Pre-Lab 05: Loops and Iterations

CIIC4010 / ICOM4015

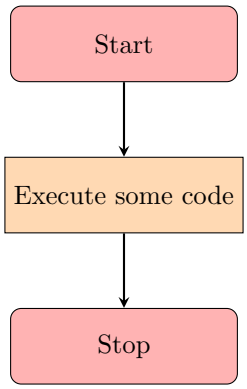
September 2020

1 Before Tackling Loops

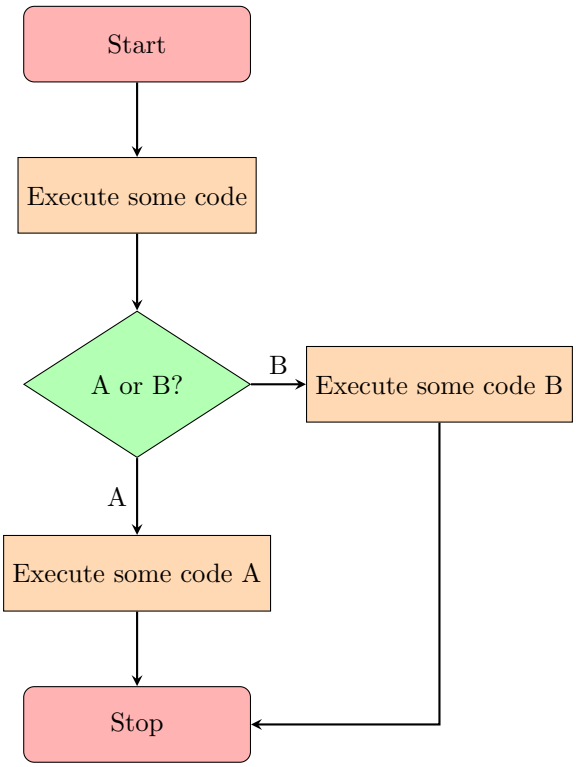
Remember the content of this course is cumulative. What has been covered in previous labs and lectures may be of utmost importance in this current lab. Keeping that in mind, if you are rusty on things such as objects or conditionals, now may be the time to brush up on those things. That being said, let's begin to explore loops!

2 Why Use Loops?

Think of the previous labs we have done in this course. If you were to chart out the execution of the programs you have written thus far, it would be pretty linear.



You may think that adding conditionals breaks this linearity. This is only partially true: Though the end result of your code may differ, your code is still linear in the sense that it goes only in one direction.



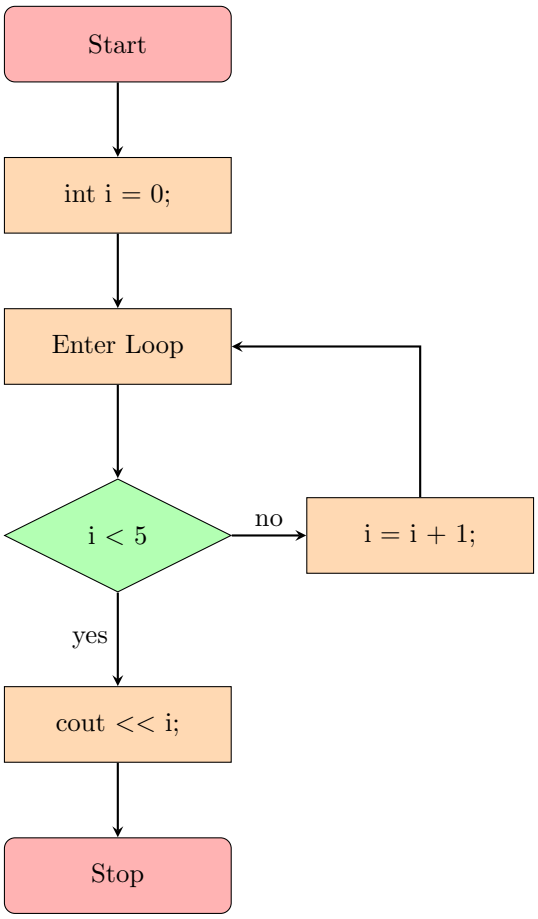
What if you wanted to execute a single block of code five times? At such a small scale it's not a big deal to write the same line of code five times. However, what if you wanted to execute it ten times? One hundred times? As scale grows, it is not feasible to manually write out these executions. Additionally, what if you wanted to control the amount of times a code block is executed depending on other variables? Quickly we realize writing these repeated blocks by hand is not realistic.

This is where loops come in to the rescue!

3 The Wonderful World of Loops

Before delving into loops, it may be useful to understand them with real-life examples. We could say the process of washing plates is done so by a loop: while there are dishes in the sink, wash a plate.

Loops are a way to control how many times a certain block of code is executed. It provides us an easy way to write code that may execute repetitive tasks, without making writing it a daunting one. Though there are many different types of loops, the principle is the same. Provide the loop a repeating condition, a block of code to execute and voila!



Above we can see a simple program that utilizes a loop. The code begins with an arbitrary variable i initialized at zero. If we wanted to get to i to be equal to five, but for x or y reason we could only add by ones, we could easily accomplish this by writing a loop. We provide the loop a working condition, and something to do while that is true. In this case, we tell the loop to execute while i is less than five, and to add 1 to i while it's at it.

The principle is simple, but it's a complete game changer for the power our code now has.

4 Types of Loops and Syntax

Though all loops aid us in the iteration of things, there are many different ways to write them. Different types of loops may be useful in different situations. In the case of C++ there are four types: the while loop, the do-while loop, the for loop, and the enhanced for loop.

4.1 The While Loop

The while loop's functionality is to the point: it tests a condition and if it is true, it executes the block of code within it. Observe the syntactical example below.

```
while (condition) {
  // code block to be executed
}
```

This is the basic syntax of a while loop. As long as the condition given is true, the code block, no matter what it is, will be executed. While loops are great if you are dealing with non-numerical conditions.

4.2 The Do-While Loop

A do-while loop is very similar to a while loop. However, a do-while, will always execute the code block once before checking the condition. A syntactical example is given below.

```
do {
  // code block to be executed
}
while (condition);
```

If you know you want your code block to be executed at least once every time, a do-while loop is an elegant way to do so.

4.3 The For Loop

Now we have the for loop. A for loop is like a version of a while loop, and can make keeping track of counter variables easy. Observe the syntactical example below.

```
for (statement 1; condition; statement 2) {
  // code block to be executed
}
```

Statement 1 will be executed once before entering the loop. The condition is defined next. Finally, statement 2 will be executed every time after the code block is executed. For loops are very useful if you are using a counter to keep track of how many times you are iterating. It is worth noting that for loops can be written as while loops using the following syntax.

```
statement 1;
while (condition) {
  // code block to be executed
  statement 2;
}
```

4.4 The Enhanced For Loop

Enhanced for loops are for loops that cut out the counter variable middle man. If you are working with a collection that supports it, such as a vector, it can make code more readable and straight to the point. Observe the syntactical example below.

```
for (DataType x : v) {
  // code block to be executed
}
```

In which v is a vector and x is an object of type DataType.

5 Loops and Vectors and Enums, Oh My!

In this lab, we will also be covering vectors and enums. Both are very powerful tools you may find yourself using throughout your computer science career! Vectors are a handy way to store data, while enums aid in storing constants.

5.1 Vectors

Though in this lab you will not be required to create vectors, it is still of importance to know how to use them. Methods such as erase(), size(), and more may need to be used in the lab. In order to get familiarized with the methods, visit, the C++ documentation here (`http://www.cplusplus.com/reference/vector/vector/`).

5.2 Enums

Enums are particularly useful if we want a static set of values. For example, if we wanted to represent days in a week in a program, it may be useful to use an enum to represent them. After all, the days in a week are a static thing that do not change (at least not for the foreseeable future).

```
#include <iostream>
using namespace std;
enum days {Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday};
int main(){
   days day;
   day = Thursday;
   cout << day;
   return 0;
}
```

If we were to run the code above, we would find a 3 in the standard out. This is the default value of the enum. Enums are great for error checking. If you were to assign an invalid value to an enum variable, it would cause a compilation error.