Some programmers always use braces, even when they're not strictly necessary:

```
while (i < n) {    /* braces allowed, but not required */
  i = i * 2;
}
```

As a second example, let's trace the execution of the following statements, which display a series of "countdown" messages:

```
i = 10;
while (i > 0) {
  printf("T minus %d and counting\n", i);
  i--;
}
```

Before the `while` statement is executed, the variable `i` is assigned the value 10. Since 10 is greater than 0, the loop body is executed, causing the message `T minus 10 and counting` to be printed and `i` to be decremented. The condition `i > 0` is then tested again. Since 9 is greater than 0, the loop body is executed once more. This process continues until the message `T minus 1 and counting` is printed and `i` becomes 0. The test `i > 0` then fails, causing the loop to terminate.

The countdown example leads us to make several observations about the `while` statement:

- The controlling expression is false when a `while` loop terminates. Thus, when a loop controlled by the expression `i > 0` terminates, `i` must be less than or equal to 0. (Otherwise, we'd still be executing the loop!)
- The body of a `while` loop may not be executed at all. Since the controlling expression is tested *before* the loop body is executed, it's possible that the body isn't executed even once. If `i` has a negative or zero value when the countdown loop is first entered, the loop will do nothing.
- A `while` statement can often be written in a variety of ways. For example, we could make the countdown loop more concise by decrementing `i` inside the call of `printf`:

Q&A

```
while (i > 0)
  printf("T minus %d and counting\n", i--);
```

Infinite Loops

A `while` statement won't terminate if the controlling expression always has a nonzero value. In fact, C programmers sometimes deliberately create an ***infinite loop*** by using a nonzero constant as the controlling expression:

idiom

```
while (1) ...
```

A `while` statement of this form will execute forever unless its body contains a statement that transfers control out of the loop (`break`, `goto`, `return`) or calls a function that causes the program to terminate.