```
ch = toupper(ch);    /* converts ch to upper case */
```

When it's called, `toupper` checks whether its argument (`ch` in this case) is a lower-case letter. If so, it returns the corresponding upper-case letter. Otherwise, `toupper` returns the value of the argument. In our example, we've used the assignment operator to store the return value of `toupper` back into the `ch` variable, although we could just as easily have done something else with it—stored it in another variable, say, or tested it in an `if` statement:

```
if (toupper(ch) == 'A') ...
```

Programs that call `toupper` need to have the following `#include` directive at the top:

```
#include <ctype.h>
```

`toupper` isn't the only useful character-handling function in the C library. Section 23.5 describes them all and gives examples of their use.

Reading and Writing Characters using `scanf` and `printf`

The `%c` conversion specification allows `scanf` and `printf` to read and write single characters:

```
char ch;

scanf("%c", &ch);    /* reads a single character */
printf("%c", ch);    /* writes a single character */
```

`scanf` doesn't skip white-space characters before reading a character. If the next unread character is a space, then the variable `ch` in the previous example will contain a space after `scanf` returns. To force `scanf` to skip white space before reading a character, put a space in its format string just before `%c`:

```
scanf(" %c", &ch);    /* skips white space, then reads ch */
```

Recall from Section 3.2 that a blank in a `scanf` format string means "skip zero or more white-space characters."

Since `scanf` doesn't normally skip white space, it's easy to detect the end of an input line: check to see if the character just read is the new-line character. For example, the following loop will read and ignore all remaining characters in the current input line:

```
do {
  scanf("%c", &ch);
} while (ch != '\n');
```

When `scanf` is called the next time, it will read the first character on the next input line.