<div style="background:yellow">
Calling scanf is a powerful but unforgiving way to read data. Many professional C programmers avoid scanf, instead reading all data in character form and converting it to numeric form later.
</div>
We'll use scanf quite a bit, especially in the early chapters of this book, because it provides a simple way to read numbers. Be aware, however, that many of our programs won't behave properly if the user enters unexpected input. As we'll see later, it's possible to have a program test whether scanf successfully read the requested data (and attempt to recover if it didn't). Such tests are impractical for the programs in this book—they would add too many statements and obscure the point of the examples.

How scanf Works

scanf can actually do much more than I've indicated so far. It is essentially a "pattern-matching" function that tries to match up groups of input characters with conversion specifications.

Like the printf function, scanf is controlled by the format string. When it is called, scanf begins processing the information in the string, starting at the left. For each conversion specification in the format string, scanf tries to locate an item of the appropriate type in the input data, skipping blank space if necessary. scanf then reads the item, stopping when it encounters a character that can't possibly belong to the item. If the item was read successfully, scanf continues processing the rest of the format string. If any item is not read successfully, scanf returns immediately without looking at the rest of the format string (or the remaining input data).

As it searches for the beginning of a number, scanf ignores *white-space characters* (the space, horizontal and vertical tab, form-feed, and new-line characters). As a result, numbers can be put on a single line or spread out over several lines. Consider the following call of scanf:

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

Suppose that the user enters three lines of input:

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

scanf sees one continuous stream of characters:

```
••1□-20•••.3□•••-4.0e3□
```

(I'm using • to represent the space character and □ to represent the new-line character.) Since it skips over white-space characters as it looks for the beginning of each number, scanf will be able to read the numbers successfully. In the following diagram, an s under a character indicates that it was skipped, and an r indicates it was read as part of an input item:

```
••1□-20•••.3□•••-4.0e3□
ssrsrrrsssrrssssrrrrrr
```