counting is printed, then i is decremented. The condition i > 0 is then tested again. The loop body will be executed 10 times in all, with i varying from 10 down to 1.

Q&A The for statement is closely related to the while statement. In fact, except in a few rare cases, a for loop can always be replaced by an equivalent while loop:

```
expr1;
while ( expr2 ) {
  statement
  expr3;
}
```

As this pattern shows, *expr1* is an initialization step that's performed only once, before the loop begins to execute, *expr2* controls loop termination (the loop continues executing as long as the value of *expr2* is nonzero), and *expr3* is an operation to be performed at the end of each loop iteration. Applying this pattern to our previous for loop example, we arrive at the following:

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

Studying the equivalent while statement can help us understand the fine points of a for statement. For example, suppose that we replace i-- by --i in our for loop example:

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

How does this change affect the loop? Looking at the equivalent while loop, we see that it has no effect:

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

Since the first and third expressions in a for statement are executed as statements, their values are irrelevant—they're useful only for their side effects. Consequently, these two expressions are usually assignments or increment/decrement expressions.

for Statement Idioms

The for statement is usually the best choice for loops that "count up" (increment a variable) or "count down" (decrement a variable). A for statement that counts up or down a total of n times will usually have one of the following forms: