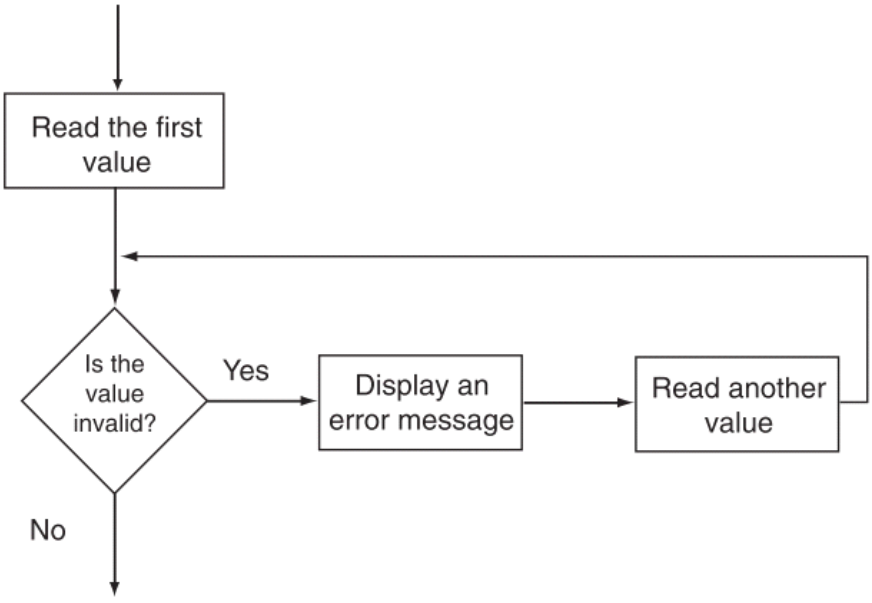
5.3 Using the `while` Loop for Input Validation

CONCEPT: The `while` loop can be used to create input routines that repeat until acceptable data is entered.

Perhaps the most famous saying of the computer industry is "garbage in, garbage out." The integrity of a program's output is only as good as its input, so you should try to make sure garbage does not go into your programs. *Input validation* is the process of inspecting data given to a program by the user and determining if it is valid. A good program should give clear instructions about the kind of input that is acceptable, and not assume the user has followed those instructions.

The `while` loop is especially useful for validating input. If an invalid value is entered, a loop can require that the user re-enter it as many times as necessary. For example, the following loop asks for a number in the range of 1 through 100:

Figure 5-3



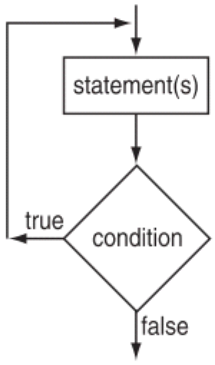
5.5 The `do-while` Loop

CONCEPT: The `do-while` loop is a post test loop, which means its expression is tested after each iteration.

In addition to the `while` loop, C++ also offers the `do-while` loop. The `do-while` loop looks similar to a `while` loop turned upside down. Figure 5-4 shows its format and a flowchart visually depicting how it works.

Figure 5-4

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



As with the `while` loop, if there is only one conditionally executed statement in the loop body, the braces may be omitted.

5.6 The `for` Loop

CONCEPT: **The `for` loop is a pretest loop that combines the initialization, testing, and updating of a loop control variable in a single loop header.**

VideoNote
The `for` Loop

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.

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 final value. When the counter variable reaches its final 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.

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

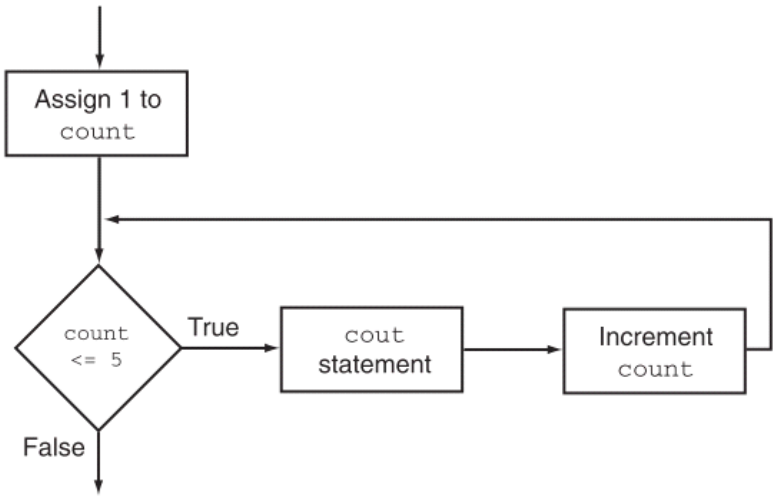
As with the other loops you have used, if there is only one statement in the loop body, the braces may be omitted.

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

```
for (count = 1; count <= 5; count++)
   cout << "Hello" << endl;
```

In this loop, the initialization expression is `count = 1`, 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 5-5 illustrates the sequence of events that take 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



5.8 Sentinels

CONCEPT: A *sentinel* is a special value that marks the end of a list of values.

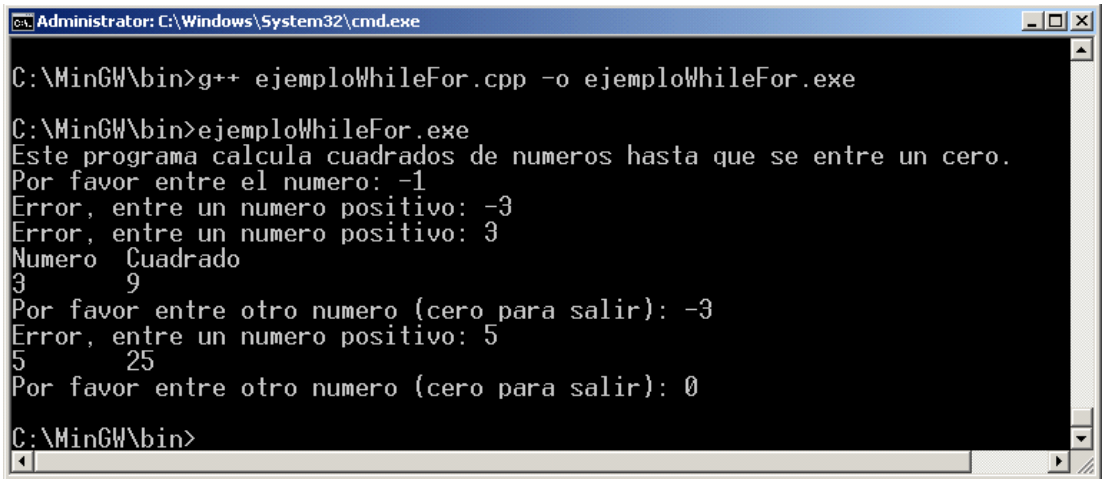
Program 5-10, in the previous section, requires the user to know in advance the number of days there are sales figures for. Sometimes the user has a list that is very long and doesn't know how many items there are. In other cases, the user might be entering several lists and it is impractical to require that every item in every list be counted.

A technique that can be used in these situations is to ask the user to enter a sentinel at the end of the list. A *sentinel* is a special value that cannot be mistaken as a member of the list and signals that there are no more values to be entered. When the user enters the sentinel, the loop terminates.

Program 5-11 provides an example of using an end sentinel. This program calculates the total points earned by a soccer team over a series of games. It allows the user to enter the series of game points, then enter –1 to signal the end of the list.

Ejercicios de clase:

```cpp
#include <iostream>
using namespace std;
int main(){
    int maxCount;
    int i;
    //cout << "Este programa calcula cuadrados de numeros del uno al que el usuario entre.\n";
    cout << "Este programa calcula cuadrados de numeros hasta que se entre un cero.\n";
    cout << "Por favor entre el numero: ";
    cin >> maxCount;
    while(maxCount < 0){
        cout << "Error, entre un numero positivo: ";
        cin >> maxCount;
    }//while

    //Usando do while para solucionar el problema
    cout << "Numero\tCuadrado\n";

    /*
    for(i=1;i<=maxCount;i++)
        cout << i <<"\t"<< i*i << endl;
    */



    /*
    i = 1;
    do{
        cout << i <<"\t"<< i*i << endl;
        i++;
    }while(i<=maxCount);
    */

    //i = 1;
    while(maxCount!=0){
        cout << maxCount <<"\t"<< maxCount*maxCount << endl;
        //i++;
        cout << "Por favor entre otro numero (cero para salir): ";
        cin >> maxCount;
        while(maxCount < 0){
            cout << "Error, entre un numero positivo: ";
            cin >> maxCount;
        }//while
    }



    return 0;
}//main
```


Administrator: C:\Windows\System32\cmd.exe
C:\MinGW\bin>g++ ejemploWhileFor.cpp -o ejemploWhileFor.exe
C:\MinGW\bin>ejemploWhileFor.exe
Este programa calcula cuadrados de numeros hasta que se entre un cero.
Por favor entre el numero: -1
Error, entre un numero positivo: -3
Error, entre un numero positivo: 3
Numero Cuadrado
3 9
Por favor entre otro numero (cero para salir): -3
Error, entre un numero positivo: 5
5 25
Por favor entre otro numero (cero para salir): 0
C:\MinGW\bin>