sscanf The sscanf function is similar to scanf and fscanf, except that it reads from a string (pointed to by its first argument) instead of reading from a stream. sscanf's second argument is a format string identical to that used by scanf and fscanf.

sscanf is handy for extracting data from a string that was read by another input function. For example, we might use fgets to obtain a line of input, then pass the line to sscanf for further processing:

```
fgets(str, sizeof(str), stdin);  /* reads a line of input */
sscanf(str, "%d%d", &i, &j);     /* extracts two integers */
```

One advantage of using sscanf instead of scanf or fscanf is that we can examine an input line as many times as needed, not just once, making it easier to recognize alternate input forms and to recover from errors. Consider the problem of reading a date that's written either in the form *month/day/year* or *month-day-year*. Assuming that str contains a line of input, we can extract the month, day, and year as follows:

```
if (sscanf(str, "%d /%d /%d", &month, &day, &year) == 3)
  printf("Month: %d, day: %d, year: %d\n", month, day, year);
else if (sscanf(str, "%d -%d -%d", &month, &day, &year) == 3)
  printf("Month: %d, day: %d, year: %d\n", month, day, year);
else
  printf("Date not in the proper form\n");
```

Like the scanf and fscanf functions, sscanf returns the number of data items successfully read and stored. sscanf returns EOF if it reaches the end of the string (marked by a null character) before finding the first item.

Q & A

Q: If I use input or output redirection, will the redirected file names show up as command-line arguments? [p. 541]

A: No: the operating system removes them from the command line. Let's say that we run a program by entering

```
demo foo <in_file bar >out_file baz
```

The value of argc will be 4, argv[0] will point to the program name, argv[1] will point to "foo", argv[2] will point to "bar", and argv[3] will point to "baz".

Q: I thought that the end of a line was always marked by a new-line character. Now you're saying that the end-of-line marker varies, depending on the operating system. How you explain this discrepancy? [p. 542]

A: C library functions make it *appear* as though each line ends with a single new-line