The compiler takes the value of the variable i and adds 1 to it each time the for loop is run through once. (Refer to Chapter 11 for more info on the art of incrementation.)

Altogether, the for loop works out to repeat itself — and any statements that belong to it — a total of five times. Table 15-1 shows how it works.

Table 15-1 How the Variable i Works Its Way through the `for` Loop

Value of i	*Is i<5 true?*	*Statement*	*Do This*
i=0	Yes, keep looping←	`printf()...`(1)	i=0+1
i=1	Yes, keep looping←	`printf()...`(2)	i=1+1
i=2	Yes, keep looping←	`printf()...`(3)	i=2+1
i=3	Yes, keep looping←	`printf()...`(4)	i=3+1
i=4	Yes, keep looping←	`printf()...`(5)	i=4+1
i=5	No — stop now!		

In Table 15-1, the value of the variable i starts out equal to 0, as set up in the for statement. Then, the second item — the comparison — is tested. Is i<5 true? If so, the loop marches on.

As the loop works, the third part of the for statement is calculated and the value of i is incremented. Along with that, any statements belonging to the for command are executed. When those statements are done, the comparison i<5 is made again and the loop either repeats or stops based on the results of that comparison.

- The for loop can be cumbersome to understand because it has so many parts. However, it's the best way in your C programs to repeat a group of statements a given number of times.
- The third item in the for statement's parentheses — do_this — is executed only once for each loop. It's true whether for has zero, one, or several statements belonging to it.
- TIP Where most people screw up with the for loop is the second item. They remember that the first item means "start here," but they think that the second item is "end here." It's not! The second item means "keep looping while this condition is true." It works like an if comparison. The compiler doesn't pick up and flag it as a boo-boo, but your program doesn't run properly when you make this mistake.