Pearls of TeX programming

The title of the BachoTeX 2005 conference was “The Art of TeX Programming.” TAOTP for short, therefore the idea of a “Pearls of TeX Programming” session arose. Bogusław Jackowski came up with the session motto: “Behold” — Bhaskara (see, e.g., http://www.aurora.edu/mathematics/bhaskara.htm).

The idea was to invite TeXies known to be TeXperts, TeX Masters or perhaps even TeX Grandmasters1 to contribute.

The call stated what was wanted:

• a short TeX, METAFONT, or METAPOST macro or macros (preferably a few lines)
• results should be virtually useful yet not obvious
• easy to explain: 10 minutes at most

Prospective contributors were asked to kindly provide the source of a macro or macros and a display or short description of the result, the size of it to be altogether not more than one A4 page, preferably — half of an A4.

We also stated that this is not a contest and that contributions were requested even from authors who are unable to attend the conference. In such a case the author was free either to elect one of the participants to present his work or “leave the proof to the gentle reader” aka “Behold”. The latter can be done anyway...

As can be seen from the examples, we did not strictly adhere to the stated program/macro limitations, with the notable exception being Frank Mittelbach’s contribution. The result is here for the gentle reader to digest and profit from.

We intend to continue the TAOTP initiative at future BachoTeX conferences: TeX has so much more up its sleeves ... A web display, similar in spirit to the “TeX Showcase” maintained by Gerben Wierda (at http://tug.org/texshowcase), is also being considered for the future.

Editor’s note: This article is reprinted with permission from the proceedings of BachoTeX 2005: Biuletyn Polskiej Grupy Użytkowników Systemu TeX, Zeszyt 22, May 2005, T. Przechlewski, B. Lichoński, S. Wawrykiewicz, P. Bolek, eds. Reprinted with permission.

1 Of course the blame for a failure to contact somebody fitting this description should be put at the doorstep of the conference organizers.

barbara beeton

New symbols from old

Sometimes one needs a symbol that can’t be found in any font, but that is either a rotation or a reflection of a symbol that is available. The graphicx package to the rescue!

\newcommand{\reflectit}[1]{\reflectbox{\ensuremath#1}}
\newcommand{\turnover}[1]{\rotatebox{180}{\ensuremath#1}}
\newcommand{\turnne}[1]{\rotatebox{45}{\ensuremath#1}}
\newcommand{\turnnw}[1]{\rotatebox{135}{\ensuremath#1}}
\newcommand{\turnsw}[1]{\rotatebox{225}{\ensuremath#1}}
\newcommand{\turnse}[1]{\rotatebox{315}{\ensuremath#1}}

\leq \equiv \geq : \equiv ; \Rightarrow : \not\ll \not\gg; \sim : \sim$

When you define new names for such symbols, it’s a good idea to specify the class (\mathord, \mathbin, etc.) in the definition so you get the correct spacing when they’re used.
Martin Schröder

Colour separation in pdFT\TeX

\newcommand*{\AC@addColor}[5]{%
  \immediate\pdfobj stream
  attr {
    /FunctionType 4
    /Domain [0.0 1.0]
    /Range [0.0 1.0 0.0 1.0 0.0 1.0 0.0 1.0]
  }
  \{ dup 72 mul exch
dup 73 mul exch
dup 74 mul exch
75 mul \} \%
}\edef\AC@ColorFunctionObj{\the\pdflastobj}%
\immediate\pdfobj {[/Separation /?1
  /DeviceCMYK
  \AC@ColorFunctionObj 0 R ]}%
\begingroup
\toks0=\expandafter{\AC@colorhook}\
def\x{\
  \endgroup
  /?1\space\the\pdflastobj\space 0 R %
}\
\x}
\edef\AC@expand{\global\pdfpageresources {\
  /ColorSpace << \AC@colorhook >>
}\
\AC@expand

David Carlisle (proposed by Frank Mittelbach)

Guess what...

% later
\edef\AC@expand{\global\pdfpageresources {\
  /ColorSpace << \AC@colorhook >>
}\
\AC@expand

This pearl is saved for you at http://www.gust.org.pl/BachoTEX/2005/pearls/
Don't try to copy it from this paper.
Karl Berry
Forcing a page or column break in the middle of a paragraph.
\parfillskip=0pt\par\vfill\penalty-10000{\everypar={}\noindent}

Taco Hoekwater
Die Hard
Here is a very short macro that immediately kills off a \TeX run, regardless of the current state of the \TeX engine, and issuing a \textit{fatal error} message before it does so.
\def\die#1{
{\immediate\write16{#1}}
\batchmode
\input junkfilethatdoesntexist }

Petr Olšák
\texttt{\expandafter\endcsname} trick.

It is better to write
\expandafter\begin{csname}
\expandafter\endcsname #1\expandafter\endcsname #2\expandafter\endcsname
than
\expandafter\begin{csname}
\endcsname #1\expandafter\endcsname #2\expandafter\endcsname

Petr Olšák
Testing whether two characters form a ligature.

\newif\ifligature
\def\testligature#1#2{\setbox0=\hbox{$\mathchar`#1 \mathrel\mathchar`#2$}
\ifdim\wd0>500pt \ligaturefalse \else \ligaturetrue \fi}

David Kastrup
Comparing two strings known to consist only of characters.
\def\strequal#1{
\number\strequalstart{}{}#1\relax
\def\strequalstart#1#2#3{\if#3#1\relax\strequalstop\fi
\strequalstart{\if#3#1}{#2}{#3}\fi}
\def\strequalstop\fi\strequalstart#1#2#3{\fi#1#3\relax#213 }
\if\strequal{junk}{#1} will be true for #1 being “junk”, and false otherwise.
David Kastrup

Sorting words by length.

“Finnegans Wake” by James Joyce is a book that is not easily comprehensible. \TeX\ can systematize the approach to the text by confronting the reader with the longest, and consequently hardest, words last.

\begin{verbatim}
\def\sorttext#1{\setbox0\vbox{\language255\hsize=0pt\hfuzz\maxdimen\parfillskip0pt\noindent#1\par}\sortvlist\unpack}\unvbox0 }
\def\sortvlist{{\unskip\unpenalty \setbox0\lastbox
\ifvoid0\noindent\else\setbox0\hbox{\unhbox0\ }\sortvlist\sortin\fi}}
\def\sortin{\setbox2\lastbox\ifdim\wd2>\wd0{\sortin}\fi\box2\box0}
\def\unpack{{\setbox0\lastbox\ifvoid0\indent\else\unpack\unhbox0\fi}}
\sorttext{riverrun, past Eve and Adam’s, ... linsfirst loved livvy.}
\end{verbatim}

Frank Mittelbach

\texttt{\looseness} not so loose.

This paragraph was set twice in a two column multicol environment. The first time it was set without any special adjustments, the second time we used -1 as the value for the \texttt{\looseness} parameter. Can you explain why the two paragraphs are differently broken into lines even though clearly the use of the parameter \texttt{\looseness} couldn’t shorten the paragraph at all?

This paragraph was set twice in a two column multicol environment. The first time it was set without any special adjustments, the second time we used -1 as the value for the \texttt{\looseness} parameter. Can you explain why the two paragraphs are differently broken into lines even though clearly the use of the parameter \texttt{\looseness} couldn’t shorten the paragraph at all?

Answer: When \texttt{\looseness} gets a non-zero value, \TeX\ will always run through all paragraph passes (i.e., breaking without hyphenation, with hyphenation and (if \texttt{\emergencystretch} is non-zero as it is inside multicol) through the emergency-pass. But adding \texttt{\emergencystretch} to every line means that the line breaks chosen in the first paragraph may fall in different fitting classes so that at different places \texttt{\adjmerits} are charged, thus making the original solution less attractive.

In fact the situation could even be worse: if a long paragraph can be broken into lines by just using \texttt{\pretolerance}, then a setting of \texttt{\looseness} to +1 might in fact result in a paragraph with one line less—all that is required is that by breaking it using \texttt{\tolerance} we would get a default line count that would be 2 lines less than in the case with \texttt{\pretolerance} (a real life example is left to the reader).
Philip Taylor
The Iterator

In general-purpose \TeX{} programming (as opposed to typesetting with \TeX{}), one of the most commonly needed techniques is the ability to iterate over an unknown number of parameters. If the number is known to be nine or less in advance, \TeX{} is quite capable of doing all that is necessary with only a little help from the user. However, if the number of parameters may exceed ten, then a rather more devious approach will be required.

\begin{verbatim}
def \forall #1#2\do #3{\relax #2#3 \ifx \relax #2\else \forall #2\do #3\fi}
\end{verbatim}

Sample usage:

\begin{verbatim}
def \debug #1{\message {[#1]}#1 }
def \forall 1234abcd{ef}{ghi}etc...\do {\debug}
\end{verbatim}

David Kastrup
Iterating with roman numerals.

Appendix D in \textit{The \TeX{} Book} has the task of defining \verb+\ast+ as a macro containing \verb+\number+n copies of an asterisk. The solutions in \textit{The \TeX{} Book} are not really fun. Here is one that is all sorts of fun, efficient and simple:

\begin{verbatim}
def\ast#1{\if#1m*\expandafter\ast\fi}
def\ast{\expandafter\ast\romannumeral\number
000\relax}
\end{verbatim}

Now for something more general: we want a macro \verb+\replicate+ that gets a number in its first argument and arbitrary tokens in its second argument and expands to the given number of repeated token strings.

It is surprisingly hard to pass both the shrinking string of \texttt{m} as well as the argument to be repeated in a useful way into the expanding first macro, and the reader is advised to try it. What I came up with was

\begin{verbatim}
\long\def\gobble#1{}\long\def\xi#1#2{\if#2m#1\expandafter\xi\else\expandafter\gobble\fi#1}\long\def\xii#1#2{\xi#1#2#1\relax}\long\def\replicate#1{\expandafter\xii\romannumeral\number\number#1 000\relax}\def\rec#1{\csname rn#1\rec\endcsname}{\long\def\rn#1{\csname rn#1\endcsname}}\long\def\replicate#1{\csname rn\expandafter\rec\endcsname}{\csname \romannumeral\number\number#1 000\endcsname}}\long\def\replicate#1#2{\ifnum#1>0 #2% \expandafter\replicate\expandafter{\number{\numexpr#1-1}\{#2\}}fi}
\end{verbatim}

Of course, if we leave the area of \TeX{} compatibility and take a look at what we can do with \texttt{-\TeX{}}, we arrive at the boring

\begin{verbatim}
def\replicate#1#2{\ifnum#1>0 #2% \expandafter\replicate\expandafter{\number{\numexpr#1-1}\{#2\}}fi}
\end{verbatim}
Krzysztof Leszczyński

\csequence stack

Often I need to save a few macros but I don’t want to \begingroup and \global-ly define those I want to keep after \endgroup. Here is a simple stack:

- \newcsstack \stackname — define a new stack
- \pushcs \stackname \cs — push a control sequence
- \popcs \stackname \cs — pop a control sequence
- \topcs \stackname \cs — equivalent to \popcs...\pushcs

\def \gobble#1{} % this macro is usually defined somewhere
\def \stackcs#1{\csname \ifnum\escapechar>-1
   \expandafter \expandafter \expandafter \gobble
   \expandafter \fi \string #1::\number#1\endcsname}
% temporarily un-outer newcount to define newcsstack
\let \topcs = \newcount \let \newcount = \relax
\def \newcsstack #1{\newcount #1\global#1=0\pushcs#1\relax}
\let \newcount = \topcs % restore \newcount
\def \pushcs#1#2{\global \advance#1 1
   \global \expandafter \expandafter \expandafter \let \stackcs{#1}= #2}
\def \topcs#1#2{\let #2\stackcs{#1}}
\def \popcs#1#2{\topcs#1#2
   \global \expandafter \expandafter \expandafter \let \stackcs{#1}\relax \global \advance #1-1 }

The above example doesn’t save parameter values, only the meaning is saved but see below.

Bogusław Jackowski

Locally changes parameter values.

Macro \local changes a value of a parameter locally (for one paragraph).
\let\restoreparams\empty
\def\local#1{% e.g., ‘‘\local\hfuzz=2pt \ldots \par’’
   \ifx\restoreparams\empty
      \let\oripar\par
   \def\par{\oripar \restoreparams \let\par\oripar \let\restoreparams\empty}%
   \fi
   \edef\restoreparams{\restoreparams#1\the#1} %
   #1}
Bogusław Jackowski
Extra Béziers

The macro `extrapolate` computes a “superpath” (as opposed to “subpath”) for a single Bézier segment in such a way that the following identity holds (for $0 \leq t_1 \leq t_2 \leq 1$):

$$\text{subpath}(t_1, t_2) \text{ of (extrapolate}(t_1, t_2) \text{ of } b) = b$$

Below, there are results of the command `extrapolate((3., 7.) of p)` for three similarly defined paths. The black line denotes the source path, the gray one—its extrapolation.

$$p = (0, 0) \{\text{right}\} \ldots \{\up\}(s, s);$$

$$p = (0, 0) \{\text{right}\} \ldots \text{tension} \, 30 \ldots \{\up\}(s, s);$$

$$p = (0, 0) \{\text{right}\} \ldots \text{tension} \, 75 \ldots \{\up\}(s, s);$$

Exercise 1. What happens if the relation $0 \leq t_1 \leq t_2 \leq 1$ is not fulfilled? (Hint: there are a few possible cases.)

Zadanie 1. Co by się stało, gdyby warunek $0 \leq t_1 \leq t_2 \leq 1$ nie był spełniony? (Wskazówka: możliwych jest kilka różnych przypadków.)

Exercise 2. True or false:

$$\text{point 1 of (extrapolate}(t_a, t) \text{ of } b) = \text{point 1 of (extrapolate}(t_b, t) \text{ of } b)$$

for $t_a \leftrightarrow t_b$

Zadanie 2. Prawda czy falso:

Exercise 3. Try to imagine the result of the extrapolation for such weird (yet trivial) paths as:

$$(0, 0) \ldots \text{controls}(0, 0) \text{ and } (100, 0) \ldots (100, 0)$$

or

$$(0, 0) \ldots \text{controls}(100, 0) \text{ and } (0, 0) \ldots (100, 0)$$

Zadanie 3. Spróbuj przewidzieć wynik ekstrapolacji dla tak dziwnych (chociaż trybnych) ścieżek jak:

```latex
\textbf{vardef} \textit{extrapolate} expr \textit{t of } b =% \textit{t pair, } b \text{ Bézier segment}
\textbf{clearxy;}
\textit{Casteljau}(\textit{xpart}(t)) = \textit{point 0 of } b;
\textit{Casteljau}(\frac{1}{3} [\textit{xpart}(t), \textit{ypart}(t)]) = \textit{point } \frac{1}{3} \text{ of } b;
\textit{Casteljau}(\frac{2}{3} [\textit{xpart}(t), \textit{ypart}(t)]) = \textit{point } \frac{2}{3} \text{ of } b;
\textit{Casteljau}(\textit{ypart}(t)) = \textit{point 1 of } b;
z_0 \ldots \text{controls } z_1 \text{ and } z_2 \ldots z_3
\textbf{enddef;}
%
def \textit{Casteljau}(expr t) =
t[t[z_0, z_1], t[z_1, z_2], t[z_1, z_2], t[z_2, z_3]]
\textbf{enddef;}
```
Bernd Raichle
Plain TeX’s accent macros revisited.

Sample output using Plain TeX’s accent macros.

Here is the output when Plain TeX’s accent macros \AA, \c, and \b are used with various glyphs from different upright and slanted fonts.

\begin{verbatim}
\def\AA{{\dimen@ 1ex\%}
{\setbox\z@\hbox{A}\dimen@\ht\z@ \advance\dimen@-.35ex\%
 \fontdimen5\font\dimen@}
\accent'27\fontdimen5\font\dimen@ A}}
\def\c#1{{\dimen@ 1ex\%
{\setbox\z@\hbox{#1}\dimen@\ht\z@ \advance\dimen@\dp\z@
 \fontdimen5\font\dimen@}\accent24\fontdimen5\font\dimen@ #1}}
\def\b#1{{\dimen@ 1ex\%
{\setbox\z@\hbox{\char22}\dimen@\ht\z@ \advance\dimen@.25ex\%
 \setbox\z@\hbox{#1}\advance\dimen@\ht\z@ \advance\dimen@\dp\z@
 \global\dimen@i\dp\z@ \global\advance\dimen@i .45ex\%
 \fontdimen5\font\dimen@\accent22\fontdimen5\font\dimen@ #1\%
 \dp\z@\dimen@i \box\z@}}
\end{verbatim}

Revised macros using the \accent primitive.

The following re-implementation does not use \halign but the \accent primitive to position the accent glyph.

\begin{verbatim}
\def\AA{{\dimen@ 1ex\%
{\setbox\z@\hbox{A}\dimen@\ht\z@ \advance\dimen@-.35ex\%
 \fontdimen5\font\dimen@ \accent'27\fontdimen5\font\dimen@ A}}
\def\c#1{{\dimen@ 1ex\%
{\setbox\z@\hbox{#1}\dimen@\ht\z@ \advance\dimen@\dp\z@
 \fontdimen5\font\dimen@ \accent24\fontdimen5\font\dimen@ #1}}
\def\b#1{{\dimen@ 1ex\%
{\setbox\z@\hbox{\char22}\dimen@\ht\z@ \advance\dimen@.25ex\%
 \setbox\z@\hbox{#1}\advance\dimen@\ht\z@ \advance\dimen@\dp\z@
 \global\dimen@i\dp\z@ \global\advance\dimen@i .45ex\%
 \fontdimen5\font\dimen@ \accent22\fontdimen5\font\dimen@ #1\%
 \dp\z@\dimen@i \box\z@}}
\end{verbatim}

Sample output using the revised macros.

Here is the output using the new definitions.

\begin{verbatim}
\def\AA{{\dimen@ 1ex\%
{\setbox\z@\hbox{A}\dimen@\ht\z@ \advance\dimen@-.35ex\%
 \fontdimen5\font\dimen@ \accent'27\fontdimen5\font\dimen@ A}}
\def\c#1{{\dimen@ 1ex\%
{\setbox\z@\hbox{#1}\dimen@\ht\z@ \advance\dimen@\dp\z@
 \fontdimen5\font\dimen@ \accent24\fontdimen5\font\dimen@ #1}}
\def\b#1{{\dimen@ 1ex\%
{\setbox\z@\hbox{\char22}\dimen@\ht\z@ \advance\dimen@.25ex\%
 \setbox\z@\hbox{#1}\advance\dimen@\ht\z@ \advance\dimen@\dp\z@
 \global\dimen@i\dp\z@ \global\advance\dimen@i .45ex\%
 \fontdimen5\font\dimen@ \accent22\fontdimen5\font\dimen@ #1\%
 \dp\z@\dimen@i \box\z@}}
\end{verbatim}

Do you see the differences? How is \accent used to achieve this effect?