Everything in the backward-counting `for` loop works per specifications. The loop begins by setting the value of the `count` variable equal to 10. And, it loops as long as the value of the `count` variable is greater than 0 (`count>0`). But, each time the loop repeats, the value of the `count` variable is decremented. It works backward.

Again, you have no reason to loop backward — except that the `printf()` statement belonging to the loop displays the numbers 10 through 1 in a countdown manner. Normally (which means about 99 percent of the time), you only loop forward. It's not only easier to do in your head, but it's also less likely to be a source for programming boo-boos than when you try to loop backward.

- Most loops count forward. The backward-counting loop in C is possible but rarely used, mostly because counting is counting and it's just easier (for humans) to do it forward.
- You can't do a backward loop without decrementing the loop's variable.
- Other than decrementing the loop's variable, the only difference is in the loop's while-true condition (the one in the middle). That requires a little more mental overhead to figure out than with normal `for` loops. It's another reason that this type of loop is rare.
- Okay. The question arises: "Why bother?" Because you have to know about decrementing and the cryptic `--` operator, covered in the next section.

Cryptic C operator symbols, Volume II: The `dec` operator (`--`)

Just as C has a shortcut for incrementing a variable's value, there is also a shortcut for decrementing a variable's value. (If you read the first half of this chapter, you saw this coming from a mile back, most likely.) As a quick review, you can add 1 to a variable's value — *increment* it — by using the following C language statement:

```
i=i+1;
```

Or, if you're cool and remember the shortcut, you can use the cryptic ++ operator to do the same thing:

```
i++;
```

Both examples add 1 to the value of variable `i`.

Consider this example:

```
d=d-1
```