1 A compendium of useful macros

This section describes `eplain.tex`, a collection of macros and other definitions that extend plain TEX. The descriptions of the various macros explain their purposes, but usually do not explain how they work or provide explicit details on how to use them. That information is contained in the source files for `eplain.tex` and in the documentation that comes with it. See "Resources" (p. 'resources') for how to obtain `eplain.tex`.

Preliminaries

We start with some macros for changing category codes and convenient definitions for two of the commonly used ones.

```
\def\makeactive#1{\catcode`#1 = \active \ignorespaces}%
\chardef\letter = 11 \chardef\other = 12
\def\uncatcodespecials{%
   \def\do##1{\catcode`##1 = \other}%
   \dospecials}% Defined in plain.
```

In order to define '^^M' as an active character, you need to encase the definition in a group and invoke some extra machinery. The `\letreturn` macro lets you define '^^M' without that extra machinery (which you can see in the definition below).

```
{\makeactive\^^M \long\gdef\letreturn#1{\let^^M = #1}}%
```

These macros consume one, two, or three arguments.

```
\def\gobble#1{}\def\gobbletwo#1#2{}%
\def\gobblethree#1#2#3{}%
```

Now we establish some conventions for reading the rest of the file. Within the file we allow "private" control sequences that contain '@' in their names. These control sequences aren't accessible outside of this file (unless you change the category code of '@' again).

```
\catcode`@ = \letter    % Allow control sequences with @.
\let\@plainwlog = \wlog % Don't log register allocations.
\let\wlog = \gobble
\newlinechar = `^^J
```

The next two macros provide convenient forms of diagnostic output. \loggingall turns on all tracing, but causes the trace output to appear only in the log file and not at your terminal. \tracingboxes causes boxes to be displayed completely when they're traced. (TEX normally shows only three levels of boxing and five items within each box.)

```
\def\loggingall{\tracingcommands\tw@\tracingstats\tw@
   \tracingpages\@ne\tracingoutput\@ne
   \tracinglostchars\@ne\tracingmacros\tw@
   \tracingparagraphs\@ne\tracingrestores\@ne
   \showboxbreadth\maxdimen\showboxdepth\maxdimen}%
\def\tracingboxes{\showboxbreadth = \maxdimen
   \showboxdepth = \maxdimen}%
```

The default thickness of rules is 0.4 pt. You can produce rules of any default thickness you choose by redefining \vruledefaultwidth, \hruledefaultheight, and \hruledefaultdepth and then using \ehrule and \evrule instead of \hrule and \vrule. (The 'e' stands for "eplain".) If you give an explicit dimension (e.g., \ehrule height 16pt), TEX will use it.

```
\newdimen\hruledefaultheight  \hruledefaultheight = 0.4pt
\newdimen\hruledefaultdepth   \hruledefaultdepth = 0.0pt
\newdimen\vruledefaultwidth   \vruledefaultwidth = 0.4pt
\def\ehrule{\hrule height\hruledefaultheight
   depth\hruledefaultdepth}%
\def\evrule{\vrule width\vruledefaultwidth}%
```

The \% convention for writing a '%' character doesn't work when you want to include that character in the token list of \write. You can use \percentchar to achieve this. We also redefine ^^L to be nonouter so that you can use it in a macro definition or argument.

```
{\catcode`\% = \other \gdef\percentchar{%}}%
 \def^^L{\par
}%
```

\tokstostring converts its argument into a list of character tokens. It uses only expansions that are handled in TEX's gullet. This property

is necessary for it to work with `\edef`. It is used by the cross-referencing macros (p. 'xrefs').

In order to split the argument up at spaces, we have to use two subsidiary macros. `\@ttsA` finds the spaces, and `\@ttsB` handles a token sequence without any spaces. Each space is replaced by the expansion of `\spacesub`.

```
\def\tokstostring#1{\@ttsA#1 \ttsmarkA}%
\def\@ttsA#1 #2\ttsmarkA{\ifempty{#1}\else
      \@ttsB #1\@ttsmarkB
      \ifempty{#2}\else
         \spacesub\@ttsA#2\ttsmarkA\fi\fi}%
\def\@ttsB#1{\ifx #1\@ttsmarkB\else
      \string #1%
      \expandafter\@ttsB\fi}%
\def\@ttsmarkB{\@ttsmarkB}% should never be expanded
\def\spacesub{+}%
```

`\ifempty` tests if its argument is empty.

```
\def\ifempty#1{\@ifempty #1\@emptymarkA\@emptymarkB}%
\def\@ifempty#1#2\@emptymarkB{\ifx #1\@emptymarkA}%
\def\@emptymarkA{\@emptymarkA}%
```

The `\for` macro implements a TEX version of the for loop in traditional programming languages. These macros come directly from LATEX.

```
\def\for#1:=#2\do#3{\edef\@fortmp{#2}%
   \ifx\@fortmp\empty \else
      \expandafter\@forloop#2,\@nil,\@nil\@@#1{#3}\fi}%
\def\@nnil{\@nil}%
\def\@fornoop#1\@@#2#3{}%
\def\@forloop#1,#2,#3\@@#4#5{\def#4{#1}\ifx #4\@nnil
   \else #5\def#4{#2} ifx #4\@nnil \else
      #5\@iforloop #3\@@#4{#5}\fi\fi}%
\def\@iforloop#1,#2\@@#3#4{\def#3{#1}\ifx #3\@nnil
   \let\@nextwhile=\@fornoop \else #4\relax
      \let\@nextwhile=\@iforloop\fi
         \@nextwhile#2\@@#3{#4}}%
```

`\obeywhitespace` is useful for reproducing line breaks, blank lines, and spaces in your input. It combines the effects of `\obeylines` and `\obeyspaces`, and also causes spaces at the start of a line to be printed. Tab characters are not affected by this; they still produce normal glue.

```
\def\alwaysspace{\hglue\fontdimen2\the\font \relax}%
{\makeactive\^^M \makeactive\ %
\gdef\obeywhitespace{%
\makeactive\^^M\def^^M{\par\indent}%
```

```
\aftergroup\@removebox% Kill extra paragraph at end.
\makeactive\ \let =\alwaysspace}}%
\def\@removebox{\setbox0=\lastbox}
```

\frac is a good way to print fractions in text when you don't want to use \over and "1/2" just doesn't look right. This macro is the answer to Exercise 11.6 of *The TEXbook*.

```
\def\frac#1/#2{\leavevmode
   \kern.1em \raise .5ex \hbox{\the\scriptfont0 #1}%
   \kern-.1em $/$%
   \kern-.15em \lower .25ex \hbox{\the\scriptfont0 #2}}%
```

The following macros produce logos that are useful in the TEX world. The $\mathcal{AMS}$-TEX logo is from page 420 of *The TEXbook*. The LATEX logo is slightly modified from the one in latex.tex (we use a different font for the 'A'); similarly, the BibTEX logo uses \sevenrm instead of a true caps-and-small-caps font. The .mf source file for the METAFONT logo is given in the METAFONT manual:

Knuth, Donald E., *The METAFONTbook*. Reading, Mass.: Addison-Wesley, 1986.

```
\def\LaTeX{L\kern-.26em \raise.6ex\hbox{\fiverm A}%
   \kern-.15em TeX}%
\def\AMSTeX{$\cal A\kern-.1667em \lower.5ex\hbox{$\cal M$}%
   \kern-.125em S$-\TeX}%
\def\BibTeX{{\rm B\kern-.05em {\sevenrm I\kern-.025em B}%
   \kern-.08em T\kern-.1667em \lower.7ex\hbox{E}%
   \kern-.125emX}}%
\font\mflogo = logo10
\def\MF{{\mflogo META}{\tenrm \-}{\mflogo FONT}}%
```

The next two macros produce boxes. \blackbox produces a "square bullet", used in the list macros (p. 'listmacs'). \makeblankbox (from page 311 of *The TEXbook*) produces an unfilled rectangle, with the thickness of the border rules given by the arguments.

```
\def\blackbox{\vrule height .8ex width .6ex depth -.2ex}%
\def\makeblankbox#1#2{%
   \hbox{\lower\dp0\vbox{\hidehrule{#1}{#2}%
      \kern -#1% overlap rules
      \hbox to \wd0{\hidevrule{#1}{#2}%
         \raise\ht0\vbox to #1{}% vrule height
         \lower\dp0\vtop to #1{}% vrule depth
         \hfil\hidevrule{#2}{#1}}%
      \kern-#1\hidehrule{#2}{#1}}}}%
\def\hidehrule#1#2{\kern-#1\hrule height#1 depth#2
   \kern-#2}%
```

```
\def\hidevrule#1#2{\kern-#1{\dimen0 = #1
   \advance\dimen0 by #2 \vrule width\dimen0}\kern-#2}%
```

`\numbername` produces the written-out form of a number. (If the number is greater than ten, the macro just reproduces the numerals of its argument.)

```
\def\numbername#1{\ifcase#1%
   zero\or one\or two\or three\or four\or five%
   \or six\or seven\or eight\or nine\or ten\or #1\fi}%
```

`\testfileexistence` determines whether a file `\jobname.#1` is non-empty and sets `\iffileexists` appropriately. The file name in the argument need not end in a space token since the macro provides the space token.

```
\newif\iffileexists
\def\testfileexistence#1{\begingroup
      \immediate\openin0 = \jobname.#1\space
      \ifeof 0\global\fileexistsfalse
      \else \global\fileexiststrue\fi
      \immediate\closein0
   \endgroup}%
```

Displays

By default, TEX centers displayed material (the material between `$$`'s). `\leftdisplays` causes displays to be left-justified by default. You can return to centered displays with `\centereddisplays`.

The macros here are more general than they need to be just for doing left-justified displays. For every display, `\ifeqno` will be true if an `\eqno` occurred in the display. `\ifleqno` will be true if an `\leqno` occurred. If either kind of equation number occurred, `\eqn` produces the text of the equation number. `\eq` always produces the text of the equation itself.

These macros are based on the code on page 376 of *The TEXbook*.

```
\newif\ifeqno \newif\ifleqno
\newtoks\@eqtoks \newtoks\@eqnotoks
\def\eq{\the\@eqtoks}\def\eqn{\the\@eqnotoks}%
\def\displaysetup#1$${%
   \@displaytest#1\eqno\eqno\@displaytest}%
\def\@displaytest#1\eqno#2\eqno#3\@displaytest{%
   \if #3% No \eqno, check for \leqno:
      \@ldisplaytest#1\leqno\leqno\@ldisplaytest
   \else
```

```
      \eqnotrue \leqnofalse % Have \eqno, not \leqno.
      \@eqnotoks = {#2}\@eqtoks = {#1}%
   \fi
   \generaldisplay$$}%
\def\@ldisplaytest#1\leqno#2\leqno#3\@ldisplaytest{%
   \@eqtoks = {#1}%
   \if #3%
      \eqnofalse % No \leqno; we're done.
   \else
      \eqnotrue \leqnotrue % Have \leqno.
      \@eqnotoks = {#2}%
   \fi}%
```

You can format displays differently by defining your own macro, analogous to `\leftdisplays`. The macro definition must place a call on `\displaysetup` in `\everydisplay` so as to ensure that `\displaysetup` is called at the start of every display. The macro definition must also include a definition of `\generaldisplay`.

```
\newtoks\previouseverydisplay
\def\leftdisplays{%
   \previouseverydisplay = \everydisplay
   \everydisplay =
     {\the\previouseverydisplay \displaysetup}%
   \def\generaldisplay{%
      \leftline{%
         \strut \indent \hskip\leftskip
         \dimen0 = \parindent
         \advance\dimen0 by \leftskip
         \advance\displaywidth by -\dimen0
         \@redefinealignmentdisplays
         \ifeqno \ifleqno
            \kern-\dimen0
            \rlap{$\displaystyle\eqn$}%
            \kern\dimen0
         \fi\fi
         $\displaystyle{\eq}$%
         \ifeqno \ifleqno\else
            \hfill $\displaystyle{\eqn}$%
         \fi\fi}}%
\def\centereddisplays{\let\displaysetup = \relax}%
```

`\leftdisplays` must go to some pains to make sure that `\display-lines`, `\eqalignno`, and `\leqalignno` still work properly. `\eq` is typeset in math mode, and `\halign` is illegal in math mode. We use `\vcenter` to change the context so that `\halign` becomes legal again. We also remove the `\hfil` commands at the left of the template to obtain the flush left formatting. Other than those changes, the macros are the same as in `plain.tex`.

```
\def\@redefinealignmentdisplays{%
   \def\displaylines##1{\displ@y
      \vcenter{\halign{\hbox to\displaywidth{$\@lign
            \displaystyle####\hfil$}\crcr##1\crcr}}}%
   \def\eqalignno##1{\displ@y
      \vcenter{\halign to\displaywidth{%
             $\@lign\displaystyle{####}$\tabskip\z@skip
            &$\@lign\displaystyle{{}####}$
               \hfil\tabskip\centering
            &\llap{$\@lign####$}\tabskip\z@skip\crcr
            ##1\crcr}}}%
   \def\leqalignno##1{\displ@y
      \vcenter{\halign to\displaywidth{%
             $\@lign\displaystyle{####}$\tabskip\z@skip
            &$\@lign\displaystyle{{}####}
               $\hfil\tabskip\centering
            &\kern-\displaywidth
             \rlap{\kern-\parindent\kern-\leftskip$
                \@lign####$}%
             \tabskip\displaywidth\crcr
            ##1\crcr}}}}%
```

Time of day

When TEX starts up, it sets the values of the `\time`, `\day`, `\month`, and `\year` parameters. `\monthname` produces the name of the month, abbreviated to three letters. `\timestring` produces the current time, as in "1:14 p.m.". `\timestamp` produces the text of the complete date, as in "23 Apr 1964 1:14 p.m.".

```
\def\monthname{%
   \ifcase\month
      \or Jan\or Feb\or Mar\or Apr\or May\or Jun%
      \or Jul\or Aug\or Sep\or Oct\or Nov\or Dec%
   \fi}%
```

```
\def\timestring{\begingroup
   \count0 = \time \divide\count0 by 60
   \count2 = \count0   % The hour.
   \count4 = \time \multiply\count0 by 60
   \advance\count4 by -\count0   % The minute.
   \ifnum\count4<10 \toks1 = {0}% Get a leading zero.
   \else            \toks1 = {}%
   \fi
   \ifnum\count2<12 \toks0 = {a.m.}%
   \else            \toks0 = {p.m.}%
      \advance\count2 by -12
   \fi
   \ifnum\count2=0 \count2 = 12 \fi % Make midnight '12'.
   \number\count2:\the\toks1 \number\count4
   \thinspace \the\toks0
\endgroup}%
\def\timestamp{\number\day\space\monthname\space
   \number\year\quad\timestring}%
```

Lists

`\numberedlist` produces numbered lists; `\endnumberedlist` ends them. `\unorderedlist` is analogous. For either of these, items inside the lists begin with `\li` ("list item"). You can put `\listcompact` at the beginning of a list if you don't want any additional space between the items of that list. Lists can be nested arbitrarily.

You can control the spacing between the items more generally by assigning values to the registers listed below. If the items in your lists tend to be long, you might want to make `\interitemskip` nonzero. The left indentation of each list item is given by `\parindent` plus `\listleftindent`; the right indentation of each list item is given by `\listrightindent`.

```
\newskip\abovelistskip  \abovelistskip = .5\baselineskip
\newskip\interitemskip  \interitemskip = 0pt
\newskip\belowlistskip  \belowlistskip = .5\baselineskip
\newdimen\listleftindent  \listleftindent = \parindent
\newdimen\listrightindent \listrightindent = 0pt
\def\listcompact{\interitemskip = 0pt \relax}%
```

Both numbered and unnumbered lists use the macros that follow. We don't change `\parindent`, since many existing macros, e.g., `\footnote`, depend on `\parindent`. We must account for the possibility that items are more than one paragraph long. In this case, all paragraphs after

the first will be indented. We use `\leftskip` and `\rightskip` to indent the list items. Indentation of displays is accounted for by changes to `\everydisplay`.

```
\newdimen\@listindent
\def\beginlist{%
   \@listindent = \parindent
   \advance\@listindent by \listleftindent
   \everydisplay = \expandafter{\the\everydisplay
      % Don't lose user's \everydisplay:
      \advance\displayindent by \@listindent
      \advance\displaywidth by -\@listindent
      \advance\displaywidth by -\listrightindent}%
   \nobreak\vskip\abovelistskip
   \parskip = 0pt
   % \leftskip shifts nested lists to the right on the page.
   \advance\leftskip by \@listindent
   \advance\rightskip by \listrightindent}%
\def\printitem{\par\noindent
   \llap{\hskip-\listleftindent \marker \enspace}}%
\def\endlist{\vskip\belowlistskip}%
```

You can change the way the item labels are typeset by redefining the `\numberedmarker` macro.

```
\newcount\numberedlistdepth \newcount\itemnumber
\newcount\itemletter
\def\numberedmarker{%
   \ifcase\numberedlistdepth
       (impossible)%
   \or \itemnumberout)%
   \or \itemletterout)%
   \else *%
   \fi}%
```

Here are the definitions of `\numberedlist` and `\unorderedlist`. Both definitions have the same structure.

```
\def\numberedlist{\environment{@numbered-list}%
   \advance\numberedlistdepth by 1
   \itemnumber = 1 \itemletter = `a
   \beginlist \let\marker = \numberedmarker
   \def\li{%
      \ifnum\itemnumber=1\else \vskip\interitemskip \fi
      \printitem
      \advance\itemnumber by 1 \advance\itemletter by 1
   }}%
\def\itemnumberout{\number\itemnumber}%
```

```
\def\itemletterout{\char\itemletter}%
\def\endnumberedlist{\par
   \endenvironment{@numbered-list}\endlist}%


\newcount\unorderedlistdepth
\def\unorderedmarker{%
   \ifcase\unorderedlistdepth
       (impossible)%
   \or \blackbox
   \or ---%
   \else *%
   \fi}%
\def\unorderedlist{\environment{@unordered-list}%
   \advance\unorderedlistdepth by 1
   \beginlist \itemnumber = 1
   \let\marker = \unorderedmarker
   \def\li{%
      \ifnum\itemnumber=1\else \vskip\interitemskip \fi
      \printitem \advance\itemnumber by 1
   }}%
\def\endunorderedlist{\par
   \endenvironment{@unordered-list}\endlist}%
```

Verbatim listing

The `\listing` macro produces a verbatim listing of a specified file in the `\tt` font. It is based on the code on page 380 of *The TEXbook*. Tabs produce a fixed amount of space, and form feeds produce a page break. Other control characters produce whatever happens to be at that font position, which is generally not very useful. By redefining `\setuplistinghook`, you can take additional actions that are appropriate for your particular fonts and/or environment before the file is read in.

```
\def\listing#1{%
   \par \begingroup \@setuplisting \setuplistinghook
   \input #1 \endgroup}%
\let\setuplistinghook = \empty
\def\@setuplisting{%
   \uncatcodespecials
   \obeywhitespace \makeactive\‘ \makeactive\^^I
   \def^^L{\vfill\eject}\tt}%
{\makeactive\‘ \gdef‘{\relax\lq}}% Defeat ligatures.
```

```
{\makeactive\^^I\gdef^^I{\hskip8\fontdimen2\tt \relax}}%
```

Tables of contents

The macro `\writetocentry` writes a macro call to the file `\jobname.toc`. The first argument of `\writetocentry`, e.g., "chapter", is used to compose the name of the called macro. The second argument is the text to appear in the table of contents entry. `\writetocentry` appends the page number to the macro call. For example:

```
\writetocentry{chapter}{Introduction}
```

will produce the line:

```
\tocchapterentry{Introduction}{2}
```

in the `.toc` file, indicating that 'Introduction' started on page 2.

You can use `\writenumberedtocentry` to provide a third parameter, such as a chapter number. For example:

```
\writenumberedtocentry{chapter}{The second chapter}{2}
```

will write a line:

```
\tocchapterentry{The second chapter}{2}{14}
```

You can also `\write` to `\tocfile` yourself.

```
\newwrite\tocfile \newif\iftocfileopened
\def\opentocfile{\iftocfileopened\else
      \tocfileopenedtrue
      \immediate\openout\tocfile = \jobname.toc
\fi}%
\def\writetocentry#1#2{\ifrewritetocfile
   \opentocfile
   \write\tocfile{%
      \expandafter\noexpand \csname toc#1entry\endcsname
      {#2}{\folio}}%
\fi\ignorespaces}%
%
\def\writenumberedtocentry#1#2#3{\ifrewritetocfile
   \opentocfile
   \write\tocfile{%
      \expandafter\noexpand \csname toc#1entry\endcsname
      {#2}{#3}{\folio}}%
\fi\ignorespaces}%
```

To produce a table of contents, read the `.toc` file with `\readtocfile`. You should call `\readtocfile` before the first `\writetocentry`. When

you're processing the table of contents without regenerating it, you should not rewrite the `.toc` file—if you do, its contents will be lost. The command `\rewritetocfilefalse` will prevent the rewrite.

```
\newif\ifrewritetocfile \rewritetocfiletrue
\def\readtocfile{\testfileexistence{toc}%
   \iffileexists
      \input \jobname.toc
      \ifrewritetocfile \opentocfile \fi
   \fi}%
```

Here are some definitions of possible `\toc...entry` macros. These definitions are meant only as examples—running leaders across the line is usually not the best way to typeset a table of contents.

```
\def\tocchapterentry#1#2{\line{\bf #1 \dotfill\ #2}}%
\def\tocsectionentry#1#2{%
   \line{\quad\sl #1 \dotfill\ \rm #2}}%
\def\tocsubsectionentry#1#2{%
   \line{\qquad\rm #1 \dotfill\ #2}}%
```

Cross-references

The macros that follow provide symbolic cross-referencing, so that you can refer to something in another part of a document by name instead of by its actual page number. `\xrdef{foo}` defines a label `foo` to be the current page number, and `\xrefn{foo}` produces that page number, e.g., 77. More often you'll want to say something like "see p. 77", so `\xref{foo}` produces "p. 77". If `foo` is not defined, a warning message will be given. `\xrefwarningfalse` suppresses the warning.

These macros provide no protection against duplicate definitions. You can check for duplicate definitions by sorting the cross-reference file and checking, either mechanically or by eye, for adjacent definitions of the same symbol.

```
\newif\ifxrefwarning  \xrefwarningtrue
\def\xrdef#1{\begingroup
   \xrlabel{#1}%
   \edef\@wr{\@writexrdef{\the\@xrlabeltoks}}%
   \@wr
   \endgroup \ignorespaces}%
\def\@writexrdef#1{\write\reffile{%
     \string\gdef
        \expandafter\string\csname#1\endcsname
```

```
        {\noexpand\folio}\percentchar}}%
\def\xrefnumber#1{%
   \xrlabel{#1}%
   % \@xrlabeltoks now has the control sequence name.
   \toks0 =
      \expandafter{\csname\the\@xrlabeltoks\endcsname}%
   \expandafter \ifx\the\toks0\relax
      \ifxrefwarning \message{Undefined label
         `\tokstostring{#1}'.}\fi
      {\let\spacesub = \space
       \expandafter\xdef\the\toks0
          {`{\tt \tokstostring{#1}}'}}\fi
   \the\toks0}%
\def\xref#1{p.\thinspace\xrefnumber{#1}}%
\def\xrefn#1{\xrefnumber{#1}}%
```

This macro turns a label into a list of character tokens in the token register `\labeltoks`. A label can include blanks and control sequences in it as well as normal characters, but it can't include braces.

```
\newtoks\@xrlabeltoks
\def\xrlabel#1{\begingroup
      \escapechar = `\_ \edef\tts{\tokstostring{#1_}}%
      \global\@xrlabeltoks = \expandafter{\tts}%
   \endgroup}%
```

It takes two passes to get the cross-references right, since the definitions are written out to the auxiliary file `\jobname.aux`. `\readreffile` reads them back in. If you don't issue this command before the first definition, you'll lose all the definitions from the previous run.

```
\newwrite\reffile \newif\ifreffileopened
\def\openreffile{\ifreffileopened\else
      \reffileopenedtrue
      \immediate\openout\reffile = \jobname.aux
   \fi}%
\def\readreffile{%
   \testfileexistence{aux}%
   \iffileexists
      \begingroup
         \@setletters
         \input \jobname.aux
      \endgroup
   \else
      \message{No cross-reference file; I won't give you
         warnings about undefined labels.}%
      \xrefwarningfalse
```

```
    \fi
    \openreffile}%
  \def\@setletters{%
    \catcode`_ = \letter \catcode`+ = \letter
    \catcode`- = \letter \catcode`@ = \letter
    \catcode`0 = \letter \catcode`1 = \letter
    \catcode`2 = \letter \catcode`3 = \letter
    \catcode`4 = \letter \catcode`5 = \letter
    \catcode`6 = \letter \catcode`7 = \letter
    \catcode`8 = \letter \catcode`9 = \letter
    \catcode`( = \letter \catcode`) = \letter}%
```

You can give symbolic names to equations in a similar way, using `\eqdef` and `\eqref`. `\eqdef` inserts its own `\eqno` command, so it must be invoked in a place where `\eqno` is legal.

```
\newcount\eqnumber
\def\eqdef#1{\global\advance\eqnumber by 1
   \expandafter\xdef
      \csname#1eqref\endcsname{\the\eqnumber}%
   \immediate\write\reffile{\string\def
      \expandafter\string\csname#1eqref\endcsname
         {\the\eqnumber}}%
   \eqno
   \eqprint{\the\eqnumber}}%
```

`\eqref` produces "(equation number)". You can handle fancier formatting by redefining `\eqprint`. For example, you could redefine it so that the equation numbers include the chapter number.

```
\def\eqref#1{%
   \expandafter \ifx \csname#1eqref\endcsname \relax
      \ifxrefwarning \message{Undefined equation label
         `#1'.}\fi
      \expandafter\def\csname#1eqref\endcsname{00}%
   \else \eqprint{\csname#1eqref\endcsname}%
   \fi}%
\def\eqprint#1{(#1)}%
```

Environments

These macros let you define your own named groups (environments) for parts of your manuscript. Like TEX groups, these groups can be nested, and in fact their nesting can be intertwined with the nesting of TEX

groups. If the names at the beginning and end of an environment don't match, you'll get an error message. The macros are designed so that the message you get when such an error occurs will give you a good chance of localizing the cause of the error easily.

You begin an environment with `\environment {foo}` and end it with `\endenvironment{foo}`, where `foo` is the name of the environment. Our macros slightly improve on the answer to Exercise 5.7 of *The TEXbook*, by doing some checks on `\begingroup` and `\endgroup` pairs, as well as making sure `\environment` and `\endenvironment` pairs match.

```
\def\environment#1{\ifx\@groupname\undefined\else
      \errhelp = \@unnamedendgrouphelp
      \errmessage{`\@groupname' was not closed by
         \string\endenvironment}\fi
   \def\@groupname{#1}%
   \begingroup
      \let\@groupname = \undefined \ignorespaces}%
\def\endenvironment#1{\endgroup
   \def\@thearg{#1}%
   \ifx\@groupname\@thearg
   \else
      \ifx\@groupname\undefined
         \errhelp = \@isolatedendenvironmenthelp
         \errmessage{Isolated
            \string\endenvironment\space for `#1'}%
      \else
         \errhelp = \@mismatchedenvironmenthelp
         \errmessage{Environment `#1' ended,
            but `\@groupname' started}%
         \endgroup % Probably a typo in the names.
      \fi
   \fi
   \let\@groupname = \undefined \ignorespaces}%
```

We also define help messages for each of the errors above.

```
\newhelp\@unnamedendgrouphelp{%
   Most likely, you just forgot an^^J%
   \string\endenvironment.
   Maybe you should try inserting another^^J%
   \string\endgroup to recover.}%
\newhelp\@isolatedendenvironmenthelp{%
   You ended an environment X, but^^J%
   no \string\environment\space to start it
   is anywhere in sight.^^J%
   You might also be at an
```

```
    \string\endenvironment\space that would match^^J%
    a \string\begingroup, i.e., you forgot an
    \string\endgroup.}%
\newhelp\@mismatchedenvironmenthelp{%
    You started an environment X, but^^J%
    you ended it with Y.  Maybe you made a typo
    in one or the other^^J%
    of the names.}%
```

Some environments should not be allowed to occur within another environment. Let's call these environments "outer environments". `\checkenv` checks that no outer environment is currently in effect and complains if one is. To use `\checkenv`, you must issue the command `\environmenttrue` at the beginning of every outer environment.

```
\newif\ifenvironment
\def\checkenv{%
    \ifenvironment
        \errhelp = \@interwovenenvhelp
        \errmessage{Interwoven environments}%
        \endgroup
    \fi}%
\newhelp\@interwovenenvhelp{%
    Perhaps you forgot to end the previous^^J%
    environment? I'm finishing off the current group,^^J%
    hoping that will fix it.}%
```

Justification

The three macros `\flushleft`, `\flushright`, and `\center` justify the text on the following lines in the indicated way. The command should appear on a line by itself. Both the command and the text should be enclosed in a group—the end of the group indicates the end of the text. The entire group is set as a single paragraph, with lines filled out on one side or another as appropriate. Blank lines are reproduced.

```
\begingroup
    \catcode `\^^M = \active
    \globaldefs = 1 %
    \def\flushleft{\beforejustify %
        \aftergroup\@endflushleft %
        \def^^M{\null\hfil\break}%
        \def\@eateol^^M{}\@eateol}%
    \def\flushright{\beforejustify %
```

```
      \aftergroup\@endflushright %
      \def^^M{\break\null\hfil}%
      \def\@eateol^^M{\hfil\null}\@eateol}%
   \def\center {\beforejustify %
      \aftergroup\@endcenter %
      \def^^M{\hfil\break\null\hfil}%
      \def\@eateol^^M{\hfil\null}\@eateol}%
\endgroup
```

The following commands are called as a result of the `\aftergroup` in the definitions of `\flushleft`, `\flushright`, and `\center`. They perform the necessary cleanup operations.

```
\def\@endflushleft{\unpenalty
   {\parfillskip = 0pt plus 1 fil\par}%
   \ignorespaces}%
\def\@endflushright{%
   % Remove the \hfil\null\break we just put on.
   \unskip \setbox0=\lastbox \unpenalty
   % We have fil glue at the left of the line;
   % \parfillskip shouldn't affect that.
   {\parfillskip = 0pt \par}\ignorespaces}%
\def\@endcenter{%
   % Remove the \hfil\null\break we just put on.
   \unskip \setbox0=\lastbox \unpenalty
   % We have fil glue at the left of the line;
   % \parfillskip must balance it.
   {\parfillskip = 0pt plus 1fil \par}\ignorespaces}%
\def\beforejustify{%
   \par\noindent
   \catcode`\^^M = \active
   \checkenv \environmenttrue}%
```

Tables

The `\makecolumns` macro enables you to give all the entries in a table without having to worry about where the columns break. For example, if you're typing a long alphabetical list that will be formatted in several columns, you usually won't know in advance where one column ends and the next begins. Moreover, if another item gets added, the column breaks will change.

`\makecolumns` takes two (delimited) arguments: the total number of entries in the table and the number of columns in the table. For example,

'`\makecolumns 37/3:`' specifies a three-column table whose entries are the next 37 lines. You can adjust the positioning of the table on the page by changing `\parindent`, which determines the space to the left, and `\hsize`, which determines the space from the left margin of the page to the right of the block. You can allow a page break above the `\valign` by changing `\abovecolumnspenalty`.

```
\newcount\abovecolumnspenalty
\abovecolumnspenalty = 10000
\newcount\@linestogo      % Lines remaining to process.
\newcount\@linestogoincolumn % Lines remaining in column.
\newcount\@columndepth    % Number of lines in a column.
\newdimen\@columnwidth    % Width of each column.
\newtoks\crtok  \crtok = {\cr}%
\def\makecolumns#1/#2: {\par \begingroup
   \@columndepth = #1 \advance\@columndepth by #2
   \advance\@columndepth by -1
   \divide \@columndepth by #2
   \@linestogoincolumn = \@columndepth \@linestogo = #1
   \def\@endcolumnactions{%
      \ifnum \@linestogo<2
         \the\crtok \egroup \endgroup \par
            % End \valign and \makecolumns.
      \else
         \global\advance\@linestogo by -1
         \ifnum\@linestogoincolumn<2
            \global\@linestogoincolumn = \@columndepth
            \the\crtok
         \else &\global\advance\@linestogoincolumn by -1
         \fi
      \fi}%
   \makeactive\^^M\letreturn\@endcolumnactions
   \@columnwidth = \hsize
   \advance\@columnwidth by -\parindent
   \divide\@columnwidth by #2
   \penalty\abovecolumnspenalty
   \noindent % It's not a paragraph (usually).
   \valign\bgroup
      &\hbox to \@columnwidth{\strut ##\hfil}\cr
}% The next end-of-line starts everything going.
```

Footnotes

Footnotes are most commonly typeset by using a raised number as the reference mark. We define the `\numberedfootnote` macro to do this. It also redefines `\vfootnote` to allow slightly more general formatting of footnotes than plain TEX does. The dimension register `\footnotemarkseparation` controls the space between the footnote mark (e.g., the number) and the beginning of the text. The `\everyfootnote` tokens are inserted before producing the footnote.

The plain TEX definitions of `\footnote` and `\vfootnote` are preserved in `\@plainfootnote` and `\@plainvfootnote` in case you should need them.

```
\newcount\footnotenumber \newtoks\everyfootnote
\newdimen\footnotemarkseparation
\footnotemarkseparation = .5em
\let\@plainfootnote = \footnote
\let\@plainvfootnote = \vfootnote
\def\vfootnote#1{\insert\footins\bgroup
  \interlinepenalty\interfootnotelinepenalty
  \splittopskip\ht\strutbox \splitmaxdepth\dp\strutbox
  \floatingpenalty\@MM
  \leftskip\z@skip \rightskip\z@skip \spaceskip\z@skip
  \xspaceskip\z@skip
  \everypar = {}%
  \the\everyfootnote
  \indent\llap{#1\kern\footnotemarkseparation}\footstrut
  \futurelet\next\fo@t}%
\def\numberedfootnote{\global\advance\footnotenumber by 1
  \@plainfootnote{$^{\number\footnotenumber}$}}%
```

Double columns

The `\doublecolumns` command begins double-column output, while the `\singlecolumn` command restores single-column output. You can switch back and forth between them on a single page. The glue specified by `\abovedoublecolumnskip` and `\belowdoublecolumnskip` is inserted before and after the double-column material.

The approach is derived from page 417 of *The TEXbook*.

```
\newskip\abovedoublecolumnskip
```

```
\newskip\belowdoublecolumnskip
\abovedoublecolumnskip = \bigskipamount
\belowdoublecolumnskip = \bigskipamount
\newdimen\gutter            \gutter = 2pc
\newdimen\doublecolumnhsize \doublecolumnhsize = \hsize
\newbox\@partialpage        \newdimen\singlecolumnhsize
\newdimen\singlecolumnvsize \newtoks\previousoutput
\def\doublecolumns{\par % Don't start in horizontal mode.
   \previousoutput = \expandafter{\the\output}
   \advance\doublecolumnhsize by -\gutter
   \divide\doublecolumnhsize by 2
   \output = {\global\setbox\@partialpage =
         \vbox{\unvbox255\vskip\abovedoublecolumnskip}}%
   \pagegoal = \pagetotal \break % Expands \output above.
   \output = {\doublecolumnoutput}%
   \singlecolumnhsize = \hsize
   \singlecolumnvsize = \vsize
   \hsize = \doublecolumnhsize \vsize = 2\vsize}%
```

The `\@doublecolumnsplit` macro does the actual splitting. Insertions are assumed to be single-column material. If you don't want this to be the case, you'll have to modify the output routine. After `\@doublecolumnsplit` has done its work, `\box255` will have the double-column material. The double-column material will be preceded by any single-column material that was typeset before `\doublecolumns` was invoked. `\box4` will have the material that didn't fit on the page.

```
\def\@doublecolumnsplit{%
   \splittopskip = \topskip \splitmaxdepth = \maxdepth
   \dimen0 = \singlecolumnvsize
      \advance\dimen0 by -\ht\@partialpage
      \advance\dimen0 by -\ht\footins
      \advance\dimen0 by -\skip\footins
      \advance\dimen0 by -\ht\topins
   \begingroup
      \vbadness = 10000
      \global\setbox1=\vsplit255 to \dimen0 \wd1=\hsize
      \global\setbox3=\vsplit255 to \dimen0 \wd3=\hsize
   \endgroup
   \global\setbox4=\vbox{\unvbox255
      \penalty\outputpenalty}%
   \global\setbox255=\vbox{\unvbox\@partialpage
      \hbox to \singlecolumnhsize{\box1\hfil\box3}%
      \vfill}}%
```

`\doublecolumnoutput` is the real output routine. We call the old `\output` to do the work of actually shipping out the box.

```
\def\doublecolumnoutput{\@doublecolumnsplit
   \hsize = \singlecolumnhsize \vsize = \singlecolumnvsize
   \previousoutput \unvbox4}%
```

`\singlecolumn` resumes typesetting in one column. It assumes that `\doublecolumns` has been called.

```
\def\singlecolumn{\par % Don't start in horizontal mode.
   \output = {\global\setbox1 =
      \vbox{\unvbox255\vskip\abovedoublecolumnskip}}%
   \pagegoal = \pagetotal \break \setbox255 = \box1
   {\singlecolumnvsize = \ht255
      \divide\singlecolumnvsize by 2
      \advance\singlecolumnvsize by +\ht\@partialpage
      \advance\singlecolumnvsize by +\ht\footins
      \advance\singlecolumnvsize by +\skip\footins
      \advance\singlecolumnvsize by +\ht\topins
   \@doublecolumnsplit}%
   \hsize = \singlecolumnhsize
   \vsize = \singlecolumnvsize
   \output = \expandafter{\the\previousoutput}%
   \unvbox255}%
```

Finishing up

We now must undo the changes that we made when we started (see p. 'eplainconv'). We also give a version identification, which is subsequently available in `\fmtname` and `\fmtversion`.

```
\let\wlog = \@plainwlog \catcode`@ = \other
\def\fmtname{eplain}%
{\edef\plainversion{\fmtversion}%
 \xdef\fmtversion{1.0: 15 May 1990
   (and plain \plainversion)}%
}%
```