‘$\triangleleft\!\!\!\!-$’ or ‘$-\!\!\!\!\triangleleft$’, respectively.

The macro copes with changing ‘styles’, so that symbols get smaller in subscripts and so on. Thus, `$X_{\overlay{\triangleleft}{-\!\!-}}$` yields ‘$X_{\triangleleft\!\!\!-}$’. Within the arguments to `\overlay`, the macro `\overlaystyle` expands out to the style, so the overlaid components can involve boxes and still respect changing styles; for example, the (rather contrived) definition

```
\def\symbol{\overlay{-}{\raise.4ex\hbox{$\overlaystyle\bullet$}}}
```

yields ‘$\overset{\bullet}{-}$’ in text style and ‘$X_{\overset{\bullet}{-}}$’ in script style—note the different `\bullets`[1].

The macro is even sufficiently robust to cope with nested overlays; the code

```
$\overlay{\lower.6ex\hbox{$\overlaystyle\smile$}}
         {\raise.6ex\hbox{$\overlaystyle
            {\roverlay{\circ}{\cdot}}{\roverlay{\circ}{\cdot}}$}}
 \;\;
 \overlay{\lower.6ex\hbox{$\overlaystyle\smile$}}
         {\raise.6ex\hbox{$\overlaystyle
            {\loverlay{\circ}{\cdot}}{\loverlay{\circ}{\cdot}}$}}$
```

yields two friendly smileys: $\overset{\circ\circ}{\smile}\;\;\overset{\circ\circ}{\smile}$

```
% overlay.sty - overlay and align two symbols, respecting styles

\def\loverlay#1#2{\mathpalette\@overlay{{#1}{#2}{}{\hfil}}}
\def\overlay#1#2{\mathpalette\@overlay{{#1}{#2}{\hfil}{\hfil}}}
\def\roverlay#1#2{\mathpalette\@overlay{{#1}{#2}{\hfil}{}}}
   % calls to \@overlay look like
   %    \@overlay\textstyle{{x}{y}{\hfil}{\hfil}}
\def\@overlay#1#2{\@@overlay#1#2}
   % strip brackets from 2nd arg, to get
   %    \@@overlay\textstyle{x}{y}{\hfil}{\hfil}
\def\@@overlay#1#2#3#4#5{{%
   \def\overlaystyle{#1}%
   \setbox0=\hbox{\m@th$\overlaystyle#2$}%
   \setbox1=\hbox{\m@th$\overlaystyle#3$}%
   \ifdim \wd0<\wd1 \setbox2=\box1 \setbox1=\box0 \setbox0=\box2\fi
      % \box0 is now the wider box
   \rlap{\hbox to\wd0{#4\box1\relax#5}}\box0}}
```

[1] Actually, this definition is not quite right: an `ex` doesn't get smaller in script styles. The best fix I know of involves defining a macro to replace the `ex`, choosing a dimension based on `\overlaystyle`. Unfortunately, I don't think `mus` can be used for vertical movement.

Making Tables by Iteration

Navindra Gambhir
Cornell University, Ithaca NY
gambhir@cs.cornell.edu

We sometimes use tables similar to the one shown on the right when grading homework or exams for a class. These tables differ from one problem set to another only in the values in the last column and the total number of rows. Therefore it is possible to build these tables by simply listing out the numbers that go in the last column. For example, the table shown here was typeset by typing '`\maxpoints 35 5 60 -1`'. In general a table for n questions with maximum possible points p_1, p_2, ..., p_n is produced by typing

Name	______________________		
Question 1	____________	out of	35
Question 2	____________	out of	5
Question 3	____________	out of	60
Total	____________	out of	100

`\maxpoints` p_1 p_2 ... p_n ⟨negative number⟩

The macro `\maxpoints`, given below, uses `\afterassignment` trickery to read the numbers one at a time. It generates a line specification for each non-negative number in the list. When it finds the negative number, it wraps up the table by generating the last ("Total") line.

```
\newcount\exno \newcount\expoints \newcount\totalexpoints
\def\doublestrut{\vrule height20.5pt depth3.5pt width0pt \relax}
% each line of the table is made the proper height by including
% \doublestrut in the preamble; interline spacing is turned off
\def\maxpoints{\noindent $$\offinterlineskip
  \halign\bgroup\doublestrut##\hfil& % preamble of the table...
      \ \ \leaders\hrule\hskip6pc \ \ out of \hfil##\cr
    Name \ \hrulefill\span\omit\cr % first line of the table
    \global\exno=0 \global\totalexpoints=0 \getnum}
% if the next number in the list is negative, wrap up the table;
% otherwise generate the next line and continue
\def\getnum{\afterassignment\g@tnum\global\expoints}
\def\g@tnum{\ifnum\expoints<0 \let\next=\finishtable
              \else \let\next=\nextline\fi \next}
\def\nextline{\global\advance\exno by 1 Question \the\exno&
  \global\advance\totalexpoints by\expoints \the\expoints\cr
  \getnum}
\def\finishtable{Total&\the\totalexpoints\cr\egroup$$}
```