Overview

You have probably noticed that much of the work a computer does is repeated many times. For example, a computer can print a personalized letter to each person in a database. The basic operation of printing the letter repeats for each person in the database. When a program repeats a group of statements a given number of times, the repetition is accomplished using a ***loop***.

In Chapter 7 you learned about sequence structures and selection structures. The final category of structures is ***iteration structures***. Loops are iteration structures. Each "loop" or pass through a group of statements is called an ***iteration***. A condition specified in the program controls the number of iterations performed. For example, a loop may iterate until a specific variable reaches the value 100.

In this chapter you will learn about the three iteration structures available in C++: the *for loop*, the *while loop*, and the *do while loop*.

CHAPTER 8, SECTION 1

The for Loop

The ***for loop*** repeats one or more statements a specified number of times. A for loop is difficult to read the first time you see one. Like an if statement, the for loop uses parentheses. In the parentheses are three items called ***parameters,*** which are needed to make a for loop work. Each parameter in a for loop is an expression. Figure 8-1 shows the format of a for loop.

```
for (initializing expression; control expression; step expression)
   {statement or statement block}
```

FIGURE 8-1
A for loop repeats one or more statements a specified number of times.

Look at the program in Figure 8-2. The variable `i` is used as a counter. The counter variable is used in all three of the for loop's expressions. The first parameter, called the ***initializing expression,*** initializes the counter variable. The second parameter is the expression that will end the loop, called the ***control expression.*** As long as the control expression is true, the loop continues to iterate. The third parameter is the ***step expression.*** It changes the counter variable, usually by adding to it.

In Figure 8-2, the statements in the for loop will repeat three times. The variable `i` is declared as an integer. In the for statement, `i` is initialized to 1. The control expression tests to see if the value of `i` is still less than or equal to 3. When `i` exceeds 3, the loop will end. The step expression increments `i` by one each time the loop iterates.

PITFALLS

Placing a semicolon after the closing parenthesis of a for loop will prevent any lines from being iterated.


```
#include <iostream.h>

main()
{
 int i;
 for(i = 1; i <= 3; i++)
   cout << i << '\n';
 return 0;
}
```

FIGURE 8-2
A for loop uses a counter variable to test the control expression.

EXERCISE 8-1 USING A for LOOP

1. Key the program from Figure 8-2 into a blank editor screen.
2. Save the source code file as *FORLOOP.CPP*.
3. Compile and run the program.
4. Close the source file.

COUNTING BACKWARD AND OTHER TRICKS

A counter variable can also count backward by having the step expression decrement the value rather than increment it.

EXERCISE 8-2 USING A DECREMENTING COUNTER VARIABLE

1. Key the following program into a blank editor screen:

```
#include <iostream.h>

main()
{
  int i;
  for(i = 10; i >= 0; i--)
    cout << i << '\n';
  cout << "End of loop.\n";
  return 0;
}
```

2. Save the source file as *BACKWARD.CPP*.
3. Compile and run the program. Figure 8-3 shows the output you should see.
4. Close the source code file.

The output prints numbers from 10 to 0 because `i` is being decremented in the step expression. The phrase "End of loop." is printed only once because the loop ends with the semicolon that follows the first cout statement.