Another possibility is that the loop terminates, but the statement that should be the loop body is executed only once, after the loop has terminated:

```
i = 11;
while (--i > 0);                          /*** WRONG ***/
  printf("T minus %d and counting\n", i);
```

This example prints the message

```
T minus 0 and counting
```

- In a `for` statement, putting a semicolon after the parentheses causes the statement that should be the loop body to be executed only once:

```
for (i = 10; i > 0; i--);                 /*** WRONG ***/
  printf("T minus %d and counting\n", i);
```

This example also prints the message

```
T minus 0 and counting
```

Q & A

Q: **The following loop appears in Section 6.1:**

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

Why not shorten the loop even more by removing the "`> 0`" test?

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

This version will stop when `i` reaches 0, so it should be just as good as the original. [p. 101]

A: The new version is certainly more concise, and many C programmers would write the loop in just this way. It does have drawbacks, though.

First, the new loop is not as easy to read as the original. It's clear that the loop will terminate when `i` reaches 0, but it's not obvious whether we're counting up or down. In the original loop, that information can be deduced from the controlling expression, `i > 0`.

Second, the new loop behaves differently than the original if `i` should happen to have a negative value when the loop begins to execute. The original loop terminates immediately, but the new loop doesn't.

Q: **Section 6.3 says that, except in rare cases, `for` loops can be converted to `while` loops using a standard pattern. Can you give an example of such a case? [p. 106]**