`category code.` The *category code* of a character determines that character's role in TEX. For instance, TEX assigns a certain role to letters, another to space characters, and so forth. TEX attaches a code to each character that it reads. When TEX reads the letter '`r`', for example, it ordinarily attaches the category code 11 (letter) to it. For simple TEX applications you won't need to worry about category codes, but they become important when you are trying to achieve special effects.

Category codes apply only to characters that TEX reads from input files. Once a character has gotten past TEX's gullet (see "anatomy of TEX", p. '`\anatomy`') and been interpreted, its category code no longer matters. A character that you produce with the `\char` command (p. '`\char`') does not have a category code because `\char` is an instruction to TEX to produce a certain character in a certain font. For instance, the ASCII code for '\' (the usual escape character) is 92. If you type '`\char92 grok`', it is *not* equivalent to `\grok`. Instead it tells TEX to typeset '*c*grok', where *c* is the character in position 92 of the code table for the current font.

You can use the `\catcode` command (p. '`\catcode`') to reassign the category code of any character. By changing category codes you can change the roles of various characters. For instance, if you type '`\catcode`‘`\@ = 11`', the category code of the at sign (`@`) will be set to "letter". You then can use '`@`' in the name of a control sequence.

Here is a list of the category codes defined by TEX, (see p. '`twocarets`' for an explanation of the `^^` notation), together with the characters in each category (as assigned by TEX and plain TEX):

Code	*Meaning*
0	Escape character `\`
1	Beginning of group `{`
2	End of group `}`
3	Math shift `$`
4	Alignment tab `&`
5	End of line `^^M` ≡ ASCII ⟨return⟩
6	Macro parameter `#`
7	Superscript `^` and `^^K`
8	Subscript `_` and `^^A`
9	Ignored character `^^@` ≡ ASCII ⟨null⟩
10	Space ␣ and `^^I` ≡ ASCII ⟨horizontal tab⟩
11	Letter `A` . . . `Z` and `a` . . . `z`
12	Other character (everything not listed above or below)
13	Active character `~` and `^^L` ≡ ASCII ⟨form feed⟩
14	Comment character `%`
15	Invalid character `^^?` ≡ ASCII ⟨delete⟩

Except for categories 11–13, all the characters in a particular category produce the same effect. For instance, suppose that you type:

```
\catcode`\[ = 1 \catcode`\] = 2
```

Then the left and right bracket characters become beginning-of-group and end-of-group characters equivalent to the left and right brace characters. With these definitions '`[a b]`' is a valid group, and so are '`[a b}`' and '`{a b]`'.

The characters in categories 11 (letter) and 12 (other character) act as commands that mean "typeset the character with this code from the current font". The only distinction between letters and "other" characters is that letters can appear in control words but "other" characters can't.

A character in category 13 (active) acts like a control sequence all by itself. TEX complains if it encounters an active character that doesn't have a definition associated with it.

If TEX encounters an invalid character (category 15) in your input, it will complain about it.

The '`^^K`' and '`^^A`' characters have been included in categories 8 (subscript) and 9 (superscript), even though these meanings don't follow the standard ASCII interpretation. That's because some keyboards, notably some at Stanford University where TEX originated, have down arrow and up arrow keys that generate these characters.

There's a subtle point about the way TEX assigns category codes that can trip you up if you're not aware of it. TEX sometimes needs to look at a character twice as it does its initial scan: first to find the end of some preceding construct, e.g., a control sequence, and later to turn that character into a token. TEX doesn't assign the category code until its *second* look at the character. For example:

```
\def\foo{\catcode`\$ = 11 }% Make $ be a letter.
\foo$ % Produces a `$'.
\foo$ % Undefined control sequence `\foo$'.
```

This bit of TEX code produces '$' in the typeset output. When TEX first sees the '`$`' on the second line, it's looking for the end of a control sequence name. Since the '`$`' isn't yet a letter, it marks the end of '`\foo`'. Next, TEX expands the '`\foo`' macro and changes the category code of '`$`' to 11 (letter). Then TEX reads the '`$`' "for real". Since '`$`' is now a letter, TEX produces a box containing the '`$`' character in the current font. When TEX sees the third line, it treats '`$`' as a letter and thus considers it to be part of the control sequence name. As a result it complains about an undefined control sequence `\foo$`.

TEX behaves this way even when the terminating character is an end of line. For example, suppose that the macro `\fum` activates the end-of-line character. Then if `\fum` appears on a line ℓ by itself, TEX will first

category code **3**

interpret the end of line of ℓ as the end of the `\fum` control sequence and then will *reinterpret* the end of line of ℓ as an active character.