

Mainly, though, these macros were written just as a challenge. I learned quite a lot about  $\TeX$  and needed some  $\TeX$ niques I'd never seen before. It was also quite pleasing to see that  $\TeX$  code can be formally verified, albeit in a rather noddy way. Without some sort of abstract view of lists, these  $\TeX$  macros could not have been written.

## 8 Acknowledgements

Thanks to Jeremy Gibbons for letting me bounce ideas off him and spotting the duff ones, to Damian Cugley for saying "Do you really think  $\TeX$  is meant to do this?", and to the Problem Solving Club for hearing me out. This work was sponsored by the Science and Engineering Research Council and Hewlett Packard.

◊ Alan Jeffrey  
Programming Research Group  
Oxford University  
11 Keble Road  
Oxford OX1 3QD  
Alan.Jeffrey@uk.ac.oxford.prg

---

## A Nestable Verbatim Mode

Philip Taylor

A few months ago, Sebastian Rahtz asked me if I could make some changes to the verbatim code which he was currently using, and sent me the source. I found it so opaque that I decided to write my own, and the following evolved over a period of a couple of weeks. I would like to acknowledge my debt to Sebastian, and also to Chris Rowley, without whose helpful comments and criticism the code could never have evolved. Of course, the code is now ten times as opaque as that originally used by Sebastian, but at least I understand it (on a good day, when the moon is the seventh house, and Jupiter  $\backslash$ haligns with Mars).

The idea is as follows: having said

```
\input verbatim
```

at the beginning of one's document, one invokes verbatim mode by

```
\verbatim <char>
```

What follows can then contain any character, with the single exception of  $\langle char \rangle$ , and all such text will be copied *verbatim*, with leading spaces retained but invisible, and all embedded spaces retained

and shewn. If  $\langle char \rangle$  is encountered,  $\TeX$  enters a new inner group (the verbatim environment is itself a group), within which the preceding meaning (i.e.  $\backslash catcode$ ) of all characters is reinstated. This new inner group continues typesetting in the normal (non-*verbatim*) manner until a further  $\langle char \rangle$  is encountered, whereupon it reverts to verbatim mode; the inner 'normal' mode can itself be interrupted by a further

```
\verbatim <char>
```

where  $\langle char \rangle$  can be the same or a different escape character. There is no theoretical limit on the level of nesting, but  $\TeX$  implementations will invariably run out of space (usually save-stack space) if too many levels are attempted.

To end verbatim mode, one enters inner 'normal' mode through the escape character and then says  $\backslash mitabrev$ . Note that this is *not* a reserved string, but simply a macro which expands to  $\{\backslash endgroup \backslash endgroup\}$ ; any other name can be chosen if one finds " $\backslash mitabrev$ " unappealing. Thus, at the outermost level, the call and end to  $\backslash verbatim$  look like:

```
\verbatim <char>
```

```
⋮
```

```
<char> \mitabrev
```

Finally, a mechanism is provided for listing arbitrary files in verbatim mode. If, while in inner 'normal' mode, one says

```
\AfterGroup {\<any balanced text>}
```

(note the case of  $\backslash AfterGroup$ ), the  $\langle balanced\ text \rangle$  will be re-inserted *with its original catcodes* immediately after the closing  $\langle char \rangle$  which terminates inner 'normal' mode. Thus it will not itself be listed *verbatim*, but will be elaborated according to  $\TeX$ 's normal conventions. Thus if one says

```
\AfterGroup {\input <filename>}
```

the contents of the file will be listed in verbatim mode. For example, to list this file itself, one can say

```
\verbatim |
| \AfterGroup {\input verbatim.tex} |
| \mitabrev
```

There remains an anomaly at present: " $\backslash$ " cannot form the escape-character as it will automatically form a  $\langle control\ sequence \rangle$  with the following character(s) when called with

```
\verbatim \
```

I will endeavour to rectify this deficiency in a future release.

The source of Verbatim.TeX follows.

```

\catcode \@ = 11          %%% we use commercial-at as a letter throughout;
\chardef \l@tter = 11    %%% and introduce synonyms for the \catcodes for
\chardef \@ther = 12     %%% <letter> and <other>;
\newcount \c@unt         %%% a loop-counter;
\newcount \ch@rcode      %%% this will hold the character-code of the
                          %%% escape character;
\newif \ifd@bugging      %%% set <true> if you want to watch the
                          %%% finite-state automaton at work;
\newif \ifshewleadingspaces %%% set <true> if you want to see leading spaces
                          %%% shewn as inverted square cup (explicit);
\newif \ifshewembeddedspacestrue %%% set <false> if you want to see embedded
\shewembeddedspacestrue %%% spaces shewn as white space (implicit);
\ifd@bugging             %%% if <debugging>,
  \let \m@ssage = \message %%% \m@ssage is synonymous with \message
\else                    %%% otherwise
  \def \m@ssage #1{%     %%% it simply throws its parameter away;
\fi
%
\def \verbatim #1%      %%% the \verbatim macro takes one parameter
  {\begingroup          %%% and immediately starts a nested group
  \def \n@sted          %%% within which \n@sted is defined
    {\begingroup       %%% to start a further group within which
    \let \n@sted        %%% \n@sted becomes a synonym for \endgroup
      = \endgroup
    \@environment      %%% and the environment is restored to that
                      %%% which obtained two levels of nesting out;
    \ignorespaces      %%% for tidyness, we ignore any <lwspace>
  }%                  %%% which follows the escape character;
\tt                   %%% we assume Knuth's font-selectors and
                      %%% select the 'typewriter' font;
\edef \@environment    %%% we initialise \@environment
  {\parindent =       %%% to prepare to restore \parindent
  \the \parindent
  \parskip =          %%% and \parskip;
  \the \parskip
  \space              %%% and ensure that the value to be assigned to
  }%                  %%% \parskip is properly terminated;
\parskip = 0 pt      %%% we then set \parindent and
\parindent = 0 pt    %%% \parskip to 0 pt;
\c@unt = 0           %%% and initialise \c@unt to 0;
\loop                %%% this loop checks the \catcode of each
                      %%% character code in the range 0...127
                      %%% (or 0...255 for TeX V3) and if it
                      %%% is other than <letter> or <other>, as
                      %%% appropriate, saves the current value in
                      %%% \@environment for subsequent restoration
                      %%% within an inner group; it then sets the
                      %%% \catcode to either <letter> or <other>;
\ifnum \c@unt < \@A%
  \s@ve \catcode \c@unt = \@ther

```



```

\else \edef \@environment
      {\@environment #1\the #2=\the #1#2 }%
      #1#2 = #3%
\fi
}%

%%% the code which follows implements the finite
%%% state automaton which determines whether
%%% <space>s are ignored, shown explicitly or
%%% implied, and which ensures that blank
%%% lines are reproduced correctly.

%
\def \void {\futurelet \nxt \voidifspace}%
\def \l@ad {\l@adingspace \futurelet \nxt \l@adifspace}%
\def \skop {\vskip \baselineskip \futurelet \nxt \l@adifspace}%
\def \emb@d {\emb@ddedspace}%
\def \sh@wspace {\char 32\relax}%
\def \h@despace {\leavevmode \kern \fontdimen 2 \font}%
\def \l@adingspace {\ifshewleadingspaces \sh@wspace \else \h@despace \fi}%
\def \emb@ddedspace {\ifshewembeddedspace \sh@wspace \else \h@despace \fi}%
\def \voidifspace {\testnxt {\afterassignment \void}}%
\def \l@adifspace {\testnxt {\afterassignment \skop}}%
%
%%% \testnxt provides a common look-ahead for
%%% \voidifspace and \l@adifspace, and also
%%% implements some essential debugging hooks.
\def \testnxt #1%
  {\ifx \nxt \sp@c@
    \m@ssage {Next character is a space}%
    \let \nxt = \relax
  \else \ifx \nxt \r@t@rn
    \m@ssage {Next character is a return}%
    \def \nxt {#1\let \nxt = }%
  \else \m@ssage {Next character is \meaning \nxt}%
    \let \nxt = \relax
    \x \let \sp@ce = \emb@d
  \fi
  \fi
  \nxt
}%

%
\catcode \ = \active%      %%% We next tamper with the \catcode of <space>
\def \sp@ce{ }%           %%% and <return>, while defining macros and
                           %%% synonyms which require them to be active;
                           %%% the \catcode is then restored to its default
                           %%% (not necessarily the previous value —
                           %%% could be improved). \@ctivespace makes
                           %%% <space> active, then defines <space> as
                           %%% \void with a synonym \sp@c@. This code is
                           %%% used by the finite-state automaton.

\def \@ctivespace%
  {\catcode \ = \active \def {\void}\let \sp@c@= }\catcode \ = 10 \relax%
%
\catcode \^M = \active %   %%% <return> is made active;
\def \r@turn {\^M}%       %%% \r@turn defined as an active <return>;
\let \r@t@rn = \^M%       %%% \r@t@rn is made a synonym;

```

```

\def \@ctivecr %          %%% and \@ctivecr is defined to
  {\catcode '\^^M = \active % %%% make <return> active, then
  \def ^^M%              %%% define <return> to manipulate the
                          %%% finite-state automaton and ...
    {\@x \def \sp@ce {\l@ad}%
    \@x \let \@x \sp@c@ \@x =\sp@ce %
%
%   \endgraf %          %%% insert a \par primitive (for blank lines).
%
%   \futurelet \n@xt \l@adifspace %
%   }%
  \let \r@t@rn = ^^M%    %%% \r@t@rn is synonymous with active <return>
}%
\catcode '\^^M = 5 %    %%% finally, the \catcode of <return> is
                          %%% restored to its normal value;
%
%                          %%% the \AfterGroup macro is intended for
%                          %%% use within a nested normal environment,
%                          %%% and causes (a concealed macro defined as)
%                          %%% its parameter text to be inserted into
%                          %%% TEX's input stream when the nested normal
%                          %%% group terminates.
%
\def \AfterGroup #1{\global \def \@ftergroup {#1}\aftergroup \@ftergroup}%
%
\let \@x = \expandafter %%% \@x is a brief synonym for \expandafter;
%
\catcode '\@ = \@ther %%% commercial-at is restored to its normal
                          %%% <other> catcode (not necessarily the
                          %%% previous value — could be improved);
%
\def \mitabrev           %%% and \mitabrev defined as the closure for
  {\endgroup \endgroup}% %%% \verbatim; any other name could be used,
                          %%% as the code performs no look-ahead for
                          %%% any particular string.
%
%                          %%% Finally we announce to the world that we
%                          %%% have been loaded, and give some clues as
%                          %%% to the usage.
%
\message {Verbatim environment loaded;}%
\message {usage: ‘\noexpand \verbatim <char> ... <char> \noexpand \mitabrev’}%

```

◊ Philip Taylor  
 Royal Holloway and Bedford New College  
 P.Taylor@Vax.Rhbc.AC.Uk