```
% #1: Some string
% #2: substring to test for whether it
%     is in #1 or not.
\def\IfSubString #1#2{%
    \edef\@MainString{#1}%
    \def\@TestSubS ##1#2##2\@Del{%
        \edef\@TestTemp{##1}}%
    \expandafter\@TestSubS
        \@MainString#2\@Del
    \ifx\@MainString\@TestTemp
        \@TestSubStringfalse
    \else
        \@TestSubStringtrue
    \fi
    \if@TestSubString
}
\catcode`@ = 12
```

Example 11: \expandafter and \csname

A character string enclosed between \csname and \endcsname expands to the token formed by the character string. \csname a?a-4\endcsname, for instance, forms the token \a?a-4. If you wanted to use this token in a macro definition you have to do it the following way:

```
\expandafter
    \def\csname a?a-4\endcsname{...}
```

The effect of the \expandafter is of course to give \csname a chance to form the requested token rather than defining a new macro called \csname.

Summary

These examples have shown some typical applications of \expandafter. Some were presented to "exercise your brains a little bit". I recommend that you take the examples and try them out; there is very little input to enter. I also encourage you to tell Barbara Beeton or me what you think about tutorials in TUGboat. There are many more subjects which could be discussed and which may be of interest to you.

This article is, as briefly mentioned in the introduction, an adaptation of a section of my book, *Another Look At TEX*, which I am currently finishing. The book, now about 800 pages long, grew out of my teaching and consulting experience. The main emphasis of the book is to give concrete and useful examples in all areas of TEX. It contains, to give just one example, 100 (!!) \halign tables. In this book you should be able to find an answer to almost any TEX problem.

Macros for Outlining

James W. Walker
Department of Mathematics
University of South Carolina

The purpose of this note is to describe standalone macros for the preparation of outlines in the standard format. For instance, the desired output might look like:

I. Vegetables
   A. Green ones
      1. lettuce
         a. iceberg
         b. leaf
      2. Broccoli, almost universally despised by children. The strong flavor is only made palatable by quick stir-frying.
   B. white ones
      1. potatoes
      2. turnips

II. Animals.

III. Minerals.

Notice that a topic is allowed to be a paragraph, not just one line, as in topic I.A.2. I wanted TEX to take care of the counting and indentation as painlessly as possible. Something like this can be done in LATEX using nested enumerate environments, but I wanted the input format to be even simpler.

When typing an outline, it is natural to show the structure by indenting with the tab key. This is particularly easy if one has a text editor with an automatic indentation feature. With that feature, hitting the Return key produces a new line with the same amount of indentation as the previous line. When the input is typed this way, we can tell the indentation level of a topic by counting tabs. We also need to mark the beginning of a topic, since not every line begins a new topic. I chose to mark a new topic with a pound sign (#). Thus, the input to produce the outline above could look something like:

```
\beginoutline
# Vegetables
        # Green ones
                # lettuce
                        # iceberg
                        # leaf
                # Broccoli, almost
                universally despised
                by children.  The
                strong flavor is
                only made palatable
                by quick stir-frying.
        # white ones
                # potatoes
                # turnips
# Animals.
# Minerals.
\endoutline
```

To make this work, an obvious step is to make the pound sign an active character which will typeset a label for a topic. However, there is no obvious way to make it look backwards and count tabs. Therefore I decided to make the tab character active also, and make it count itself. More precisely, the first tab on a line uses `\futurelet` to see whether the next token is a tab, a pound sign, or something else. If the next token is a tab, it increments a counter, gobbles the tab, and recursively looks for more tabs. If a pound sign is the first thing after a sequence of tabs, then the macro formats a topic at the appropriate indentation level. If the first thing after a sequence of tabs is anything else, nothing happens. Notice that the pound sign comes into play as an active character only for level 1 topics, i.e., when the pound sign is not preceded by any tabs.

And now, the macros. We begin by making sure that the macros are not loaded twice, and resetting the category code of the at sign. We save the old category code of the at sign, because in some formats (e.g., AMS-TEX) the at sign might have a category code other than "other".

```
\ifx\outlineformatloaded\relax
        \endinput
\else
        \let\outlineformatloaded=\relax
\fi

\chardef\oldatsigncatcode=\catcode`\@
\catcode`\@=11
```

There is one count register for each of 5 levels of indentation, which is perhaps a bit extravagant. The counter `\outline@lastlevel` is used to write an error message if the indentation level increases by more than 1 at a time. The only parameter that should be directly altered by the user is `\outlineindent`, the width of each indentation. If this is not large enough, and if the topic numbers get large, an overfull hbox could result.

```
\newdimen\outlineindent
\outlineindent=2em

\newcount\outline@i
\newcount\outline@ii
\newcount\outline@iii
\newcount\outline@iv
\newcount\outline@v
\newcount\outline@lastlevel
\newcount\outline@levelcount
```

Next we define `\beginoutline` and `\endoutline`. Be warned that we must not format the definition of `\beginoutline` with tabs, only with spaces.

```
{%
   \catcode`\#=\active
   \catcode`\^^I=\active
   \gdef\beginoutline{%
      \par
      \bgroup
      \outline@i=1
      \outline@lastlevel=0
      \catcode`\#=\active
      \let#=\outline@topicmarker
      \catcode`\^^I=\active
      \let^^I=\outline@selfcount
   }% End of \beginoutline.
}%

\def\endoutline{%
    \par
    \medbreak
    \egroup
}%
```

A level 1 topic is marked with an active pound sign, which is let equal to the following macro.

```
\def\outline@topicmarker{%
    \par
    \parindent=\outlineindent
    \medbreak
    \hang
    \indent
    \llap{\hbox to \outlineindent{%
            \global\outline@ii=1
            \uppercase
            \expandafter
            {\romannumeral
            \outline@i}%
            .%
            \hfil
    }}% end of \hbox and \llap.
    \global\advance\outline@i by 1
    \outline@lastlevel=1
    \ignorespaces
}% End of \outline@topicmarker.
```

The active tab character is made to count tabs using the following macros. Note that the parameter of `\outline@innerselfcount` will always be an `\outline@selfcount` token, which is just counted and then thrown away.

```
\def\outline@selfcount{%
    \outline@levelcount=2
    \futurelet\next\outline@next
}%

\def\outline@innerselfcount#1{%
    \advance\outline@levelcount by 1
    \futurelet\next\outline@next
}%

\def\outline@next{%
    \ifx\next\outline@selfcount
        \let\next
            =\outline@innerselfcount
    \else
        \ifx\next\outline@topicmarker
            \let\next=\outline@subtopic
        \else
            \let\next=\ignorespaces
        \fi
    \fi
    \next
}% End of \outline@next.
```

A sequence of tabs ended by a pound sign starts a subtopic.

```
\def\outline@subtopic#1{%
    \par
    \parindent=%
      \outline@levelcount\outlineindent
    \ifnum \outline@levelcount=2
        \smallbreak
    \fi
    \advance\outline@lastlevel by 1
    \ifnum \outline@levelcount>%
            \outline@lastlevel
        \errmessage{The outline level
            can't increase by more
            than 1 at a time!}%
    \fi
    \outline@lastlevel
        =\outline@levelcount
    \hang
    \indent
    \llap{\hbox to \outlineindent{%
        \ifcase\outline@levelcount
        \or % case 1: done elsewhere.
        \or % case 2: A, B, C, etc.
            \global\outline@iii=1
            \count0=\outline@ii
            \advance\count0 by `A%
            \advance\count0 by -1
            \char\count0.%
            \global\advance
                \outline@ii by 1
        \or % case 3: 1,2,3, etc.
            \global\outline@iv=1
            \number\outline@iii.%
            \global\advance
                \outline@iii by 1
        \or % case 4: a,b,c, etc.
            \global\outline@v=1
            \count0=\outline@iv
            \advance\count0 by `a%
            \advance\count0 by -1
            \char\count0.%
            \global\advance
                \outline@iv by 1
        \or % case 5: i,ii,iii,iv etc.
            \romannumeral\outline@v.%
            \global\advance
                \outline@v by 1
        \else % all deeper levels
            $\bullet$%
        \fi
        \hfil
    }}% end of \hbox and \llap.
    \ignorespaces
}% End of \outline@subtopic.
```

The outlining macros are now complete. There is one small problem: One might occasionally need to use the pound sign for its normal TEX function of marking a parameter in a \def or \halign, inside an outline. We can make that possible by providing a macro that temporarily changes the category code of the pound sign back to normal.

```
\def\normalpoundsign{%
    \bgroup
    \catcode`\#=6
    \innernormalpoundsign
}%
\def\innernormalpoundsign#1{#1\egroup}%
```

Thus an \halign could be enclosed in \normalpoundsign{...}.

Finally we restore the at sign to its former category code.

```
\catcode`\@=\oldatsigncatcode
```

A Macro Writing Tool: Generating New Definitions

Amy Hendrickson
TEXnology Inc.

Suppose you come upon a situation where you need a macro which will generate another new macro every time it is used. I came upon a solution to this problem and want to share it with TUG readers in case someone would find it an useful macro writing tool, or maybe just find it amusing.

The problem that I ran into that necessitated this kind of macro (it is by no means the only application) had to do with a set of macros that I was writing recently for slide generation: How can you take large chunks of text possibly containing tables, listings, verbatim text, or section headers, and a) print the chunk where it appears in the document, then b) send it to the end of the file to be printed in slide format. (This format would include larger font and baselineskip, possibly be in landscape mode, and have rounded corner edging.)

Since you cannot send a large body of text to an auxiliary file, the solution seemed to be to write one macro which would generate as many definitions as there were chunks of text to be made into slides, and send only the control sequence and slide formatting information to an auxiliary file. The auxiliary file can then be input at the end of the original file, and the definitions that were made earlier in the file will produce the slides.

But how can one generate such a series of definitions, each with a new name? The solution involves using the letters of roman numerals as the name of the each new macro. A counter is advanced to produce a new roman numeral each time the macro is used. With the right macro expansion, the roman numerals will be interpreted as a sequence of letters, and a new sequence of letters will be available each time.

For instance, say we set the counter equal to 637 to start, and advance it by one every time the macro is used. The first set of letters that will become a control sequence will be \dcxxxvii, the second \dcxxxviii, etc.

To make certain that these letters have not already been used in a definition, we can also supply, following the roman numeral, a sequence of letters that does not change, and thus make the possiblity of renaming a previously defined control sequence very small. That is the function of the \unique definition below.

Here is some code, showing how \newdefs can be used to define #1 as a new definition every time the macro is used.

```
\newcount\definitionnum  \definitionnum=2001

\def\newdefs#1{\advance\definitionnum by 1
  \def\unique{\the\definitionnum ZZZZ}
  \expandafter\gdef
    \csname\romannumeral\unique\endcsname{#1}}
```

In use,

```
\newdefs{This is a chunk of text}
```

will produce

```
\gdef\mmiiZZZZ{This is a chunk of text}
```

a control sequence that can be called for later in the file in whatever application it might be useful.