TutorTube: Loops in C++

Fall 2020

Introduction

Hello, and welcome to TutorTube! Where the Learning Center's Lead Tutors help you understand challenging course concepts with easy to understand videos. My name is Matt Curtin, lead tutor for computer science. In today's video, we will be exploring for loops and while loops in C++. Let's get ahead and get started.

Recalling If Statements

Let's begin talking about loops by recalling what we know about branches. I'm going to set up an example. Let's say that an integer **x** is equal to five. If **x** is less than ten, then **cout** "**x** is less than 10."

We supply our **if** statement with a condition. That's right here. If this condition is true, then we execute the instructions between the brackets: the **cout** statement. After we execute the instructions in between the brackets, we then exit the **if** statement. So, once this is done, we just go on with the rest of our program. We do the next line, and the next line, and the next line. Let's run this code and see what happens.

We're going to compile and run. You see that "5 is less than 10," so the branch executes and we get this output.

While Loops

While loops are very similar but a little different to **if** statements. We write **while** loops almost exactly like an **if** statement, the only difference is that we write "while" instead of "if," so all you have to do to make a **while** loop is to make an **if** statement and switch out "if" with "while." And there we go! We have a **while** loop. So, just like an **if** statement, our **while** loop has a condition. The instructions in between the brackets will execute if the condition is true, so this will happen if this is true just like our **if** statement. The only difference is is that once the instructions in between the brackets execute, we go back up to the condition, we check if it's true, and if it's true, we run it again. Once we run this again, we go back up to the condition and see if it's true. Let's say it's not true. In this case if this isn't true then we skip the brackets and keep going on with our program.

Let's compile and run and see what happens. We notice that there's an infinite loop. It's pretty crazy. Why does this happen? Well, let's look at our code and find out.

If you notice, our condition is dependent on this variable **x**. However, **x** never changes, so if this condition is true from the start then it'll be true for every iteration of our **while** loop. We notice that we start with **x = 5**, but **x** never changes values. So, to fix that, let's add some instructions inside of our while loop. Let's increase **x** by 1 every time the loop executes. So, if **x** is less than 10, we're going to print out "x is less than 10" and then increment **x** by 1. We're gonna make it one more bigger.

Let's compile and run. We don't have an infinite loop anymore which is pretty cool, and also our loop works as intended. We have "5 is less than 10," "6 is less than 10," "7 is less than 10," "8 is less than 10," and "9 is less than 10." When we are at nine...

Once we got to nine, then we **cout** "9 is less than ten" and we incremented nine plus plus, so we get ten. **x** is now equal to ten. If **x** is equal to 10, **x** can't be less than 10, so the loop exited, and that's why we only got up to "9 is less than 10" and not like "10 is less than 10".

For Loops

Let's now take a look at **for** loops. A **for** loop is very similar to a **while** loop. There are a few key differences though. The **for** loop is made up of three parts: the first one is the starting statement, the second one is the condition, and the third one is the ending statement.

The starting statement is executed before the loop runs. So, happens before the loop starts. The condition is checked at every iteration of the loop, and if it's true, the loop will execute. If it's false the loop won't execute. And - oops misspelled "ending" - and the ending statement happens after after every iteration of the loop. We'll see more examples of this further on in the video, but for right now I just want you to keep this in the back of your head.

One thing also about **for** loops is that they have something called an indexing variable. You can kind of think of this as a counter. For us in this while loop up here, we used **x** as a variable and it counted up. **For** loops usually have **i**. It's just the standard. It's a computer science convention to use **i** in your **for** loops. So, let's take a look at what that looks like.

So to declare a **for** loop, you have to type "for", and open parentheses. And after the open parenthesis you need to put the starting statement. So, I'm going to say "**int i = 0**". I declare an integer **i** and set it equal to zero. This will run before my loop starts.

The next thing I'm going to type is the condition. I'm going to type "**i < 10**". This condition will be checked every iteration of the loop to see if it executes.

The next thing I'm going to type is the ending statement: "**i++**". This statement happens after every iteration of the loop.

I'll show you what happens and walk you through the code and how it's executed and in what order in just a minute. I'll also explain what all of these are doing, but for right now we're just going to dive feet first into the deep end and see where we land.

So, then after the **for** loop you put your brackets. The brackets is where you put what happens if when the loop executes. I'm just going to "**cout << i**" so whatever the value of **i** is at that iteration of the loop is what we're going to print out.

Let's look at these statements one by one and try to guess what will happen when we run the code. So, we're right here in the program, just before the **for** loop starts. Once we get to the **for** loop, we look at this first statement "**int i = 0**". Within the **for** loop we've created a variable **i** and set it equal to zero. The next thing we do is check the condition. Is **i** less than ten? Well, 0 is less than 10, so the condition is true. Because the condition is true, the thing in between the brackets executes. That's the **cout** statement. So, we're going to print out what i is at that current iteration. Once we get to the end, instead of exiting the **for** loop we go back up to the top and that's where this third statement comes in. We do **i++**. We're going to increase **i** by one.

After we execute this ending statement, we're gonna go back to the condition. Is **i** still less than 10? Well, now **i** is one. One is less than ten, so the **cout** will execute. After we're done with printing this to the console, we're gonna **i++** again and then check the condition and we're gonna go all the way up until this condition turns to false.

Let's see what happens when we compile and run. So, here we have what our **while** loop was doing before. "5 is less than 10" ... so that was our **while** loop. Here, look at our **for** loop. We go from 0 1 2 3 4 5 6 7 8 9. So, just like last time, instead of going to 10 we stopped at 9, and that's because when **i** is equal to 10, 10 is not less than 10, and so the loop exits.

Changing Starting and Stopping Points

We can try out different starting and stopping points in our **for** loop. **For** loops are really good at going from a specific starting number to a specific ending number. What I'm going to do is instead of saying "**int i = 0**", I'm gonna start at 25,

and what I want to do is go all the way up to 32. So your first inclination might be to put a 32 here. If i want to print out the value 32, then I need to take a little bit more time to consider. If **i** is equal to 32, then that means 32 cannot be less than 32, so this never runs. What I need to do is either do "**i <= 32**" so that I can include 32, or I can say "**i < 33**". However you want to do it. To me it makes more sense to do "**i <= 32**", but it is your preference.

So, let's see what happens when we compile and run this. Yeah, so we start at 25 and we end at 32. This shows that for loops are very very nice when it comes to starting at a certain spot and ending at a certain spot.

Outro

Thank you for watching TutorTube! I hope you enjoyed this video. Please subscribe to our channel for more exciting videos. Check out the links in the description below for more information about The Learning Center and follow us on social media. See ya!

Code

```
#include <iostream>

using namespace std;

int main(){

    //Let's say that an integer x is equal to five

    int x = 5;


    //If x < 10 then we are going to cout the value of x is less than 10

    if( x < 10 ){

        cout << x << " is less than 10" << endl;

    }


    //Changing the if statement to a while loop

    while(x < 10){

        cout << x << " is less than 5" << endl;

    }
```

```
	//Removing infinite loop
	while(x < 10){
		cout << x << " is less than 5" << endl;
		x++;
	}

	//Making a similar for loop
	for(int i = 5; i < 10; i++){
		cout << "i = " << i << endl;
	}

	//We can change where the for loop starts and stops
	for(int i = 25; i <=32; i++){
		cout << "i = " << i << endl;
	}
	return 0;
}
```