When an `if` statement is executed, the expression in the parentheses is evaluated; if the value of the expression is nonzero—which C interprets as true—the statement after the parentheses is executed. Here's an example:

```
if (line_num == MAX_LINES)
  line_num = 0;
```

The statement `line_num = 0;` is executed if the condition `line_num == MAX_LINES` is true (has a nonzero value).

Don't confuse == (equality) with = (assignment). The statement

```
if (i == 0) ...
```

tests whether `i` is equal to 0. However, the statement

```
if (i = 0) ...
```

assigns 0 to `i`, then tests whether the *result* is nonzero. In this case, the test always fails.

Confusing == with = is perhaps the most common C programming error, probably because = means "is equal to" in mathematics (and in certain programming languages). Some compilers issue a warning if they notice = where == would normally appear.

Q&A

Often the expression in an `if` statement will test whether a variable falls within a range of values. To test whether $0 \le i < n$, for example, we'd write

idiom

```
if (0 <= i && i < n) ...
```

To test the *opposite* condition (`i` is outside the range), we'd write

idiom

```
if (i < 0 || i >= n) ...
```

Note the use of the || operator instead of the && operator.

Compound Statements

In our `if` statement template, notice that *statement* is singular, not plural:

`if` (*expression*) *statement*

What if we want an `if` statement to control *two* or more statements? That's where the ***compound statement*** comes in. A compound statement has the form

compound statement

{ *statements* }

By putting braces around a group of statements, we can force the compiler to treat it as a single statement.