A string, like a number, can be printed within a field. The `%ms` conversion will display a string in a field of size *m*. (A string with more than *m* characters will be printed in full, not truncated.) If the string has fewer than *m* characters, it will be right-justified within the field. To force left justification instead, we can put a minus sign in front of *m*. The *m* and *p* values can be used in combination: a conversion specification of the form `%m.ps` causes the first *p* characters of a string to be displayed in a field of size *m*.

`printf` isn't the only function that can write strings. The C library also provides `puts`, which is used in the following way:

```
puts(str);
```

`puts` has only one argument (the string to be printed). After writing the string, `puts` always writes an additional new-line character, thus advancing to the beginning of the next output line.

Reading Strings Using `scanf` and `gets`

The `%s` conversion specification allows `scanf` to read a string into a character array:

```
scanf("%s", str);
```

There's no need to put the `&` operator in front of `str` in the call of `scanf`; like any array name, `str` is treated as a pointer when passed to a function.

white-space characters ➤3.2

When `scanf` is called, it skips white space, then reads characters and stores them in `str` until it encounters a white-space character. `scanf` always stores a null character at the end of the string.

A string read using `scanf` will never contain white space. Consequently, `scanf` won't usually read a full line of input; a new-line character will cause `scanf` to stop reading, but so will a space or tab character. To read an entire line of input at a time, we can use `gets`. Like `scanf`, the `gets` function reads input characters into an array, then stores a null character. In other respects, however, `gets` is somewhat different from `scanf`:

- `gets` doesn't skip white space before starting to read the string (`scanf` does).
- `gets` reads until it finds a new-line character (`scanf` stops at any white-space character). Incidentally, `gets` discards the new-line character instead of storing it in the array; the null character takes its place.

To see the difference between `scanf` and `gets`, consider the following program fragment:

```
char sentence[SENT_LEN+1];

printf("Enter a sentence:\n");
scanf("%s", sentence);
```