Macros

Multiple, Independent Marks

Jim Fox
University of Washington

TEX provides marks to help synchronize output routines with macro expansion. They are necessary because macros are expanded before text is fitted onto a page — and therefore are expanded before page numbers are assigned or page boundaries are known. The TEXbook has numerous examples of elementary mark usage. And, as long as only one kind of information, chapter numbers, for example, needs to be communicated, the marks as described work well. However, some implementations need to keep track of a couple of things, or need to use all of the marks on a page. A more flexible approach is required.

Knuth considers this problem in the TEXbook when he suggests passing several 'independent' items of information with "`\a \or \b \or \c ...`" constructions, but this is really a deception. The items in the `\or` constructs are not very independent — they all have to change values at the same time.

The macro package presented here provides multiple, independent marks for output routine synchronization. It solves the concurrency problem by keeping all mark text in lists — with seperate lists for independent marks. A continually incrementing sequence number identifies which elements of the list correspond to the conventional top, first, and bot marks. This sequence number constitutes the only real mark in the system.

The mark lists are both an advantage and an encumberance. Since all marks on a page are recorded, they are all available to the output routine and can be used for seemingly un-mark related functions. (An example is given at the end of the article.) The marks also take up macro space. Documents such as dictionaries that use a lot of marks should be carefully coded to use the shortest mark text possible. The TEX primitive `\noexpand` can be very useful here.

How the marks are used

Since each of the multiple marks is completely independent of the others, their usage is quite simple. First define a new mark, `\abc`, for example, by

```
\newmark\abc
```

Then use `\abc` as if it were `\mark`

```
\abc{mark text}
```

or

```
\abc{top text \else bot text}
```

In the output routine extract the `abc` marks with

```
\getmarks\abc
```

This defines three control sequences: `\topabc`, `\firstabc`, and `\botabc`.

You can then use these just as you would use the conventional `\topmark`, `\firstmark`, and `\botmark`.

How the macros work

List macros are discussed in Appendix D of the TEXbook. Mark lists are similar to those in the book, but have a sequence number in addition to the list marker and text. The lists have the form:

`\m@rker`⟨seq⟩`{`⟨text⟩`}` `\m@rker`⟨seq⟩`{`⟨text⟩`}` ...

where `\m@rker` is undefined until the list is expanded, ⟨seq⟩ is an increasing decimal number, and ⟨text⟩ is the mark text.

The argument of `\newmark` is defined to add a new item to the corresponding list. Note that the sequence number is shared by all lists. For example:

```
\newmark\x \newmark\y
\x{a} \y{b} \x{c}
```

defined the mark lists to be

```
\list@x={\m@rker0{}\m@rker1{a}\m@rker3{c}}
\list@y={\m@rker0{}\m@rker2{b}}
```

When the lists are expanded in an output routine, via `\getmarks`, each sequence number is compared to the numbers in the real `\topmark` and `\botmark`. Let ⟨top⟩ and ⟨bot⟩ be those numbers, respectively, and let the list we are processing be `\list@xxx` (from `\newmark\xxx`). Then the text of the list element with the greatest ⟨seq⟩ where ⟨seq⟩ $\leq$ ⟨top⟩ becomes the topmark (`\topxxx`). The text of the first list element where ⟨seq⟩ $>$ ⟨top⟩ becomes the firstmark (`\firstxxx`). Finally, the text of the last list element where ⟨seq⟩ $\leq$ ⟨bot⟩ becomes the botmark (`\botxxx`).

An example that uses all of the marks

Here is a partial crossreference capability that makes use of mark lists.

Define an alternate mark list scanner

```
\catcode`@=11
\def\m@rksdef@#1{%
   \ifnum\curm@rk>\bm@rk \let\m@rker=\m@rkrecover
     \t@b=\e@{\e@\m@rker\the\curm@rk{#1}}%
   \@else \ifnum\curm@rk>0
     \m@rk@csafter{\xdef}{#1}{\number\pageno}\@fi
     \t@a={\m@rker0{}}\@fi}
\catcode`@=12
```

At the beginning of your file initialize the cross-reference mark list

```
\newmark\xref % marker for cross references
```

and in the output routine

```
\begingroup\let\m@rkscan@=\m@rksdef@
 \getmarks\xref
 \endgroup
```

Then a cross-reference definition, `\xref{alfa}`, for example, will define the control sequence `\alfaxref` to expand to the page number of the page containing the "`\xref{alfa}`". The output routine must have processed the page in question, of course.

The marks macro package

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% multi-marks macro package
% by Jim Fox, Oct 1986
%
\catcode`\@=11
%
%     \expandafter is used a lot, so we use a short name for it
%
\let\e@=\expandafter
%
\toksdef\t@a=0 \toksdef\t@b=2 \toksdef\t@c=4 % temporary registers
%
%     \list@cs makes a control sequence by adding the preface 'list@'
%     to its argument;  e.g., \list@cs\xx --> \list@xx
%     Note that it requires a double expansion
%
%     \list@csafter reaches over its argument to create a control sequence
%     via \list@cs; e.g., \list@csafter\def\xx --> \def\list@xx
%
\def\list@cs#1{\csname list@\string#1\endcsname}
\def\list@csafter#1{\e@\e@\e@#1\list@cs}
```

```
%
%     \newmark\xx defines a new mark.  It initializes the mark's
%     list macro, \list@xx, and defines \xx.
%
%     The list macros, \list@xx in this example, have the form:
%         \m@rker<seq>{<text>}
%         \m@rker<seq>{<text>}.....
%     where \m@rker is undefined until the list is actually expanded,
%     <seq> is a continually incrementing, decimal sequence number,
%     and <text> is the actual mark text.
%
%     The new mark, \xx in this example, is defined to add its argument,
%     along with a \m@rker and sequence number to the end of the
%     corresponding list.  It also sets a 'real' mark containing the
%     new sequence number.
%
\newcount\m@rkcounter\m@rkcounter=0
\def\newmark#1{\begingroup\escapechar=-1
  \list@csafter\gdef#1{\m@rker0{}}% start list
  \long\gdef#1##1{\begingroup\escapechar=-1\setm@rks
    \list@csafter\addm@rk#1{\the\m@rkcounter{##1}}\endgroup}\endgroup}
%
%     \addm@rk puts the \m@rker, sequence number, and new mark text
%     at the end of a 'list' macro.
%
\long\def\addm@rk#1#2{\t@a=\e@{#1}\xdef#1{\the\t@a\noexpand\m@rker#2}}
%
%     \setm@rks increments the mark counter and sets the mark
%
\def\setm@rks{\global\advance\m@rkcounter by1\mark{\the\m@rkcounter}}
%
%     In an output routine \getmarks\xx will extract from \list@xx
%     the appropriate \topxx, \firstxx, and \botxx marks.
%
\newcount\tm@rk \newcount\bm@rk  % top and bot mark numbers
\newcount\ecs@ve  % saves the value of \escapechar
\newif\ifnofirstm@rk  % true when \firstxx has been defined
\newcount\curm@rk  % records the sequence number from the mark list
%
%     \m@rk@csafter reaches over its first argument to build
%     one of the control sequences; \topxx, \firstxx, or \botxx
%     e.g., if the text was \getmarks\xx, then \m@rkname expands to 'xx'
%     and  \m@rk@csafter\show{top} --> \show\topxx
%     and  \m@rk@csafter{\long\e@\def}{bot} --> \long\def\botxx
%
\def\m@rk@csafter#1#2{\e@#1\csname#2\m@rkname\endcsname}
```

```
%
%     \getmarks scans the mark list and sets:
%
%        \topxx   = the text of the last list item whose sequence number
%                   is less than or equal to \topmark
%        \firstxx = the text of the first list item whose sequence number
%                   is greater than \topmark
%        \botxx   = the text of the last list item whose sequence number
%                   is less than or equal to \botmark
%
\def\getmarks#1{\tm@rk=0\topmark \bm@rk=0\botmark
  \t@b={}\nofirstm@rktrue
  \ecs@ve=\escapechar \escapechar=-1
  \edef\m@rkname{\string#1}%  used by \m@rk@csafter
  \let\@fi=\fi\let\fi=\relax \let\@or=\or\let\or=\relax
  \let\@else=\else\let\else=\relax % so mark text can contain \else,\or,\fi
  \let\m@rker=\m@rkscan \list@cs#1\m@rkend \m@rkrestore % 'do' the list
  \let\else=\@else \let\fi=\@fi \let\or=\@or
  \escapechar=\ecs@ve}
%
%     \m@rkscan looks at the next list item and defines the appropriate
%     \topxx, \firstxx, and \botxx control sequences
%     They are \long\def'd to handle the unusual case of marks with \par's
%
\def\m@rkscan{\afterassignment\m@rkscan@\curm@rk=}
\long\def\m@rkscan@#1{%
  \ifnum\curm@rk>\tm@rk\@else\m@rk@csafter{\long\e@\def}{top}{#1}\@fi
  \ifnum\curm@rk>\bm@rk \let\m@rker=\m@rkrecover
       \t@b=\e@{\e@\m@rker\the\curm@rk{#1}}%
  \@else \ifnofirstm@rk\m@rk@csafter{\long\e@\def}{first}{#1}%
            \ifnum\curm@rk>\tm@rk\nofirstm@rkfalse\@fi\@fi
     \m@rk@csafter{\long\e@\def}{bot}{#1}%
    \t@a=\e@{\e@\m@rker\the\curm@rk{#1}}\@fi}
%
%     at the end some of the list must be restored
%
%     \m@rkrecover is used when there are unscanned list items, otherwise
%     \m@rkend is reached and stops the scan
%
\long\def\m@rkrecover#1\m@rkend{\t@c={\m@rker#1}}
\def\m@rkend{\t@c={}}
%
%     \m@rkrestore rebuilds the list (with at least a new \topxx}
%
\def\m@rkrestore{%
   \m@rk@csafter{\long\e@\xdef}{list@}{\the\t@a\the\t@b\the\t@c}}
%
\catcode`\@=12
%
```