Looping in C++

Author: Dom Scola

Introduction

While programming in C++, looping can be an invaluable tool used to increase the functionality and overall use of your program. A loop is a control structure that can cause a statement or block of code to be executed repeatedly, or until a programmer-defined limit can terminate it. Loops are essentially used to repeat the same lines of codes over and over until some termination condition is met.

Looping is very useful in coding because it allows the user to continually execute the block of codes already written in the program. For example: if we have a program that asks the user to enter a price for an item being purchased and then use that price to calculate the sales tax associated with that price, implementing a loop into this program will allow us to calculate the price and sales tax of many items and even total the price of all these items onto a customer receipt. Instead of entering one item every time the program is executed, we can enter in as many items as the user would like, making the program much more efficient. For new and experienced programmers, the concept of looping in C++ makes code much more efficient and easier for others to read. Learning the ins and outs of looping allows users to build their programming skills and become more knowledgeable in C++.

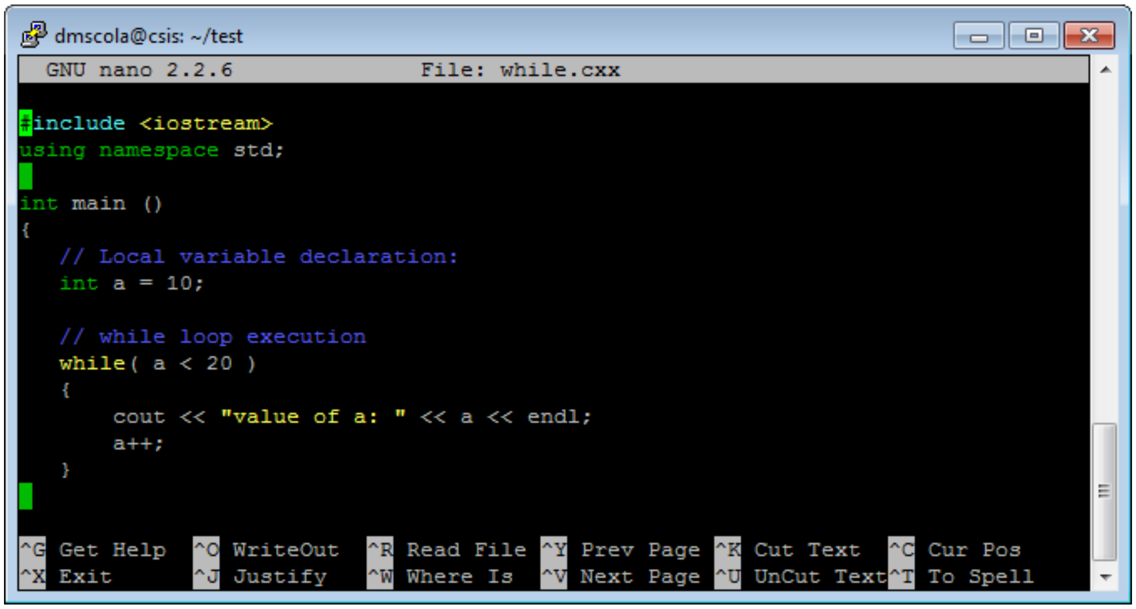
Parts of a Loop

Each time a loop is executed, we call this an **iteration**. When the flow of control of a program reaches the start of a loop, it will evaluate whether or not the **loop test**, or expression that determines whether or not another iteration will be executed, is true. If the loop test is true, the loop will execute whatever commands and expressions are located within the block of the loop. Once the loop test fails, the loop will no longer execute and will continue on to the next statement or expression following the loop. When the loop test fails, we call the expression or statement causing it to fail the **termination condition**.

Types of Loops in C++

There are three typical loops used within C++ programs: the **while loop**, the **for loop**, and the **do while loop**. Each type of loop uses different syntax and has a different structure than the others. These loops are not interchangeable; each one requires a different setup and placement of the block of code being executed within the loop. Programmers must be careful not to mix syntax of different loops together and to arrange their loops in a way that allows them to fulfill the purpose of their program more efficiently

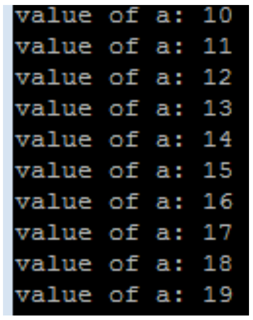
Description of While Loop



```
dmscola@csis: ~/test
  GNU nano 2.2.6                 File: while.cxx

#include <iostream>
using namespace std;

int main ()
{
   // Local variable declaration:
   int a = 10;

   // while loop execution
   while( a < 20 )
   {
       cout << "value of a: " << a << endl;
       a++;
   }


^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text^T To Spell
```

Figure 1: A while loop used to output the value of a variable "a" while the value stored in "a" is less than 20.

A **while loop** will repeatedly execute a statement or block of statements as long as the given starting condition is true. In Figure 1, the while loop will continue to execute as long as the variable 'a' is less than 20. Everything inside the brackets beneath the loop test starting with "while..." will execute as long as ‘a’ is less than 20. So the program will display "value of a: " and the current value of ‘a’, and then add one to the value of ‘a’. The program then moves back to the loop test and evaluates whether or not it is true. The value of ‘a’ is now 11, which is still less than 20, so another iteration of the loop is executed. This continues until the value of ‘a’ is no longer less than 20, and then continues to the next line of the program. In this case, because ‘a’ increases by one each time the loop is iterated, the termination condition of the loop would be a = 20.

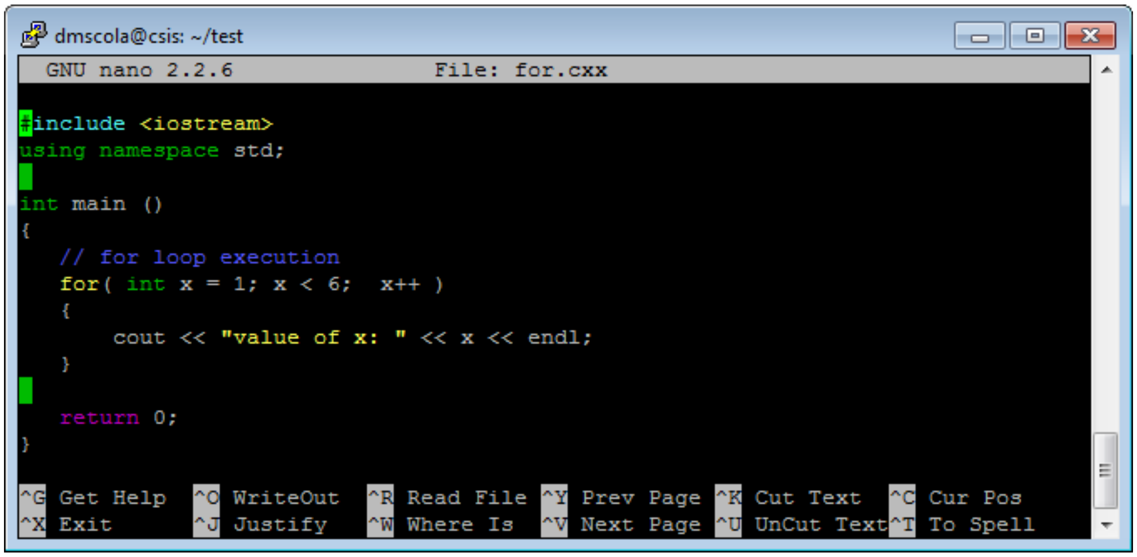


```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

Figure 2: The output of the program in Figure 1

The last line of output is “value of a: 19.” After this line, ‘a’ is incremented to 20. Because the programmer specified that a = 20 is the termination condition, when a reaches 20, it no longer enters the loop structure. It skips the instructions within the block and then proceeds to the next statement or execution of the program. If ‘a’ is not incremented within the loop, the termination condition would never be met, and the code would continue to loop endlessly.

Description of For Loop



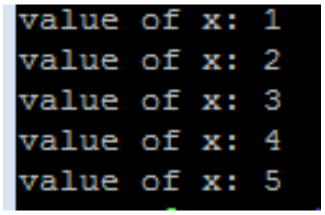
dmscola@csis: ~/test

```
  GNU nano 2.2.6                     File: for.cxx

#include <iostream>
using namespace std;

int main ()
{
   // for loop execution
   for( int x = 1; x < 6;  x++ )
   {
       cout << "value of x: " << x << endl;
   }

   return 0;
}

^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text^T To Spell
```

Figure 3: A for loop used to display the value of variable "x" while the value stored in "x" is less than 6.

A **for loop** is another type of loop and is especially useful when the programmer would like the loop to repeat a specified amount of times. In the above figure, the loop is controlled by a for statement. "Int x = 1;" defines and initializes the variable 'x' as an integer with value 1. The next part of the for statement, "x < 6;" is the loop test. As long as x is less than 6, the loop will iterate again. The last part "x++" is used to increase the count of how many times the loop is executed. If the programmer would like the statement enclosed in brackets "cout << "value of x: " << x << endl;" to execute a set number of times, he can determine this by setting the loop test to a certain value and incrementing the count until arriving at that point.

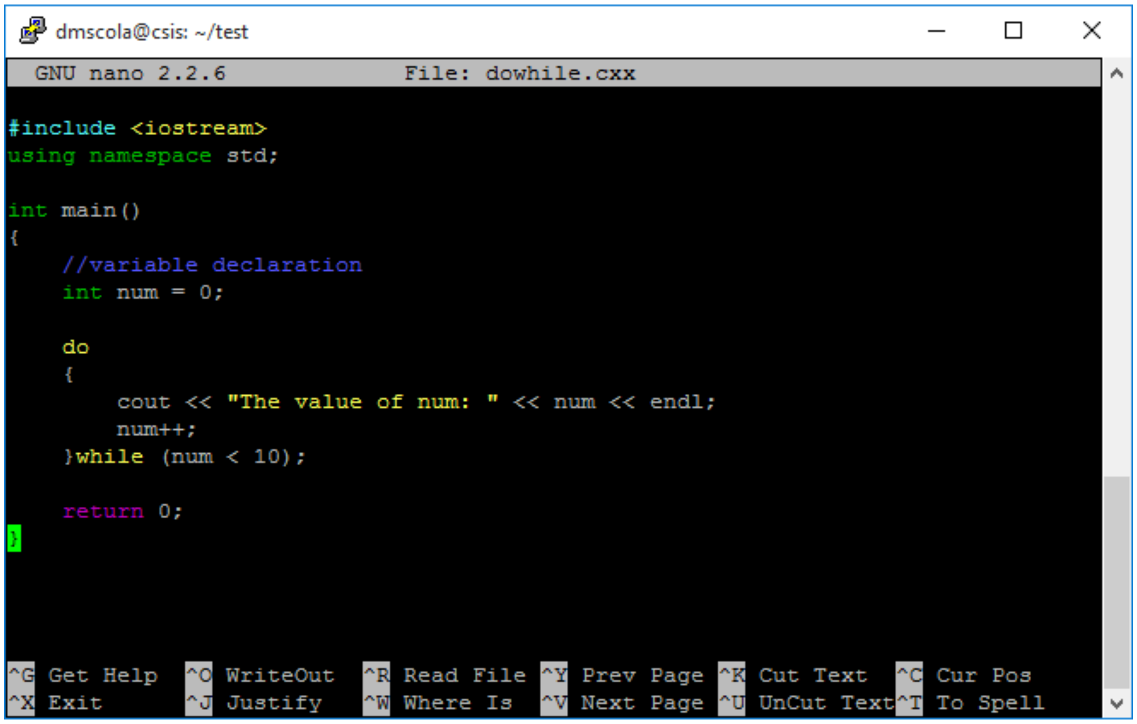
In Figure 3, the programmer is allowing the loop to iterate 5 times because the variable starts at 1 and will continue to execute until it reaches 6. We know that it will execute 5 times because 6 is the terminating value, and 1 is the initial value of x: 6 -1 = 5. Using a for statement allows the programmer to effectively dictate exactly how many times the loop will execute. The output of this code can be found in Figure 4. The code only executes for every value of x that is less than 6. After displaying the last line of output, the flow of control moves out of the loop, and continues on to the next executable statement.



```
value of x: 1
value of x: 2
value of x: 3
value of x: 4
value of x: 5
```

Figure 4: Output of code from Figure 3

Description of Do While Loop



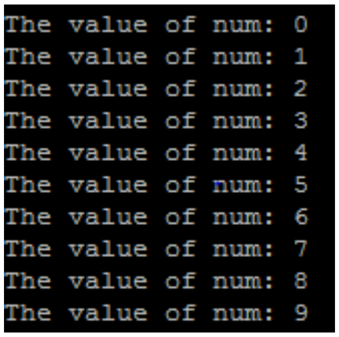
dmscola@csis: ~/test

```
  GNU nano 2.2.6              File: dowhile.cxx

#include <iostream>
using namespace std;

int main()
{
    //variable declaration
    int num = 0;

    do
    {
        cout << "The value of num: " << num << endl;
        num++;
    }while (num < 10);

    return 0;
}



^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text^T To Spell
```

Figure 5: A do while loop used to display the value of variable "num" while the value stored in "num" is less than 10.

The last type of loop that can be used is a **do while loop**. A do while is similar to a while loop, but the loop test is located at the bottom of the block of code that is being looped. The compiler will read the "do" statement and execute the statements located in the brackets before it reaches the loop test. This guarantees that block will be executed at least once. If the loop test is true, then the flow of control moves back to the top of the loop and executes again. Once the loop test fails and the termination condition is reached, then the compiler will simply move on to the next line of code. In Figure 5, the loop will iterate until num is no longer less than 10. However, the loop will iterate on the first iteration, even if the value of num was greater than 10.

The output of this code is shown in Figure 6. Once the variable reaches the termination condition, the flow of control moves out of the loop and continues on with the rest of the program. In this case, once the variable num was no longer less than 10, the compiler did no execute the block of code in the loop again.



```
The value of num: 0
The value of num: 1
The value of num: 2
The value of num: 3
The value of num: 4
The value of num: 5
The value of num: 6
The value of num: 7
The value of num: 8
The value of num: 9
```

Figure 6: Output of code from Figure 3

Which Loop to Use

While each of the loops will allow the user to repeat the same blocks of code over and over until some termination expression is reached, there are advantages of using specific types of loops according to your program's needs. Each of the three loops are slightly nuanced in their purpose. When deciding which type of loop is needed for a program, a programmer must assess what purpose their program is trying to fulfill. This assessment will allow the programmer to ultimately decide which loop is best suited for the needs of the program.

While Loop

A while loop is best used when testing for a condition to be true before the loop is evaluated. The block located within the loop will be completely skipped if the initial condition is false. For example: You are designing a program to calculate the sales tax while checking out at the grocery store. Prompt the user if they would like to checkout their items, allowing them to enter "Y" for yes or "N" for no into a char variable called "checkout". The loop test would be: while(checkout == 'Y'). Inside the loop block, allow the user to enter the price of the item, multiply it by the tax rate, add that price to the total, and then prompt the user if they would like to enter another item. When the user now enters 'Y' into the checkout variable, the loop will iterate again, but if the user enters 'N', the loop will end. If the user entered "N" the first time the program prompted them for checkout, the loop will be completely skipped.

Do While Loop

A do while loop can be used similarly to the while loop, but it has minor variances. The do while loop ensures that you will at least go through the loop once, so it is best suited for a program where you know the user is going to need the code block at least once. For example: in the grocery store program, the same code would be located in the block, except for a statement asking the user if this is the last item they would like to checkout. If the user enters 'Y' into the char variable last, then we know this would be the last item they are checking out. The loop test at the bottom would be while(last = 'Y'). If the user enters anything else, the loop will continue to iterate, but if the user declares the last item, it will still add the item's price and tax to the total.

For Loop

A for loop is best used when the programmer wants to make sure the user only enters the loop a specified number of times. The block will only execute as long as a counter is less than a certain value that the programmer establishes. In the grocery store example, imagine the store is offering a 50% discount on the first 5 items that are checked out. The loop test would be the following: For(i=0; i<5; i++). Inside the block, you can apply a 50% discount to the items, calculate their values and add it to the totals. At the end of block, the compiler will increment the variable "i" in the loop test, and evaluate it again. Once the fifth item is input, the loop will no longer run and then will move onto the rest of the code. The programmer specifies that they want exactly 5 iterations of the loop performed.

Additional Resources

The loop is an essential concept used in many different programming languages to simplify code and to avoid mistakes in syntax and program structure. Programmers can use the different types of loops to increase the functionality in their programs and to fulfill the purposes that their programs are designed to complete. For more information on the different types of loops and to see more examples of the uses of loops within programs, there are many online articles and guides for programming with loops in C++. Cprogramming.com and tutorialspoint.com have very useful guides to the understanding the structure and functionality of loops. Online forums like dreamincode.net are a useful tool for debugging and for help to better understand the syntax of loops within specific programs. Consider visiting these sites for more information on loops in C++.

Works Cited

Allain, Alex. "Lesson 3: Loops." *For, While, and Do While Loops in C++*. N.p., n.d. Web. 29 Nov. 2015.

"C++ Loop Types." *Www.tutorialspoint.com*. N.p., n.d. Web. 29 Nov. 2015.

"Statements and Flow Control." - *C++ Tutorials*. N.p., n.d. Web. 29 Nov. 2015.

"Types of Loops in C++." *Types of Loops in C++*. N.p., n.d. Web. 29 Nov. 2015.

"Understanding Loops In C++ - C++ Tutorials | Dream.In.Code." *Understanding Loops In C++ - C++ Tutorials | Dream.In.Code*. N.p., n.d. Web. 29 Nov. 2015.