To count backward, you subtract 1 from a variable's value, which is exactly the way you do it in your head: 10, 9, 8, 7, and so on. It looks identical to the incrementing statement, except for the minus sign:

```
b=b-1;
```

The value of variable `b` is 1 less than it was before. If `b` came in with a value of `5`, this statement sets `b`'s value to `4`. This process is known as *decrementing* a variable's value.

- Decrementing, or subtracting 1 (or any number) from a variable's value is just common subtraction. The only big deal here is that decrementing is done in a loop, which makes the loop count backward.
- Incrementing means adding (1) to a variable's value.
- Decrementing means subtracting (1) from a variable's value.
- Decrementing works because C first figures out what's on the right side of the equal sign:

  ```
  b=b-1;
  ```

  First comes `b-1`, so the computer subtracts 1 from the value of variable `b`. Then, that value is slid through the equal signs, back into the variable `b`. The variable is decremented.

How counting backward fits into the `for` loop

Take another look at Line 7 from the OLLYOLLY.C program:

```
for(count=10;count>0;count=count-1)
```

It's basic `for` loop stuff. The loop has a starting place, a while-true condition, and a do-this thing. The parts are listed in Table 16-1.

Table 16-1 How the `for` Loop Counts Backward

Loop Part	*Condition*
Starting	count=10
While-true	count>0
Do-this	count=count-1