find_int calls ferror and feof to see if the problem was a read error or end-of-file. If not, fscanf must have failed because of a matching error, so find_int skips the rest of the characters on the current line and tries again. Note the use of the conversion %*[^\n] to skip all characters up to the next new-line. (Now that we know about scansets, it's time to show off!)

22.4 Character I/O

In this section, we'll examine library functions that read and write single characters. These functions work equally well with text streams and binary streams.

You'll notice that the functions in this section treat characters as values of type int, not char. One reason is that the input functions indicate an end-of-file (or error) condition by returning EOF, which is a negative integer constant.

Output Functions

```
int fputc(int c, FILE *stream);
int putc(int c, FILE *stream);
int putchar(int c);
```

putchar

putchar writes one character to the stdout stream:

```
putchar(ch);      /* writes ch to stdout */
```

fputc
putc

fputc and putc are more general versions of putchar that write a character to an arbitrary stream:

```
fputc(ch, fp);    /* writes ch to fp */
putc(ch, fp);     /* writes ch to fp */
```

Although putc and fputc do the same thing, putc is usually implemented as a macro (as well as a function), while fputc is implemented only as a function. putchar itself is usually a macro defined in the following way:

```
#define putchar(c) putc((c), stdout)
```

It may seem odd that the library provides both putc and fputc. But, as we saw in Section 14.3, macros have several potential problems. The C standard allows the putc macro to evaluate the stream argument more than once, which fputc isn't permitted to do. Although programmers usually prefer putc, which gives a faster program, fputc is available as an alternative. Q&A

If a write error occurs, all three functions set the error indicator for the stream and return EOF; otherwise, they return the character that was written.