7.7 — Intro to loops and while statements

ALEX AUGUST 30, 2021

Introduction to loops

And now the real fun begins -- in the next set of lessons, we'll cover loops. Loops are control flow constructs that allow a piece of code to execute repeatedly until some condition is met. Loops add a significant amount of flexibility into your programming toolkit, allowing you to do many things that would otherwise be difficult.

For example, let's say you wanted to print all the numbers between 1 and 10. Without loops, you might try something like this:

```
#include <iostream>

int main()
{
    std::cout << "1 2 3 4 5 6 7 8 9
10";
    std::cout << " done!";
    return 0;
}
```

While that's doable, it becomes increasingly less so as you want to print more numbers: what if you wanted to print all the numbers between 1 and 1000? That would be quite a bit of typing! But such a program is writable in this way because we know at compile time how many numbers we want to print.

Now, let's change the parameters a bit. What if we wanted to ask the user to enter a number and then print all the numbers between 1 and the number the user entered? The number the user will enter isn't knowable at compile-time. So how might we go about solving this?

While statements

The while statement (also called a while loop) is the simplest of the three loop types that C++ provides, and it has a definition very similar to that of an `if statement`:

```
while (condition)
    statement;
```

A `while statement` is declared using the while keyword. When a `while statement` is executed, the `condition` is evaluated. If the

condition evaluates to `true`, the associated statement executes.

However, unlike an `if statement`, once the statement has finished executing, control returns to the top of the `while statement` and the process is repeated. This means a `while statement` will keep looping for as long as the condition evaluates to `true`.

Let's take a look at a simple while loop that prints all the numbers from 1 to 10:

```
#include <iostream>

int main()
{
    int count{ 1 };
    while (count <= 10)
    {
        std::cout << count << '
';
        ++count;
    }

    std::cout << "done!";

    return 0;
}

```

This outputs:

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

Let's take a closer look at what this program is doing. First, `count` is initialized to `1`, which is the first number we'll print. The condition `count <= 10` is `true`, so the statement executes. In this case, our statement is a block, so all the statements in the block will execute. The first statement in the block prints `1` and a space, and the second increments `count` to 2. Control now returns back to the top of the `while statement`, and the condition is evaluated again. `2 <= 10` evaluates to true, so the code block is executed again. The loop will repeatedly execute until `count` is `11`, at which point `11 <= 10` will evaluate to `false`, and the statement associated with the loop will be skipped. At this point, the loop is done.

While this program is a bit more code than typing all the numbers between 1 and 10, consider how easy it would be to modify the program to print all the numbers between 1 and 1000: all you'd need to do is change `count <= 10` to `count <= 1000`.

While statements that evaluate to false initially

Note that if the condition initially evaluates to `false`, the associated statement will not execute at all. Consider the following program:

```
#include <iostream>

int main()
{
    int count{ 15 };
    while (count <= 10)
    {
        std::cout << count << '
';
        ++count;
    }

    std::cout << "done!";

    return 0;
}

```

The condition `15 <= 10` evaluates to `false`, so the associated statement is skipped. The program continues, and the only thing printed is `done!`.

Infinite loops

On the other hand, if the expression always evaluates to true, the while loop will execute forever. This is called an infinite loop. Here is an example of an infinite loop:

```cpp
#include <iostream>

int main()
{
    int count{ 1 };
    while (count <= 10) // this condition will never be false
    {
        std::cout << count << ' '; // so this line will repeatedly execute
    }

    return 0; // this line will never execute
}
```

Because `count` is never incremented in this program, `count <= 10` will always be true. Consequently, the loop will never terminate, and the program will print "1 1 1 1 1"... forever.

Intentional infinite loops

We can declare an intentional infinite loop like this:

```cpp
while (true)
{
  // this loop will execute forever
}
```

The only way to exit an infinite loop is through a return statement, a break statement, an exit statement, a goto statement, an exception being thrown, or the user killing the program.

Here's a silly example demonstrating this:

```cpp
#include <iostream>

int main()
{

    while (true) // infinite loop
    {
        std::cout << "Loop again (y/n)?
";
        char c{};
        std::cin >> c;

        if (c == 'n')
            return 0;
    }

    return 0;
}

```

This program will continuously loop until the user enters `n` as input, at which point the `if statement` will evaluate to `true` and the associated `return 0;` will cause function `main()` to exit, terminating the program.

It is common to see this kind of loop in web server applications that run continuously and service web requests.

Best practice

Favor `while(true)` for intentional infinite loops.

Loop variables

Often, we want a loop to execute a certain number of times. To do this, it is common to use a loop variable, often called a counter. A loop variable is an integer that is used to count how many times a loop has executed. In the examples above, the variable `count` is a loop variable.

Loop variables are often given simple names, such as `i`, `j`, or `k`. However, if you want to know where in your program a loop variable is used, and you use the search function on `i`, `j`, or `k`, the search function will return half your program! For this reason, some developers prefer loop variable names like `iii`, `jjj`, or `kkk`. Because these names are more unique, this makes searching for loop variables much easier, and helps them stand out as loop variables. An even better idea is to use "real" variable names, such as `count`, or a name that gives more detail about what you're counting (e.g. `userCount`).

Loop variables should be signed

Loop variables should almost always be signed, as unsigned integers can lead to unexpected issues. Consider the following code:

```
1  #include <iostream>
2  int main()
3  {
4      unsigned int count{ 10 };
5
       // count from 10 down to 0
       while (count >= 0)
6      {
7          if (count == 0)
           {
               std::cout <<
8  "blastoff!";
           }
           else
9          {
10             std::cout << count << '
   ';
           }
11         --count;
12     }

       return 0;
   }
13
```

Take a look at the above example and see if you can spot the error. It's not very obvious.

It turns out, this program is an infinite loop. It starts out by printing `10 9 8 7 6 5 4 3 2 1 blastoff!` as desired, but then goes off the rails, and starts counting down from `4294967295`. Why? Because the loop condition `count >= 0` will never be false! When count is `0`, `0 >= 0` is true. Then `--count` is executed, and count wraps around back to `4294967295` (Assuming 32-bit integers). And since `4294967295 >= 0` is true, the program continues. Because `count` is unsigned, it can never be negative, and because it can never be negative, the loop won't terminate.

Best practice

Loop variables should be of type (signed) int.

Doing something every N iterations

Each time a loop executes, it is called aniteration.

Often, we want to do something every 2nd, 3rd, or 4th iteration, such as print a newline. This can easily be done by using the modulus operator on our counter:

```
#include <iostream>

// Iterate through every number between 1 and 50
int main()
{
    int count{ 1 };
    while (count <= 50)
    {
        // print the number (pad numbers under 10 with a leading 0 for formatting purposes)
        if (count < 10)
        {
            std::cout << '0';
        }

        std::cout << count << ' ';

        // if the loop variable is divisible by 10, print a newline
        if (count % 10 == 0)
        {
            std::cout << '\n';
        }

        // increment the loop counter
        ++count;
    }

    return 0;
}
```

This program produces the result:

```
01 02 03 04 05 06 07 08 09 10
11 12 13 14 15 16 17 18 19 20
21 22 23 24 25 26 27 28 29 30
31 32 33 34 35 36 37 38 39 40
41 42 43 44 45 46 47 48 49 50
```

Nested loops

It is also possible to nest loops inside of other loops. In the following example, the inner loop and outer loop each have their own counters. However, note that the loop expression for the inner loop makes use of the outer loop's counter as well!

```cpp
#include <iostream>

// Loop between 1 and 5
int main()
{
    int outer{ 1 };
    while (outer <= 5)
    {
        // loop between 1 and outer
        int inner{ 1 };
        while (inner <= outer)
        {
            std::cout << inner << ' ';
            ++inner;
        }

        // print a newline at the end of each row
        std::cout << '\n';
        ++outer;
    }

    return 0;
}
```

This program prints:

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

Quiz time

Question #1

In the above program, why is variable `inner` declared inside the while block instead of immediately following the declaration of `outer`?

Show Solution

Question #2

Write a program that prints out the letters a through z along with their ASCII codes.

Show Hint

Show Solution

Question #3

Invert the nested loops example so it prints the following:

```
5 4 3 2 1
4 3 2 1
3 2 1
2 1
1
```

Show Solution

Question #4

Now make the numbers print like this:

```
        1
      2 1
    3 2 1
  4 3 2 1
5 4 3 2 1
```

Hint: Figure out how to make it print like this first:

```
X X X X 1
X X X 2 1
X X 3 2 1
X 4 3 2 1
5 4 3 2 1
```

Show Solution

Next lesson
7.8 Do while statements

Back to table of contents

Previous lesson
7.6 Goto statements

@ Email*

Avatars from https://gravatar.com/ are connected to your provided email address.

Notify me about replies: POST COMMENT

DP N N FOUT

Newest