B)
```
int v = 10;
do
   cout << v << endl;
while (v < 5);
```

C)
```
int count = 0, number = 0, limit = 4;
do
{
   number += 2;
   count++;
} while (count < limit);
cout << number << " " << count << endl;
```

5.6 The for Loop

CONCEPT: The for loop is ideal for performing a known number of iterations.

In general, there are two categories of loops: conditional loops and count-controlled loops. A *conditional loop* executes as long as a particular condition exists. For example, an input validation loop executes as long as the input value is invalid. When you write a conditional loop, you have no way of knowing the number of times it will iterate.


VideoNote
The for Loop

Sometimes you know the exact number of iterations that a loop must perform. A loop that repeats a specific number of times is known as a *count-controlled* loop. For example, if a loop asks the user to enter the sales amounts for each month in the year, it will iterate twelve times. In essence, the loop counts to twelve and asks the user to enter a sales amount each time it makes a count. A count-controlled loop must possess three elements:

1. It must initialize a counter variable to a starting value.
2. It must test the counter variable by comparing it to a maximum value. When the counter variable reaches its maximum value, the loop terminates.
3. It must update the counter variable during each iteration. This is usually done by incrementing the variable.

Count-controlled loops are so common that C++ provides a type of loop specifically for them. It is known as the `for` loop. The `for` loop is specifically designed to initialize, test, and update a counter variable. Here is the format of the `for` loop when it is used to repeat a single statement:

```
for (initialization; test; update)
  statement;
```

The format of the `for` loop when it is used to repeat a block is

```
for (initialization; test; update)
{
  statement;
  statement;
  // Place as many statements here
  // as necessary.
}
```

The first line of the `for` loop is the *loop header*. After the key word `for`, there are three expressions inside the parentheses, separated by semicolons. (Notice there is not a semicolon after the third expression.) The first expression is the *initialization expression*. It is normally used to initialize a counter variable to its starting value. This is the first action performed by the loop, and it is only done once. The second expression is the *test expression*. This is an expression that controls the execution of the loop. As long as this expression is true, the body of the `for` loop will repeat. The `for` loop is a pretest loop, so it evaluates the test expression before each iteration. The third expression is the *update expression*. It executes at the end of each iteration. Typically, this is a statement that increments the loop's counter variable.

Here is an example of a simple `for` loop that prints "Hello" five times:

```
for (count = 0; count < 5; count++)
   cout << "Hello" << endl;
```

In this loop, the initialization expression is `count = 0`, the test expression is `count < 5`, and the update expression is `count++`. The body of the loop has one statement, which is the `cout` statement. Figure 5-6 illustrates the sequence of events that takes place during the loop's execution. Notice that Steps 2 through 4 are repeated as long as the test expression is `true`.

Figure 5-6

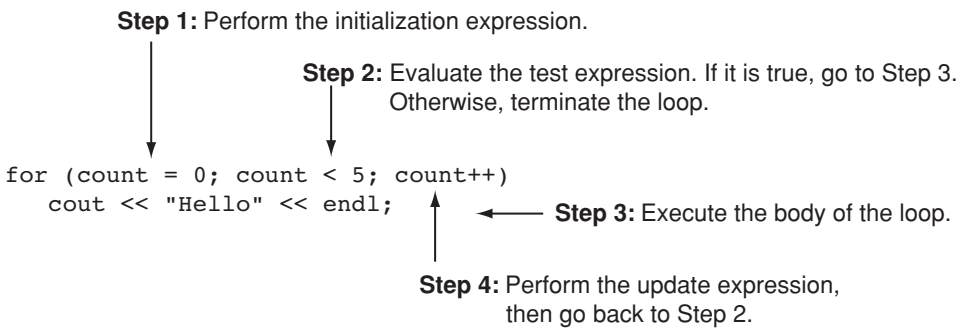
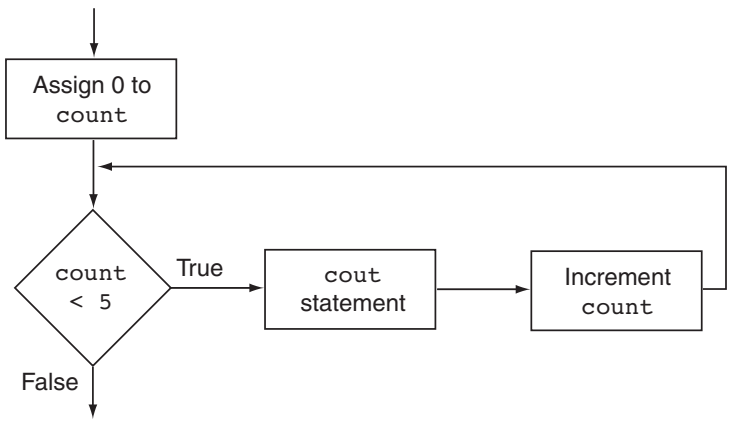


Figure 5-7 shows the loop's logic in the form of a flowchart.

Figure 5-7



Notice how the counter variable, count, is used to control the number of times that the loop iterates. During the execution of the loop, this variable takes on the values 1 through 5, and when the test expression count < 5 is false, the loop terminates. Also notice that in this example the count variable is used only in the loop header, to control the number of loop iterations. It is not used for any other purpose. It is also possible to use the counter variable within the body of the loop. For example, look at the following code:

```
for (number = 1; number <= 10; number++)
   cout << number << " " << endl;
```

The counter variable in this loop is number. In addition to controlling the number of iterations, it is also used in the body of the loop. This loop will produce the following output:

```
1 2 3 4 5 6 7 8 9 10
```

As you can see, the loop displays the contents of the number variable during each iteration. Program 5-9 shows another example of a for loop that uses its counter variable within the body of the loop. This is yet another program that displays a table showing the numbers 1 through 10 and their squares.

Program 5-9

```
1  // This program displays the numbers 1 through 10 and
2  // their squares.
3  #include <iostream>
4  using namespace std;
5
6  int main()
7  {
8       const int MIN_NUMBER = 1,    // Starting value
9                 MAX_NUMBER = 10;   // Ending value
10      int num;
11
12      cout << "Number Number Squared\n";
13      cout << "-------------------------\n";
14
15      for (num = MIN_NUMBER; num <= MAX_NUMBER; num++)
16           cout << num << "\t\t" << (num * num) << endl;
17
18      return 0;
19 }
```

Program Output

```
Number Number Squared
---------------------
1           1
2           4
3           9
4           16
5           25
6           36
7           49
8           64
9           81
10          100
```

Figure 5-8 illustrates the sequence of events performed by this for loop, and Figure 5-9 shows the logic of the loop as a flowchart.

Figure 5-8

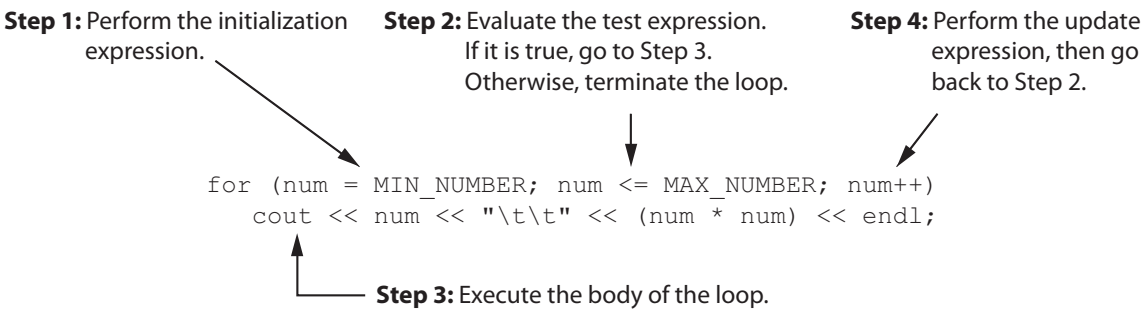
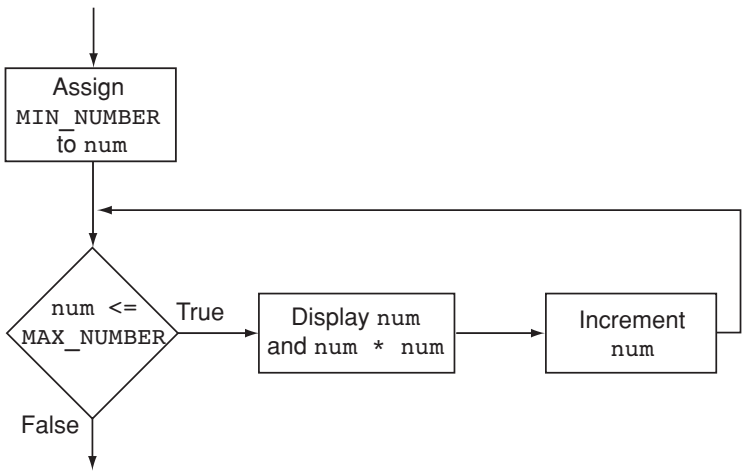


Figure 5-9

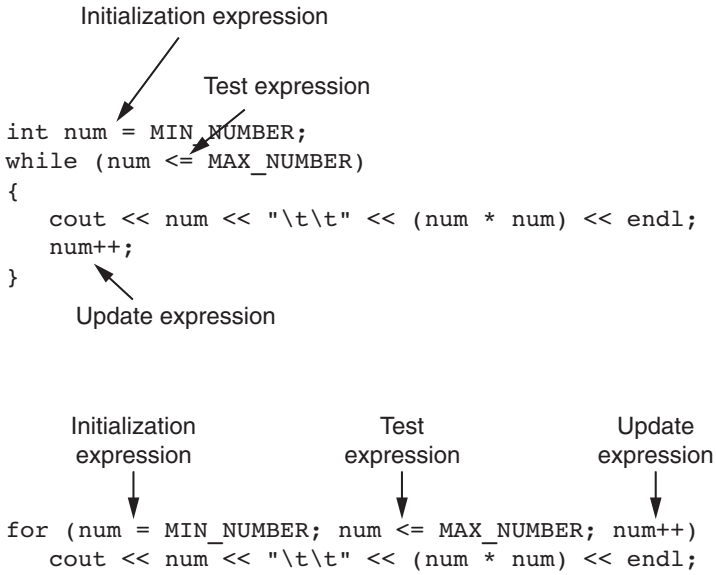


Using the `for` Loop Instead of `while` or `do-while`

You should use the `for` loop instead of the `while` or `do-while` loop in any situation that clearly requires an initialization, uses a false condition to stop the loop, and requires an update to occur at the end of each loop iteration. Program 5-9 is a perfect example. It requires that the `num` variable be initialized to 1, it stops the loop when `num` is greater than 10, and it increments `num` at the end of each loop iteration.

Recall that when we first introduced the idea of a counter variable we examined Program 5-6, which uses a `while` loop to display the table of numbers and their squares. Because the loop in that program requires an initialization, uses a false test expression to stop, and performs an increment at the end of each iteration, it can easily be converted to a `for` loop. Figure 5-10 shows how the `while` loop in Program 5-6 and the `for` loop in Program 5-9 each have initialization, test, and update expressions.

Figure 5-10



The for Loop Is a Pretest Loop

Because the `for` loop tests its test expression before it performs an iteration, it is a pretest loop. It is possible to write a `for` loop in such a way that it will never iterate. Here is an example:

```
for (count = 11; count <= 10; count++)
   cout << "Hello" << endl;
```

Because the variable `count` is initialized to a value that makes the test expression false from the beginning, this loop terminates as soon as it begins.

Avoid Modifying the Counter Variable in the Body of the for Loop

Be careful not to place a statement that modifies the counter variable in the body of the `for` loop. All modifications of the counter variable should take place in the update expression, which is automatically executed at the end of each iteration. If a statement in the body of the loop also modifies the counter variable, the loop will probably not terminate when you expect it to. The following loop, for example, increments `x` twice for each iteration:

```
for (x = 1; x <= 10; x++)
{
   cout << x << endl;
   x++;  // Wrong!
}
```

Other Forms of the Update Expression

You are not limited to using increment statements in the update expression. Here is a loop that displays all the even numbers from 2 through 100 by adding 2 to its counter:

```
for (num = 2; num <= 100; num += 2)
   cout << num << endl;
```

And here is a loop that counts backward from 10 down to 0:

```
for (num = 10; num >= 0; num--)
   cout << num << endl;
```

Defining a Variable in the `for` Loop's Initialization Expression

Not only may the counter variable be initialized in the initialization expression, it may be defined there as well. The following code shows an example. This is a modified version of the loop in Program 5-9.

```
for (int num = MIN_NUMBER; num <= MAX_NUMBER; num++)
   cout << num << "\t\t" << (num * num) << endl;
```

In this loop, the `num` variable is both defined and initialized in the initialization expression. If the counter variable is used only in the loop, it makes sense to define it in the loop header. This makes the variable's purpose more clear.

When a variable is defined in the initialization expression of a `for` loop, the scope of the variable is limited to the loop. This means you cannot access the variable in statements outside the loop. For example, the following program segment will not compile because the last `cout` statement cannot access the variable `count`.

```
for (int count = 1; count <= 10; count++)
   cout << count << endl;
cout << "count is now " << count << endl;    // ERROR!
```

Creating a User Controlled `for` Loop

Sometimes you want the user to determine the maximum value of the counter variable in a `for` loop, and therefore determine the number of times the loop iterates. For example, look at Program 5-10. This is another program that displays a list of numbers and their squares. Instead of displaying the numbers 1 through 10, this program allows the user to enter the minimum and maximum values to display.

Program 5-10

```
1  // This program demonstrates a user controlled for loop.
2  #include <iostream>
3  using namespace std;
4
5  int main()
6  {
7       int minNumber,   // Starting number to square
8           maxNumber;   // Maximum number to square
9
```

```
10      // Get the minimum and maximum values to display.
11      cout << "I will display a table of numbers and "
12           << "their squares.\n"
13           << "Enter the starting number: ";
14      cin >> minNumber;
15      cout << "Enter the ending number: ";
16      cin >> maxNumber;
17
18      // Display the table.
19      cout << "Number Number Squared\n"
20           << "-------------------------\n";
21
22      for (int num = minNumber; num <= maxNumber; num++)
23          cout << num << "\t\t" << (num * num) << endl;
24
25      return 0;
26  }
```

Program Output with Example Input Shown in Bold

```
I will display a table of numbers and their squares.
Enter the starting number: 6 [Enter]
Enter the ending number: 12 [Enter]
Number Number Squared
---------------------
6           36
7           49
8           64
9           81
10          100
11          121
12          144
```

Before the loop, the code in lines 11 through 16 asks the user to enter the starting and ending numbers. These values are stored in the `minNumber` and `maxNumber` variables. These values are used in the `for` loop's initialization and test expressions:

```
for (int num = minNumber; num <= maxNumber; num++)
```

In this loop, the `num` variable takes on the values from `maxNumber` through `maxValue`, and then the loop terminates.

Using Multiple Statements in the Initialization and Update Expressions

It is possible to execute more than one statement in the initialization expression and the update expression. When using multiple statements in either of these expressions, simply separate the statements with commas. For example, look at the loop in the following code, which has two statements in the initialization expression.

```
int x, y;
for (x = 1, y = 1; x <= 5; x++)
{
   cout << x << " plus " << y
        << " equals " << (x + y)
        << endl;
}
```

This loop's initialization expression is

```
x = 1, y = 1
```

This initializes two variables, `x` and `y`. The output produced by this loop is

```
1 plus 1 equals 2
2 plus 1 equals 3
3 plus 1 equals 4
4 plus 1 equals 5
5 plus 1 equals 6
```

We can further modify the loop to execute two statements in the update expression. Here is an example:

```
int x, y;
for (x = 1, y = 1; x <= 5; x++, y++)
{
   cout << x << " plus " << y
        << " equals " << (x + y)
        << endl;
}
```

The loop's update expression is

```
x++, y++
```

This update expression increments both the `x` and `y` variables. The output produced by this loop is

```
1 plus 1 equals 2
2 plus 2 equals 4
3 plus 3 equals 6
4 plus 4 equals 8
5 plus 5 equals 10
```

Connecting multiple statements with commas works well in the initialization and update expressions, but do *not* try to connect multiple expressions this way in the test expression. If you wish to combine multiple expressions in the test expression, you must use the `&&` or `||` operators.

Omitting the `for` Loop's Expressions

The initialization expression may be omitted from inside the `for` loop's parentheses if it has already been performed or no initialization is needed. Here is an example of the loop in Program 5-10 with the initialization being performed prior to the loop:

```
int num = 1;
for ( ; num <= maxValue; num++)
   cout << num << "\t\t" << (num * num) << endl;
```

You may also omit the update expression if it is being performed elsewhere in the loop or if none is needed. Although this type of code is not recommended, the following `for` loop works just like a `while` loop:

```
int num = 1;
for ( ; num <= maxValue; )
{
   cout << num << "\t\t" << (num * num) << endl;
   num++;
}
```

You can even go so far as to omit all three expressions from the `for` loop's parentheses. Be warned, however, that if you leave out the test expression, the loop has no built-in way of terminating. Here is an example:

```
for ( ; ; )
  cout << "Hello World\n";
```

Because this loop has no way of stopping, it will display `"Hello World\n"` forever (or until something interrupts the program).

In the Spotlight:

Designing a Count-Controlled Loop with the `for` Statement

Your friend Amanda just inherited a European sports car from her uncle. Amanda lives in the United States, and she is afraid she will get a speeding ticket because the car's speedometer indicates kilometers per hour. She has asked you to write a program that displays a table of speeds in kilometers per hour with their values converted to miles per hour. The formula for converting kilometers per hour to miles per hour is:

*MPH = KPH * 0.6214*

In the formula, *MPH* is the speed in miles per hour and *KPH* is the speed in kilometers per hour.

The table that your program displays should show speeds from 60 kilometers per hour through 130 kilometers per hour, in increments of 10, along with their values converted to miles per hour. The table should look something like this:

KPH	MPH
60	37.3
70	43.5
80	49.7
etc. . . .	
130	80.8

After thinking about this table of values, you decide that you will write a `for` loop that uses a counter variable to hold the kilometer-per-hour speeds. The counter's starting value will be 60, its ending value will be 130, and you will add 10 to the counter variable after each iteration. Inside the loop you will use the counter variable to calculate a speed in miles-per-hour. Program 5-11 shows the code.