Incidentally, you can't just put a single \ character in a string; the compiler will assume that it's the beginning of an escape sequence. To print a single \ character, put two \ characters in the string:

```
printf("\\");    /* prints one \ character */
```

3.2 The scanf Function

Just as `printf` prints output in a specified format, `scanf` reads input according to a particular format. A `scanf` format string, like a `printf` format string, may contain both ordinary characters and conversion specifications. The conversions allowed with `scanf` are essentially the same as those used with `printf`.

In many cases, a `scanf` format string will contain only conversion specifications, as in the following example:

```
int i, j;
float x, y;

scanf("%d%d%f%f", &i, &j, &x, &y);
```

Suppose that the user enters the following input line:

```
1 -20 .3 -4.0e3
```

`scanf` will read the line, converting its characters to the numbers they represent, and then assign 1, –20, 0.3, and –4000.0 to `i`, `j`, `x`, and `y`, respectively. "Tightly packed" format strings like `"%d%d%f%f"` are common in `scanf` calls. `printf` format strings are less likely to have adjacent conversion specifications.

`scanf`, like `printf`, contains several traps for the unwary. When using `scanf`, the programmer must check that the number of conversion specifications matches the number of input variables and that each conversion is appropriate for the corresponding variable—as with `printf`, the compiler isn't required to check for a possible mismatch. Another trap involves the & symbol, which normally precedes each variable in a `scanf` call. The & is usually (but not always) required, and it's the programmer's responsibility to remember to use it.

⚠ Forgetting to put the & symbol in front of a variable in a call of `scanf` will have unpredictable—and possibly disastrous—results. A program crash is a common outcome. At the very least, the value that is read from the input won't be stored in the variable; instead, the variable will retain its old value (which may be meaningless if the variable wasn't given an initial value). Omitting the & is an extremely common error—be careful! Some compilers can spot this error and produce a warning message such as *"format argument is not a pointer."* (The term *pointer* is defined in Chapter 11; the & symbol is used to create a pointer to a variable.) If you get a warning, check for a missing &.
