**Loops and Statements**

**Loops**

Loops are very useful in coding. There are three main loops we will cover here:

* For Loops
* While Loops
* Do Loops

**For Loops**

For loops allow you to repeat a particular section of code a set number of times. Identical to Java, the for loop has three sections; initialisation, testing condition, and operation.

Initialisation is called when the for loop starts for the first time. It is only called once and is useful for initialising variables (such as a counter) that the for loop might need. You don’t have to do it here, but there are very few places where it’s worth initialising the counter anywhere else.

The testing condition is what the for loop checks against to see if it does another loop. If the statement is true, it loops again. Otherwise the for loop closes and the program continues.

The operation is performed after each successful loop. Usually it is used to increment or decrement a counter but you can really perform any operation here if you want. The following are a couple of examples of for loops in C++ code.



For loops are very useful for repeating code with slight variations in data. Another good tip is if you have to copy and paste code a large number of times, or have a section that contains a lot of repeating code, then in all likelihood, we can replace it with a for loop.

**While Loops**

While loops are sections of code that loop while a particular test condition is true. If the condition ever turns false, the loop ends. The last example of the for loop acts in a similar manner to a while loop, though in that case the loop will never end. Similar to for loops, while loop are identical to their implementations in Java and C#. Below are some examples:



**Do Loop**

A do loop and a while loop are closely related; the difference being a while loop checks the test condition first then executes the code, whereas a do loop does it in reverse; the code is run once then the test condition is checked. It can be initialised like so:



The other different is the presence of a semicolon at the end of the loop, which is not present in the definition of a normal while loop.

**Statements**

Statements in code are very useful for controlling the flow of the code. There are three ones we will cover here:

* If Statements
* Switch Statements
* Try-Catch Statements

**If Statement**

If statements are some of the most common statements you will encounter when coding. Almost all languages have some an if statement in one form or another. C++ handles if statements the same as Java and C#. Each if statement has a single test condition. If it is true, the statement runs the code, if not, it will move on.



We can add else statements afterwards to catch cases where the if statement fails. We can also join else and a new if statement together to create a chain.



You can join as many else if’s together as you want but it can become ungainly and hard to manage if you use too many. This brings us nicely into the next statement, the switch statement.

**Switch Statement**

A switch statement is a method of choosing between multiple sections of code based on a *test condition*. Switch statements in C++ are again, very similar to their counterparts in C# and Java. The implementation is as follows:



The types you can use with switch statements in C++ are either ints or enums. You can also use chars as they can be converted very easily to ints and classes that are implicitly convertible to an int.



**Try-Catch Statement**

C# and Java make heavy use of try-catch statement to handle potentially volatile code. C++ can do the same though the use is not quite as widespread. Still it is a very useful tool to have and is necessary with certain libraries such as C++ Accelerated Massive Parallelism (C++ AMP). The implementation of a try-catch statement is as follows:



As we can see above, the first part of the statement is the try section. In here we attempt to run some code that may or may not work. This could be something such as a network request that may time out or fail to connect. Should this happen we throw the appropriate exception, which moves us on to the next section.

The catch section handles the exceptions thrown during the try. If no exceptions are thrown then we never enter the catch. We can chain catches as well so we can handle specific exceptions as they are thrown.



**Exercises**