- All this conversion-character stuff can get complex. Rest assured that seldom does anyone memorize it. Often, advanced programmers have to consult their C language references and run some tests to see which formatting command does what. Most of the time, you aren't bothered with this stuff, so don't panic.

`scanf` Is Pronounced "Scan-Eff"

Output without input is like Desi without Lucy, yang without yin, Caesar salad without the garlic. It means that the seven dwarves would be singing "Oh, Oh, Oh" rather than "I/O, I/O." Besides — and this may be the most horrid aspect of all — without input, the computer just sits there and talks *at* you. That's just awful.

C has numerous tools for making the computer listen to you. A number of commands read input from the keyboard, from commands that scan for individual characters to the vaunted `scanf()` function, which is used to snatch a string of text from the keyboard and save it in the cuddly, warm paws of a string variable.

- `scanf()` is a function like `printf()`. Its purpose is to read text from the keyboard.
- Like the *f* in `printf()`, the *f* in `scanf()` means *formatted.* You can use `scanf()` to read a specifically formatted bit of text from the keyboard. In this chapter, however, you just use `scanf()` to read a line of text, nothing fancy.

Putting `scanf` together

To make `scanf()` work, you need two things. First, you need a storage place to hold the text you enter. Second, you need the `scanf` function itself.

The storage place is called a *string variable. String* means a string of characters — text. *Variable* means that the string isn't set — it can be whatever the user types. A string variable is a storage place for text in your programs. (Variables are discussed at length in Chapter 8.)

The second thing you need is `scanf()` itself. Its format is somewhat similar to the advanced, cryptic format for `printf()`, so there's no point in wasting any of your brain cells covering that here.