1 Tips and techniques

TEX is a complex program that occasionally works its will in mysterious ways. In this section we offer some tips on solving problems that you might encounter and explain some handy techniques.

Correcting bad page breaks

Sometimes TEX breaks a page right in the middle of material that you want to keep together—for example, a section heading and the text that follows it, or a short list of related items. There are two ways to correct the situation:

- You can force the material to be kept together.
- You can force a page break at a different place.

The simplest way to force TEX to keep material together on a page is to enclose the material in a vbox using the `\vbox` command (p. '`\vbox`'). A vbox is ordinarily better than an hbox for this purpose because most often the material to be kept together, e.g., a sequence of paragraphs, will be vertical mode material. You should precede and follow the vbox by an implicit or explicit paragraph command (either a blank line or `\par`); otherwise TEX may try to make the vbox part of an an adjacent paragraph. The vbox method has an important limitation: you can't apply it to portions of text smaller than a paragraph.

You can sometimes keep the lines of a single paragraph together by enclosing the paragraph in a group and assigning `\interlinepenalty` (p. '`\interlinepenalty`') a value of 10000 at the start of the group (or elsewhere before the end of the paragraph). This method causes TEX to

consider page breaks within that paragraph to be infinitely undesirable. However, if all the page breaks that TEX can find are infinitely undesirable, it may break the page within the paragraph anyway.

A `\nobreak` command (p. '`vnobreak`') after the end of a paragraph prevents TEX from breaking the page at the following item (unless that item happens to be a penalty of less than 10000). This is also the best way to prevent a page break after a heading, since a heading usually behaves like a paragraph. The `\nobreak` must follow the blank line or `\par` that ends the paragraph so that TEX won't treat the `\nobreak` as part of the paragraph. For the `\nobreak` to be effective, it must also come before any legal breakpoint at the end of the paragraph. The glue that TEX inserts before the next paragraph is such a breakpoint, and so is any vertical glue that you insert explicitly after a paragraph. Thus the `\nobreak` should usually be the very first thing after the end of the paragraph or heading.

You can use the `\eject` command (p. '`\eject`') to force TEX to break a page at a particular place. Within a paragraph, you can use the combination '`\vadjust{\vfill\eject}`' (p. '`\vadjust`') to force a break after the next complete output line. The reason for preceding `\eject` by `\vfill` (p. '`\vfill`') is to get TEX to fill out the page with blank space. However, using `\eject` to fix page break problems has a major disadvantage: if the page boundaries in your document change, the page breaks that you've inserted may no longer be where you want them.

If you don't provide TEX with a `\vfill` command to fill out the page before an `\eject`, TEX redistributes the extra blank space as best it can and then usually complains that "an underfull `\vbox` (badness 10000) has occurred while `\output` is active." You may encounter a similar problem with any of the methods mentioned above for enclosing material that you want to keep together.

The `\filbreak` command (p. '`\filbreak`') provides a way of keeping the lines of one or more paragraphs (or other vertical mode material) together on a page. If you enclose a paragraph in `\filbreak`s, TEX will effectively ignore the `\filbreak`s if the paragraph fits on the current page and break the page before the first `\filbreak` if the paragraph doesn't fit. If you put `\filbreak`s around each paragraph in a sequence of paragraphs, like this:

```
\filbreak
⟨paragraph⟩
\filbreak
⟨paragraph⟩
\filbreak
  ⋮
⟨paragraph⟩
\filbreak
```

TEX will keep the lines of each paragraph together on a page. If TEX breaks a page at a `\filbreak`, it will fill the bottom of the page with blank space.

Sometimes you can get TEX to modify the length of a page by changing the `\looseness` parameter (p. '`\looseness`') for one or more paragraphs. Setting `\looseness` negative within a paragraph causes TEX to try to squeeze the paragraph into fewer lines; setting it positive causes TEX to try to expand the paragraph into more lines. The disadvantage of changing `\looseness` is that the interword spacing in the affected region won't be optimal. You can get further information about TEX's attempted line breaks by setting `\tracingpages` (p. '`\tracingpages`') to 1.

Preserving the end of a page

Sometimes you need to modify something on a single page and you want to avoid reprinting the entire document. If your modification doesn't change the page length too much, there's hope. You need to fix the end of the page so that it falls in the same place; the methods are similar to the ones for fixing a bad page break.

If the original end of page came between paragraphs, you can force a page break at the same place using any of the methods we've described above. Otherwise, you must force *both* a line break and a page break at a particular place. If the new page is shorter than the old one, the sequence:

```
\vadjust{\vfill\eject}\break
```

should do the trick. But if the new page is longer, the problem is far more difficult because TEX has probably already squeezed the page as tightly as it can. Your only hopes in this case are to set `\looseness` (p. '`\looseness`') to a negative value, to shorten some of the vertical skips on the page, to add some shrink to `\parskip` (p. '`\parskip`') if it was nonzero, or, as a last resort, to decrease `\baselineskip` (p. '`\baselineskip`') ever so slightly.

Leaving space at the top of a page

You can usually use the `\vskip` command (p. '`\vskip`') to leave vertical space on a page. That doesn't work at the top of a page, however, since TEX discards glue, kerns, and penalties that occur just after a page break.

Use the \topglue command (p. '\topglue') instead; it produces glue that never disappears.

Correcting bad line breaks

If TEX breaks a line in the middle of material that you wanted to keep on a single line, there are several ways to correct the situation:

- You can force a break in a nearby place with the \break command (p. 'hbreak').
- You can insert a tie (~) between two words (see p. '@not') to prevent a break between them.
- You can tell TEX about hyphenations that it wouldn't otherwise consider by inserting one or more discretionary hyphens in various words (see \-, p. '\@minus').
- You can enclose several words in an hbox using the \hbox command (p. '\hbox').

The disadvantage of all of these methods, except for inserting discretionary hyphens, is that they may make it impossible for TEX to find a satisfactory set of line breaks. Should that happen, TEX will set one or more underfull or overfull boxes and complain about it. The hbox method has a further disadvantage: because TEX sets an hbox as a single unit without considering its context, the interword space within the hbox may not be consistent with the interword space in the rest of the line.

Correcting overfull or underfull boxes

If TEX complains about an overfull box, it means you've put more material into a box than that box has room for. Similarly, if TEX complains about an underfull box, it means you haven't put enough material into the box. You can encounter these complaints under many different circumstances, so let's look at the more common ones:

- An overfull hbox that's a line of a paragraph indicates that the line was too long and that TEX couldn't rearrange the paragraph to make the line shorter. If you set \emergencystretch (p. '\emergencystretch') to some nonzero value, that may cure the problem by allowing TEX to put more space between words. Another solution is to set \tolerance (p. '\tolerance') to 10000, but that's likely to yield lines with far too much space in them. Yet another solution is to insert a discretionary hyphen in a critical word that TEX didn't know how to hyphenate. If

all else fails, you might try rewording the paragraph. A solution that is rarely satisfactory is increasing `\hfuzz` (p. '`\hfuzz`'), allowing TEX to construct lines that project beyond the right margin.

- An underfull hbox that's a line of a paragraph indicates that the line was too short and that TEX couldn't rearrange the paragraph to make the line longer. TEX will set such a line by stretching its interword spaces beyond their normal limits. Two of the cures for overfull lines mentioned above also apply to underfull lines: inserting discretionary hyphens and rewording the paragraph. Underfull lines won't trouble you if you're using ragged right formatting, which you can get with the `\raggedright` command (p. '`\raggedright`').
- The complaint:

```
Underfull \vbox (badness 10000) has occurred
   while \output is active
```

  indicates that TEX didn't have enough material to fill up a page. The likely cause is that you've been using vboxes to keep material together and TEX has encountered a vbox near the bottom of a page that wouldn't fit on that page. It has put the vbox on the next page, but in doing so has left too much empty space in the current page. In this case you'll either have to insert some more space elsewhere on the current page or break up the vbox into smaller parts.

  Another possible cause of this complaint is having a long paragraph that occupies an entire page without a break. Since TEX won't ordinarily vary the spacing between lines, it may be unable to fill a gap at the bottom of the page amounting to a fraction of the line spacing. This can happen if `\vsize` (p. '`\vsize`'), the page length, is not an even multiple of `\baselineskip` (p. '`\baselineskip`'), the space between consecutive baselines.

  Yet another cause of this complaint, similar to the previous one, is setting `\parskip` (p. '`\parskip`'), the interparagraph glue, to a value that doesn't have any stretch or shrink. You can fix these last two problems by increasing `\vfuzz` (p. '`\vfuzz`').
- The complaint

```
Overfull \vbox (296.30745pt too high) has occurred
   while \output is active
```

  indicates that you constructed a vbox that was longer than the page. You'll just have to make it shorter.
- The only cures for an overfull hbox or vbox that you've constructed with the `\hbox` or `\vbox` commands (pp. '`\hbox`', '`\vbox`') are to take something out of the box, to insert some negative glue with `\hss` or `\vss` (p. '`\hss`'), or to increase the size of the box.

- If you encounter an underfull hbox or vbox that you've constructed with `\hbox` or `\vbox`, you're usually best off to fill out the box with `\hfil` or `\vfil` (p. '`\hfil`').

Recovering lost interword spaces

If you find that TEX has run two words together, the likely cause is a control sequence that's absorbed the spaces after it. Put a control space (`\␣`) after the control sequence.

Avoiding unwanted interword spaces

If you get a space in your document where you don't want and don't expect one, the most likely cause, in our experience, is an end of line or a space following a brace. (If you're doing fancy things with category codes, you've introduced lots of other likely causes.) TEX ordinarily translates an end-of-line into a space, and it considers a space after a right or left brace to be significant.

If the unwanted space is caused by a space after a brace within an input line, then remove that space. If the unwanted space is caused by a brace at the end of an input line, put a '`%`' immediately after the brace. The '`%`' starts a comment, but this comment needn't have any text.

A macro definition can also introduce unwanted spaces if you haven't written it carefully. If you're getting unwanted spaces when you call a macro, check its definition to be sure that you don't have an unintended space after a brace and that you haven't ended a line of the definition immediately after a brace. People often end lines of macro definitions after braces in order to make the definitions more readable. To be safe, put a '`%`' after any brace that ends a line of a macro definition. It may not be needed, but it won't do any harm.[1]

When you're having trouble locating the source of an unwanted space, try setting `\tracingcommands` (p. '`\tracingcommands`') to 2. You'll get a `{blank space}` command in the log file for each space that TEX sees.

It helps to know TEX's rules for spaces:

1) Spaces are ignored at the beginnings of input lines.
2) Spaces at the ends of input lines are ignored under *all* circumstances, although the end of line itself is treated like a space. (A completely blank line, however, generates a `\par` token.)

[1] Admittedly there are rare cases where you really do want an end of line after a brace.

3) Multiple spaces are treated like a single space, but only if they appear together in your input. Thus a space following the arguments of a macro call is not combined with a final space produced by the macro call. Instead, you get two spaces.
4) Spaces are ignored after control words.
5) Spaces are in effect ignored after numbers, dimensions, and the '`plus`' and '`minus`' in glue specifications.[2]

If you've changed the category code of the space or the end-of-line character, all bets are off.

Avoiding excess space around a display

If you're getting too much space above a math display, it may be because you've left a blank line in your input above the display. The blank line starts a new paragraph and puts TEX into vertical mode. When TEX sees a '`$`' in vertical mode, it switches back to horizontal mode and inserts the interparagraph glue (`\parskip`) followed by the interline glue (`\baselineskip`). Then, when it starts the display itself, it inserts *more* glue (either `\abovedisplayskip` or `\abovedisplayshortskip`, depending on the length of the preceding line). This last glue is the only glue that you want. To avoid getting the interparagraph glue as well, don't leave a blank line above a math display or otherwise end a paragraph (with `\par`, say) just before a math display.

Similarly, if you're getting too much space below a math display, it may be because you've left a blank line in your input below the display. Just remove it.

Avoiding excess space after a paragraph

If you get too much vertical space after a paragraph that was produced by a macro, you may be getting the interparagraph glue produced by the macro, an empty paragraph, and then more interparagraph glue. You can get rid of the second paragraph skip by inserting:

```
\vskip -\parskip
\vskip -\baselineskip
```

[2] Actually, TEX ignores only a single space in these places. Since multiple spaces ordinarily reduce to a single space, however, the effect is that of ignoring any number of spaces.

just after the macro call. If you always get this problem with a certain macro, you can put these lines at the end of the macro definition instead. You may also be able to cure the problem by never leaving a blank line after the macro call—if you want a blank line just to make your input more readable, start it with a '`%`'.

Changing the paragraph shape

Several TEX parameters—`\hangindent`, `\leftskip`, etc.—affect the way that TEX shapes paragraphs and breaks them into lines. These parameters are used indirectly in plain TEX commands such as `\narrower` and `\hang`; you can also assign to them directly. If you've used one of these commands (or changed one of these parameters), but the command or parameter change does not seem to be having any effect on a paragraph, the problem may be that you've ended a group before you've ended the paragraph. For example:

```
{\narrower She very soon came to an open field, with
a wood on the other side of it: it looked much darker
than the last wood, and Alice felt a little timid
about going into it.}
```

This paragraph won't be set narrower because the right brace at the end terminates the `\narrower` group before TEX has had a chance to break the paragraph into lines. Instead, put a `\par` before the right brace; then you'll get the effect you want.

Putting paragraphs into a box

Suppose you have a few paragraphs of text that you want to put in a particular place on the page. The obvious way to do it is to enclose the paragraphs in an hbox of an appropriate size, and then place the hbox where you want it to be. Alas, the obvious way doesn't work because TEX won't do line breaking in restricted horizontal mode. If you try it, you'll get a misleading error message that suggests you're missing the end of a group. The way around this restriction is to write:

`\vbox{\hsize =` ⟨*dimen*⟩ ... ⟨*paragraphs*⟩ ...`}`

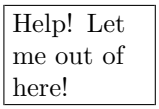
where ⟨*dimen*⟩ is the line length that you want for the paragraphs. This is what you need to do, in particular, when you want to enclose some paragraphs in a box (a box enclosed in ruled lines, not a TEX box).

Drawing lines

You can use the `\hrule` and `\vrule` commands (p. '`\hrule`') to draw lines, i.e., rules. You'll need to know (a) where you can use each command and (b) how TEX determines the lengths of rules when you haven't given the lengths explicitly.

- You can only use `\hrule` when TEX is in a vertical mode and `\vrule` when TEX is in a horizontal mode. This requirement means that you can't put a horizontal rule into an hbox or a vertical rule into a vbox. You can, however, construct a horizontal rule that looks vertical by specifying all three dimensions and making it tall and skinny. Similarly, you can construct a vertical rule that looks horizontal by making it short and fat.
- A horizontal rule inside a vbox has the same width as does the vbox if you haven't given the width of the rule explicitly. Vertical rules inside hboxes behave analogously. If your rules are coming out too long or too short, check the dimensions of the enclosing box.

As an example, suppose we want to produce:



The following input will do it:

```
\hbox{\vrule
   \vbox{\hrule \vskip 3pt
      \hbox{\hskip 3pt
          \vbox{\hsize = .7in \raggedright
             \noindent Help! Let me out of here!}%
      \hskip 3pt}%
   \vskip 3pt \hrule}%
\vrule}
```

We need to put the text into a vbox in order to get TEX to process it as a paragraph. The four levels of boxing are really necessary—if you doubt it, try to run this example with fewer levels.

Creating multiline headers or footers

You can use the `\headline` and `\footline` commands (p. '`\footline`') to produce headers and footers, but they don't work properly for headers and footers having more than one line. However, you can get multiline headers and footers by redefining some of the subsidiary macros in TEX's output routine.

For a multiline header, you need to do three things:

1) Redefine the `\makeheadline` macro that's called from TEX's output routine.
2) Increase `\voffset` by the amount of vertical space consumed by the extra lines.
3) Decrease `\vsize` by the same amount.

The following example shows how you might do this:

```
\advance\voffset by 2\baselineskip
\advance\vsize by -2\baselineskip
\def\makeheadline{\vbox to 0pt{\vss\noindent
   Header line 1\hfil Page \folio\break
   Header line 2\hfil\break
   Header line 3\hfil}%
   \vskip\baselineskip}
```

You can usually follow the pattern of this definition quite closely, just substituting your own header lines and choosing an appropriate multiple of `\baselineskip` (one less than the number of lines in the header).

For a multiline footer, the method is similar:

1) Redefine the `\makefootline` macro that's called from TEX's output routine.
2) Decrease `\vsize` by the amount of vertical space consumed by the extra lines.

The following example shows how you might do this:

```
\advance\vsize by -2\baselineskip
\def\makefootline{%
   \lineskip = 24pt
   \vbox{\raggedright\noindent
   Footer line 1\hfil\break
   Footer line 2\hfil\break
   Footer line 3\hfil}}
```

Again, you can usually follow the pattern of this definition quite closely. The value of `\lineskip` determines the amount of space between the

baseline of the last line of the main text on the page and the baseline of the first line of the footer.

Finding mismatched braces

Most times when your TEX input suffers from mismatched braces, you'll get a diagnostic from TEX fairly near the place where you actually made the mistake. But one of the most frustrating errors you can get from a TEX run, just before TEX quits, is the following:

```
(\end occurred inside a group at level 1)
```

This indicates that there is an extra left brace or a missing right brace somewhere in your document, but it gives you no hint at all about where the problem might be. So how can you find it?

A debugging trick we've found useful is to insert the following line or its equivalent at five or six places equally spaced within the document (and not within a known group):

```
}% a fake ending
```

Let's assume the problem is an extra left brace. If the extra left brace is, say, between the third and fourth fake ending, you'll get error messages from the first three fake endings but not from the fourth one. The reason is that TEX will ignore the first three fake endings after complaining about them, but the fourth fake ending will match the extra left brace. Thus you know that the extra left brace is somewhere between the third and fourth fake ending. If the region of the error is still too large for you to find it, just remove the original set of fake endings and repeat the process within that region. If the problem is a missing right brace rather than an extra left brace, you should be able to track it down once you've found its mate.

This method doesn't work under all circumstances. In particular, it doesn't work if your document consists of several really large groups. But often you can find some variation on this method that will lead you to that elusive brace.

If all else fails, try shortening your input by removing the last half of the file (after stashing away the original version first!) or inserting a `\bye` command in the middle. If the error persists, you know it's in the first half; if it goes away, you know it's in the second half. By repeating this process you'll eventually find the error.

Setting dimensions

The simplest way to set a dimension is to specify it directly, e.g.:

```
\hsize = 6in
```

You can also specify a dimension in terms of other dimensions or as a mixture of different units, but it's a little more work. There are two ways to construct a dimension as such a combination:

1) You can add a dimension to a dimension parameter or to a dimension register. For example:

   ```
   \hsize = 6in \advance\hsize by 3pc % 6in + 3pc
   ```

2) You can indicate a dimension as a multiple of a dimension or glue parameter or register. In this case, TEX converts glue to a dimension by throwing away the stretch and shrink. For example:

   ```
   \parindent = .15\hsize
   \advance\vsize by -2\parskip
   ```

Creating composite fonts

It's sometimes useful to create a "composite font", named by a control sequence $\mathcal{F}$, in which all the characters are taken from a font f_1 except for a few that are borrowed from another font f_2. You can then set text in the composite font by using $\mathcal{F}$ just as you'd use any other font identifier.

You can create such a composite font by defining $\mathcal{F}$ as a macro. In the definition of $\mathcal{F}$, you first select font f_1 and then define control sequences that produce the borrowed characters, set in f_2. For example, suppose that you want to create a composite font `\britrm` which has all the characters of `cmr10` except for the dollar sign, for which you want to borrow the pound sterling symbol from font `cmti10`. The pound sterling symbol in `cmti10` happens to be in the same font position as the dollar sign in `cmr10`. Here's how to do it:

```
\def\britrm{%
   \tenrm % \tenrm names the cmr10 font
   \def\${{\tenit\char `\$}}% \tenit names the cmti10 font.
}
```

Now whenever you start the font named `\britrm`, `\$` will produce a pound sterling symbol.

You can also get the same effect by changing the category codes of the characters in question to make those characters active and then providing a definition for the character. For example:

```
\catcode `* = \active
\def*{{\tentt \char `\*}}
```

In this case the asterisk will be taken from the `\tentt` font. If you then type the input line:

```
Debbie was the * of the show.
```

it will be set as:

Debbie was the * of the show.

Reproducing text verbatim

Verbatim text is text that is reproduced in a typeset document just as it appeared in the input. The most common use of verbatim text is in typesetting computer input, including both computer programs and input to TEX itself. Computer input is not easy to produce verbatim for two reasons:

1) Some characters (control symbols, escape characters, braces, etc.) have special meanings to TEX.
2) Ends of line and multiple spaces are translated to single spaces.

In order to produce verbatim text, you have to cancel the special meanings and disable the translation. This is best done with macros.

To cancel the special meanings, you need to change the category codes of those characters that have special meanings. The following macro illustrates how you might do it:

```
\chardef \other = 12
\def\deactivate{%
   \catcode`\\ = \other   \catcode`\{ = \other
   \catcode`\} = \other   \catcode`\$ = \other
   \catcode`\& = \other   \catcode`\# = \other
   \catcode`\% = \other   \catcode`\~ = \other
   \catcode`\^ = \other   \catcode`\_ = \other
}
```

But beware! Once you've changed the category codes in this way, you've lost the ability to use control sequences since there's no longer an escape character. You need some way of getting back to the normal mode of operation. We'll explain how to do that in a moment, after considering the other problem: disabling the translation of spaces and ends of line.

Plain TEX has two commands that together nearly solve the problem: `\obeyspaces` (p. '`\obeyspaces`') and `\obeylines` (p. '`\obeylines`'). The two things that they don't do are to preserve spaces at the start of a line and to preserve blank lines. For that you need stronger measures—which are provided by the `\obeywhitespace` macro that we are about to define.

TEX normally insists on collecting lines into paragraphs. One way to convince it to take line boundaries literally is to turn individual lines into paragraphs.[3] You can do this by redefining the end of line character to produce the `\par` control sequence. The following three macro definitions show how:

```
\def\makeactive#1{\catcode`#1 = \active \ignorespaces}
{% The group delimits the text over which ^^M is active.
   \makeactive\^^M %
   \gdef\obeywhitespace{%
   % Use \gdef so the definition survives the group.
      \makeactive\^^M %
      \let^^M = \newline %
      \aftergroup\removebox % Kill extra paragraph at end.
      \obeyspaces %
   }%
}
\def\newline{\par\indent}
\def\removebox{\setbox0=\lastbox}
```

A subtle point about the definition of `\obeywhitespace` is that `^^M` must be made active both when `\obeywhitespace` is being *defined* and when it is being *used*.

In order to be able to get back to normal operation after verbatim text, you need to choose a character that appears rarely if at all in the verbatim text. This character serves as a temporary escape character. The vertical bar (`|`) is sometimes a good choice. With this choice, the macros:

```
\def\verbatim{\par\begingroup\deactivate\obeywhitespace
   \catcode `\| = 0 % Make | the new escape character.
}

\def\endverbatim{\endgroup\endpar}

\def\||{|}
```

will do the trick. Within the verbatim text, you can use a double vertical bar (`||`) to denote a single one, and you end the verbatim text with `|endverbatim`.

[3] Another way is to turn the end of line character into a `\break` command and provide infinite glue at the end of each line.

There are many variations on this technique:

- If a programming language has keywords, you can turn each keyword into a command that typesets that keyword in boldface. Each keyword in the input should then be preceded by the temporary escape character.
- If you have a character (again, let's assume it's the vertical bar) that *never* appears in the verbatim text, you can make it active and cause it to end the verbatim text. The macro definitions then go like this:

```
{\catcode `\| = \active
\gdef\verbatim{%
   \par\begingroup\deactivate\obeywhitespace
   \catcode `| = \active
   \def |{\endgroup\par}%
}}
```

The ideas presented here provide only a simple approach to typesetting computer programs. Verbatim reproduction is often not as revealing or easy to read as a version that uses typographical conventions to reflect the syntax and even the semantics of the program. If you'd like to pursue this subject further, we recommend the following book:

Baecker, Ronald M., and Marcus, Aaron, *Human Factors and Typography for More Readable Programs*. Reading, Mass.: Addison-Wesley, 1990.

Using outer macros

If TEX complains about a "forbidden control sequence", you've probably used an outer macro in a non-outer context (see "outer", p. `outer`). An outer macro is one whose definition is preceded by `\outer`. An outer macro can't be used in a macro argument, in a macro definition, in the preamble of an alignment, or in conditional text, i.e., text that will be expanded only when a conditional test has a particular outcome. Certain macros have been defined as outer because they aren't intended to be used in these contexts and such a use is probably an error. The only ways around this problem are to redefine the macro or to move its use to an acceptable context.

Using an outer macro in an improper context can also cause TEX to complain about a runaway situation or an incomplete conditional. The problem can be hard to diagnose because the error message gives no hint as to what it is. If you get such an error message, look around for a call on an outer macro. You may not always know that a particular

macro is outer, but the command '`\show\a`' (p. '`\show`') will show you the definition of `\a` and also tell you if `\a` is outer.

Changing category codes

Sometimes it's useful to make local changes to the category code of a character in some part of your document. For instance, you might be typesetting a computer program or something else that uses normally active characters for special purposes. You'd then want to deactivate those characters so that TEX will treat them as being like any other character.

If you make such a local change to the category code of a character, you may sometimes be dismayed to find that TEX seems to be paying no attention whatsoever to your change. Two aspects of TEX's behavior are likely causes:

1) TEX determines the category code of an input character and attaches it to the character when it reads in the character. Let's say you read in a tilde (`~`) and later change the category code of tildes, but make the change before TEX's stomach has actually processed that *particular* tilde (see "anatomy of TEX", p. '`\anatomy`'). TEX will still respond to that tilde using the category code as it was before the change. This difficulty typically arises when the tilde is part of an argument to a macro and the macro itself changes the category code of tilde.
2) When TEX is matching a call of a macro to the definition of that macro, it matches not just the characters in the parameter pattern but also their category codes. If the category code of a pattern character isn't equal to the category code of the same character in the call, TEX won't consider the characters as matching. This effect can produce mysterious results because it *looks* as though the pattern should match. For example, if you've defined a macro:

   ```
   \def\eurodate#1/#2/#3{#2.#1.#3}
   ```

   then the slash character must have the same category code when you call `\eurodate` as it had when you defined `\eurodate`.

If the problem arises because the troublesome character is an argument to a macro, then the usual cure is to redefine the macro as a pair of macros `\mstart` and `\mfinish`, where `\mstart` is to be called before the argument text and `\mfinish` is to be called after it. `\mstart` then sets up the category codes and `\mfinish` undoes the change, perhaps just by ending a group.

Making macro files more readable

You can make a file of macros more readable by setting the category codes of space to 9 (ignored character) and `\endlinechar` (p. '`\endlinechar`') to −1 at the beginning of the file. Then you can use spaces and ends of line freely in the macro definitions without getting unwanted spaces when you call the macros. The ignored characters won't generate spaces, but they'll still act as terminators for control sequences. If you really do want a space, you can still get it with the `\space` command (p. '`\space`').

Of course you'll need to restore the category codes of space and end of line to their normal values (10 and 5, respectively) at the end of the file. You can do this either by enclosing the entire file in a group or by restoring the values explicitly. If you choose to enclose the file in a group, then you should also set `\globaldefs` to 1 so that all the macro definitions will be global and thus visible outside of the group.

A miniature example of a macro file of this form is:

```
\catcode `\  = 9  \endlinechar = -1

\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 }

\catcode `\  = 10  \endlinechar = `\^^M
```

Without the changed category codes, these macros would have to be written much more compactly, using fewer spaces and more '`%`'s at the ends of lines.