This guide contains most of the basics of C++ (or programming in general), and I hope you learn from it!

Create a Project

1. Open Visual Studio
2. Click “Clone or check out code”
3. Paste in the link to the repository (https://github.com/WesleyMatthews/Hour-of-CPP.git) and select a path
4. Click “Clone”

Hello World

You will need to right click on “HelloWorld” in the Solution Explorer, and select “Set as StartUp Project”. Then open “HelloWorld.cpp”.

At the top of the code, we have the line `#include <iostream>`, which tells the computer that we will be using the `iostream` library. A C++ **library** is a .cpp file that is meant for use across a variety of programs.

Below that, we define a function with the line `int main()`, which creates a function called `main`. A **function** is a block of code that can be run (other terms include **call** and invoke). Every C++ program as a `main` function, as it is the function the program will run from the start. The next line has a `{`, which tells the **compiler** that this is where the `main` function starts. At the end of the function there is a matching `}`.

Inside of the `main` function, there is a line of code that says `std::cout << “Hello World!\n”;`. This prints “Hello World” to

the **console**, along with a new line (represented in the string as `\n`, which is the new line character).

`std::` tells the program to use the standard input and output library, and `cout <<` tells the program to print to the console. `"Hello World!\n"` says what to print to the console, and `;` ends the statement. Semicolons are always required at the end of a **statement**, which is a piece of code that defines a variable, calls a function, etc.

The last line of code in the `main` function is `return 0;`, which tells the `main` function to stop running, and gives a value of 0, which is an `int` (integer).

Now, run the program by clicking the "Local Windows Debugger" (next to the green play icon) at the top of the screen.

Math and Variables

You will need to right click on "MathAndVariables" in the Solution Explorer, and select "Set as StartUp Project". Then open "MathAndVariables.cpp".

A quick side note, you may have noticed **comments** in the code. These are either single line, starting with `//`, or multiline, starting with `/*` and ending with `*/`. The computer ignores these comments, as they are meant for humans.

Looking into the code, there is an `add` function defined at the top. Notice how we say `int add(int a, int b)`, which tells the compiler that the function has a **return type** of `int`, and two parameters of type `int`. This function will take two integers (`a` and `b`), add them, and return the result. It does this first by saying `return`, which signifies the start of a **return statement**, and then by saying `a + b`. Once again, the `;` just signifies the end of a statement.

Exercise 1 - Write a `subtract` function that takes two integers and returns the result.

Next we move into the main function, where we create a **variable** called `c` that has a value of `1 + 2`. You can think of variables as a container that stores a value, such as a box that contains cereal. Notice how we call the `add` function, using `add(1, 2)`. In this case, we are storing the value returned by calling `add(1, 2)` into the variable `c`. When calling a function, specify the name of the function (`add`), followed by parentheses. Arguments (in this case, `a = 1` and `b = 2`) go inside of the parentheses.

When we defined the `add` function, we created two parameters called `a` and `b`. In order to call the function, we need to give it two arguments, or values, for `a` and `b`. A **parameter** can be seen as the variable that a function needs, whereas an **argument** is the value of that variable.

The result of calling the `add` function is stored in the variable `c`. `c` is defined as an integer with the value `add(1, 2)`, or 3.

Exercise 2 - Create a variable `d` that stores the result of calling `subtract(c, 1)`. Note that by saying `c`, we are using the value stored in the variable `c`, which is 3.

Conditions

You will need to right click on “Conditions” in the Solution Explorer, and select “Set as StartUp Project”. Then open “Conditions.cpp”.

At the start of the main function, we declare that a double name x is equal to 3.5. Double is another variable or **data type** that is like an integer, but supports decimal points. There are many different

data types, but some common ones are: `int`, `double`, `string` (`std::string`), and `boolean` (true or false).

This program contains **conditional statements** (which begin `{` with and end with `}` instead of just ending with `;`). The logic is structured similar to if this is true...then this will happen. Conditional statements start with `if (condition)`, where the code is run if `condition` is true. `condition` could be `x > 1`, `9 < 1 || 3 == 2`, or anything else that returns `true` or `false`. However, you may need to check if something else is true but the original condition is not.

You can do this using `else if` and `else`. `else if` is used the same as `if`, but can only be right after an `if` statement. `else` does not need a condition, as it will run if all of the preceding `if` and `else if` statements are false.

There are different conditions using boolean logic, such as `x > 1` or `9 < 1 || 3 == 2`. In the first example, the boolean operator is `>`, which checks if a value is greater than another value. The second example includes the `||` (or) operator, which checks if one condition is true, or the other. There is a great resource to learn about C++ boolean operators here (https://www.tutorialspoint.com/cplusplus/cpp_operators.htm).

Exercise 3 - Try running the code with "Local Windows Debugger". What is the result? Now, change the value of `x` and see how that affects what the console prints.

Loops

You will need to right click on "Loops" in the Solution Explorer, and select "Set as StartUp Project". Then open "Loops.cpp".

Loops are a good way to repeat something multiple times. There are technically 3 types of loops, but we will only talk about `for` and `while` loops. They each have different uses, highlighted below.

`for` loops are used to repeat something a known amount of times, such saying "Hello" five times. For loops are started like `for (variable; condition; change`. `variable` determines what variable is used in the loop, such as `int i = 0`. `condition` determines if the loop still runs, such as `i < 5`. `change` determines how the variable will change, such as `i++` or `++i` (`++` increases the variable by one. The order does not matter in loops, but in more complex programs it does matter). The code inside of the `{}` will be run as long as `condition` remains true.

`while` loops are structured similarly to an `if` statement, starting with `while(condition)`, and ending with code inside of `{}`. `while` loops are used to repeat something an unknown amount of times, rather than a known amount of times (such as 5 in the above example). The code going along with this guide shows how to use `while` loops in the above example.

Arrays

You will need to right click on "Arrays" in the Solution Explorer, and select "Set as StartUp Project". Then open "Arrays.cpp".

Arrays can be complicated, but at the most basic level they are a list of items, similar to a shopping cart that contains multiple items. In the example, we have an array of fruits with a length of 3, and an array of vegetables with an undefined length (meaning as many items as there are when the array is defined). Arrays can be made like so: `type name[length] = { items, separated, by, commas }`.

Often, `for` loops are used with arrays to do something. In the example, each fruit is printed to the console. This can be done by getting the length of the array: `sizeof(array) / sizeof(array[0])`, and looping as long as i < the length. Notice how we say `array[0]`, indicating we want the first element of the array. In this case, 0 is the **index**, or position of an element in the array. Array indices start from 0, and count upwards. We also call the `sizeof` function to determine how much memory the array takes up, and divide it by how much memory one element takes up.

Glossary

Argument - A value given to a function parameter.

Array - A list of elements or items.

Call - The same as running or invoking a function. The function's code will be run.

Comment - Text that is ignored by the computer and meant for humans to read.

Compiler - A computer program that turns code into an executable file.

Conditional Statement - A statement that runs code under a given condition.

Console - A program that can show text output.

Data Type - Determines what type a given value is, such as `int` for integer, or `bool` for boolean.

Function - A group of code that can be run in multiple places, and generally used to reduce repetition in code.

Index - The position of an element in an array, starting from 0 and increasing from there.

Library - A .cpp file that adds more functions to a program to use.

Loop - Code that repeats a certain number of times (`for` loop), or until a certain condition is met (`while` loop).

Parameter - A variable used by a function it is called.

Return Statement - A statement that exits a function, returning a particular value or none at all if the return type is `void`.

Return Type - The data type a function will return. `void` if not returning a value.

Statement - Generally a line of code ending in a `;`.

Variable - A container (in memory) used to store a value.