

statement won't execute. If the mode of the increment operator is changed, however, the `if` statement will compare 11 to 10, and the `cout` statement will execute:

```
x = 10;
if (++x > 10)
    cout << "x is greater than 10.\n";
```



## Checkpoint

5.1 What will the following program segments display?

- A) 

```
x = 2;
y = x++;
cout << x << y;
```
- B) 

```
x = 2;
y = ++x;
cout << x << y;
```
- C) 

```
x = 2;
y = 4;
cout << x++ << --y;
```
- D) 

```
x = 2;
y = 2 * x++;
cout << x << y;
```
- E) 

```
x = 99;
if (x++ < 100)
    cout << "It is true!\n";
else
    cout << "It is false!\n";
```
- F) 

```
x = 0;
if (++x)
    cout << "It is true!\n";
else
    cout << "It is false!\n";
```

## 5.2

## Introduction to Loops: The `while` Loop

**CONCEPT:** A loop is part of a program that repeats.

Chapter 4 introduced the concept of control structures, which direct the flow of a program. A *loop* is a control structure that causes a statement or group of statements to repeat. C++ has three looping control structures: the `while` loop, the `do-while` loop, and the `for` loop. The difference between these structures is how they control the repetition.

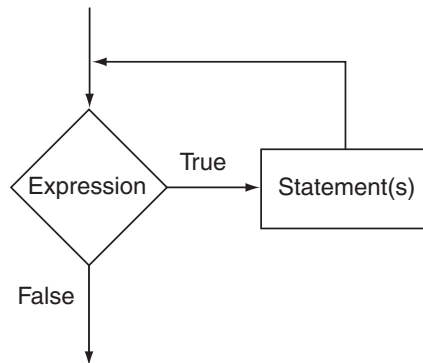


VideoNote  
The `while`  
Loop

## The `while` Loop

The `while` loop has two important parts: (1) an expression that is tested for a true or false value, and (2) a statement or block that is repeated as long as the expression is true. Figure 5-1 shows the logic of a `while` loop.

Figure 5-1



Here is the general format of the while loop:

```
while (expression)
    statement;
```

In the general format, *expression* is any expression that can be evaluated as true or false, and *statement* is any valid C++ statement. The first line shown in the format is sometimes called the *loop header*. It consists of the key word `while` followed by an *expression* enclosed in parentheses.

Here's how the loop works: the *expression* is tested, and if it is true, the *statement* is executed. Then, the *expression* is tested again. If it is true, the *statement* is executed. This cycle repeats until the *expression* is false.

The statement that is repeated is known as the *body* of the loop. It is also considered a conditionally executed statement, because it is executed only under the condition that the *expression* is true.

Notice there is no semicolon after the expression in parentheses. Like the `if` statement, the `while` loop is not complete without the statement that follows it.

If you wish the `while` loop to repeat a block of statements, its format is:

```
while (expression)
{
    statement;
    statement;
    // Place as many statements here
    // as necessary.
}
```

The `while` loop works like an `if` statement that executes over and over. As long as the expression inside the parentheses is true, the conditionally executed statement or block will repeat. Program 5-3 uses the `while` loop to print “Hello” five times.

**Program 5-3**

```

1  // This program demonstrates a simple while loop.
2  #include <iostream>
3  using namespace std;
4
5  int main()
6  {
7      int number = 0;
8
9      while (number < 5)
10     {
11         cout << "Hello\n";
12         number++;
13     }
14     cout << "That's all!\n";
15     return 0;
16 }

```

**Program Output**

```

Hello
Hello
Hello
Hello
Hello
That's all!

```

Let's take a closer look at this program. In line 7 an integer variable, `number`, is defined and initialized with the value 0. In line 9 the `while` loop begins with this statement:

```
while (number < 5)
```

This statement tests the variable `number` to determine whether it is less than 5. If it is, then the statements in the body of the loop (lines 11 and 12) are executed:

```
cout << "Hello\n";
number++;
```

The statement in line 11 prints the word “Hello.” The statement in line 12 uses the increment operator to add one to `number`. This is the last statement in the body of the loop, so after it executes, the loop starts over. It tests the expression `number < 5` again, and if it is true, the statements in the body of the loop are executed again. This cycle repeats until the expression `number < 5` is false. This is illustrated in Figure 5-2.

Each repetition of a loop is known as an *iteration*. This loop will perform five iterations because the variable `number` is initialized with the value 0, and it is incremented each time the body of the loop is executed. When the expression `number < 5` is tested and found to be false, the loop will terminate and the program will resume execution at the statement that immediately follows the loop. Figure 5-3 shows the logic of this loop.

Figure 5-2

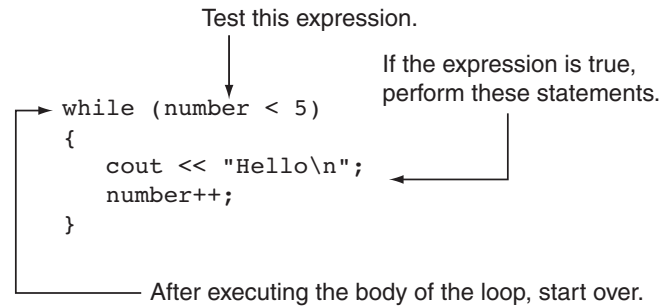
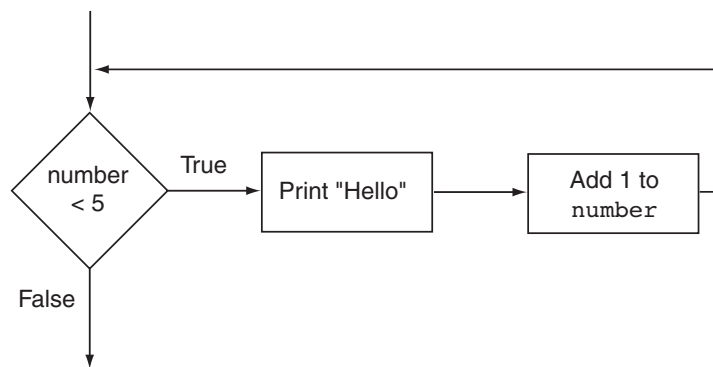


Figure 5-3



In this example, the `number` variable is referred to as the *loop control variable* because it controls the number of times that the loop iterates.

## The while Loop Is a Pretest Loop

The while loop is known as a *pretest* loop, which means it tests its expression before each iteration. Notice the variable definition in line 7 of Program 5-3:

```
int number = 0;
```

The `number` variable is initialized with the value 0. If `number` had been initialized with the value 5 or greater, as shown in the following program segment, the loop would never execute:

```
int number = 6;
while (number < 5)
{
    cout << "Hello\\n";
    number++;
}
```

An important characteristic of the while loop is that the loop will never iterate if the test expression is false to start with. If you want to be sure that a while loop executes the first time, you must initialize the relevant data in such a way that the test expression starts out as true.

## Infinite Loops

In all but rare cases, loops must contain within themselves a way to terminate. This means that something inside the loop must eventually make the test expression false. The loop in Program 5-3 stops when the expression `number < 5` is false.

If a loop does not have a way of stopping, it is called an infinite loop. An *infinite loop* continues to repeat until the program is interrupted. Here is an example of an infinite loop:

```
int number = 0;
while (number < 5)
{
    cout << "Hello\n";
}
```

This is an infinite loop because it does not contain a statement that changes the value of the `number` variable. Each time the expression `number < 5` is tested, `number` will contain the value 0.

It's also possible to create an infinite loop by accidentally placing a semicolon after the first line of the `while` loop. Here is an example:

```
int number = 0;
while (number < 5); // This semicolon is an ERROR!
{
    cout << "Hello\n";
    number++;
}
```

The semicolon at the end of the first line is assumed to be a null statement and disconnects the `while` statement from the block that comes after it. To the compiler, this loop looks like:

```
while (number < 5);
```

This `while` loop will forever execute the null statement, which does nothing. The program will appear to have “gone into space” because there is nothing to display screen output or show activity.

## Don't Forget the Braces with a Block of Statements

If you write a loop that conditionally executes a block of statements, don't forget to enclose all of the statements in a set of braces. If the braces are accidentally left out, the `while` statement conditionally executes only the very next statement. For example, look at the following code.

```
int number = 0;
// This loop is missing its braces!
while (number < 5)
    cout << "Hello\n";
    number++;
```

In this code the `number++` statement is not in the body of the loop. Because the braces are missing, the `while` statement only executes the statement that immediately follows it. This loop will execute infinitely because there is no code in its body that changes the `number` variable.

Another common pitfall with loops is accidentally using the = operator when you intend to use the == operator. The following is an infinite loop because the test expression assigns 1 to remainder each time it is evaluated instead of testing whether remainder is equal to 1.

```
while (remainder = 1) // Error: Notice the assignment
{
    cout << "Enter a number: ";
    cin >> num;
    remainder = num % 2;
}
```

Remember, any nonzero value is evaluated as true.

## Programming Style and the while Loop

It's possible to create loops that look like this:

```
while (number < 5) { cout << "Hello\n"; number++; }
```

Avoid this style of programming. The programming style you should use with the while loop is similar to that of the if statement:

- If there is only one statement repeated by the loop, it should appear on the line after the while statement and be indented one additional level.
- If the loop repeats a block, each line inside the braces should be indented.

This programming style should visually set the body of the loop apart from the surrounding code. In general, you'll find a similar style being used with the other types of loops presented in this chapter.

### In the Spotlight:

#### Designing a Program with a while Loop



A project currently underway at Chemical Labs, Inc. requires that a substance be continually heated in a vat. A technician must check the substance's temperature every 15 minutes. If the substance's temperature does not exceed 102.5 degrees Celsius, then the technician does nothing. However, if the temperature is greater than 102.5 degrees Celsius, the technician must turn down the vat's thermostat, wait 5 minutes, and check the temperature again. The technician repeats these steps until the temperature does not exceed 102.5 degrees Celsius. The director of engineering has asked you to write a program that guides the technician through this process.

Here is the algorithm:

1. Prompt the user to enter the substance's temperature.
2. Repeat the following steps as long as the temperature is greater than 102.5 degrees Celsius:
  - a. Tell the technician to turn down the thermostat, wait 5 minutes, and check the temperature again.
  - b. Prompt the user to enter the substance's temperature.
3. After the loop finishes, tell the technician that the temperature is acceptable and to check it again in 15 minutes.