C++ Coding Bootcamp

Session 1

Approach

- Hand-in-hand with Mick's approach of getting us there quick.
- Since its mostly beginners here, we'll go slow and sometimes in excruciating detail. The idea is to fill in around Mick's teaching.
- Should help in the final projects.
- Lots of live programming, follow along!
- We'll approach C++ ideas in a specific order - variables, conditionals, loops, functions, objects (with a little STL mixed in here and there).
- We'll also get into conventions and nomenclature at every step - so you know what to google.

Love myself a terrible stock photo

Why C++?

- Because we have to for our masters - it's a terrible language (still better than plain C).
- But the overspecified nature also gives us more control.
- And makes the language really really fast.
- Also, once you do C++, there’s nothing you can’t understand anymore (loosely tested by yours truly)
- Multi-platform support.

What is C++?

- Made in 1979 as an extension of C by this guy (who probably doesn't realise how much of a nerd he is in this picture)
- Unlike C, it supports classes and objects natively.
- It also has the STL (Standard Template Library) which makes an awful lot of things much easier/better to do.
- eg: Arrays in C vs. Vectors in C++
  Better strings!

Bjarne Stroustrup

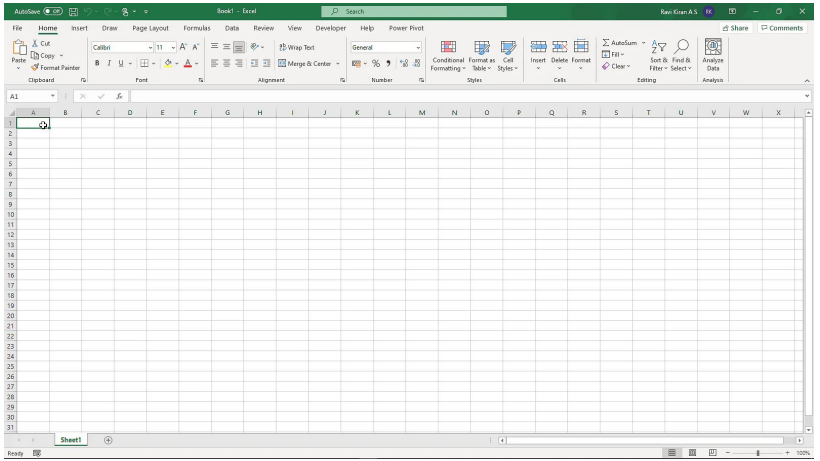
The Hello World Programme

```cpp
#include <iostream>  // Including iostream that lets you use IO to/from the console amongst other things
using namespace std; // Indicating that we're using the standard namespace

int main()  // Essential to every programme
{
    string myPhrase = "Helloworld";  // Declaring and defining string in main() scope
    cout << myPhrase << endl;        // Output the string to the console.

}
```

- #include is required, whether for using STL libraries or your own.
- We use <> for standard library files and “file.h” for your own files - the compiler looks in the local directory
- using namespace std - one sentence to prevent hours of frustration.
- main() - This is what the compiler looks for. If you don’t include this, nothing will work. Sometimes, nothing will work regardless.
- cout << is for output, cin >> is for input. endl just makes things look clean.

Variables and Data Types

- Just like in Arduino in the last semester, C++ has a bunch of data types.
- The important ones to know are the same:
  - **Int** - For any integers. There are some more types of ints that support different ranges but the ones you need to use mostly are int, unsigned ints and longs.
  - **Float** - The decimal point data type. Double gives double the range of floating point values.
  - **Bool** - Binary value. Generally, ‘true’ and ‘false’ are used in C++ which correspond to 1 and 0.
  - **Char** - Stores a single character. eg: “y” or “2” or “%”
  - **String** - Like a char but stores many characters + Supports multiple operations that chars don’t.



Modifiers and typedef

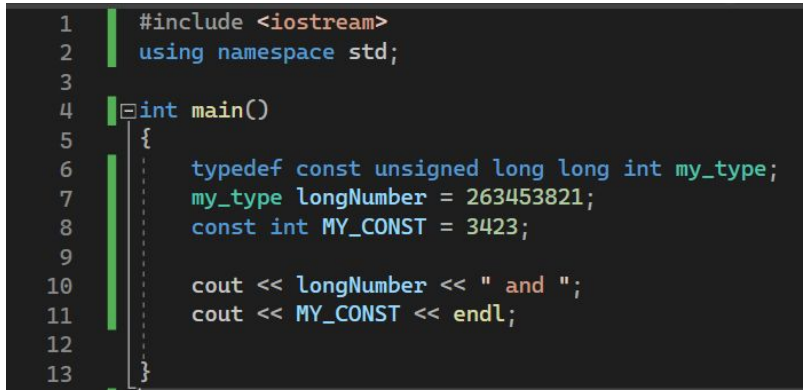
- In C++, you can ‘modify’ the data type by using keywords such as ‘unsigned’, ‘signed’, ‘long’ and ‘short’.
- Why though? Storage optimisation! Short takes less space, long takes more but allows for a larger range. Unsigned pushes all the range to +ve. Signed is the default.
- You can also define your own data types by using typedef - incase you don’t want to type ‘unsigned long long int’ over and over again like in the example.

Type	Definition	Control Character	Limits
int	Integer		-2147483648 to 2147483647
short	Short Integer		-32768 to 32767
long	Long Integer	l or L	-2147483648 to 2147483647
float	Floating Decimal Number	f or F	1.17549e-038 to 3.40282e+038
double	Double Decimal Number		2.22507e-308 to 1.79769e+308
long double	Long Decimal Number		2.22507e-308 to 1.79769e+308
char	Character		-128 to 127
unsigned int	Unsigned Integer		0 to 4294967295
unsigned short	Unsigned Short Integer		0 to 65535
unsigned long	Unsigned Long Integer		0 to 4294967295
unsigned char	Unsigned Character		0 to 255
bool	True or False		True = 1 and False = 0

```cpp
#include <iostream>
using namespace std;

int main()
{
    typedef unsigned long long int my_type;
    my_type longNumber = 263453821;

    cout << longNumber << endl;

}
```

Qualifiers

- Apart from modifiers, you can also use qualifiers as a prefix - These include const, restrict, volatile, etc.
- Const makes the data constant. They are conventionally named using capital letters and underscores eg: MY_CONST.
- You can also use qualifiers with typedef like in the example.
- If you’re using modifiers and qualifiers at the same time, conventionally, qualifiers go first although both are legal.



```
#include <iostream>
using namespace std;

int main()
{
    typedef const unsigned long long int my_type;
    my_type longNumber = 263453821;
    const int MY_CONST = 3423;

    cout << longNumber << " and ";
    cout << MY_CONST << endl;

}
```

Naming Conventions for Variables

You'll see all kinds of names in other people's code when you inevitably need to google something to make your own code work or when you use other people's libraries. Some of these conventions are:

- Most variables will use camelCase - myInt, anotherInt, aStringNow, etc.
- A lot of people may use 'g_' for declaring global variables. eg: g_LivesLeft.
- Constants will be screaming case with underscores. eg: MY_CONSTANT.
- If a name starts with a capital letter, it is most likely a class. Eg: ZooAnimal.
- But objects will be in camelCase.
- Most nouns will be variables and verbs will be functions!
- Most super local-scope variables will be single letters eg: i, j, x, y.

Declaring and Defining Variables.

```cpp
#include <iostream>
using namespace std;


int main()
{
    int myVar1 = 2;
    int myVar2(6);
    int myVar3{9};

    // auto myVar4;
    auto myVar5 = 16;

    decltype (myVar1) myVar6 = 5;

    cout << myVar1 << myVar2 << myVar3 << myVar5 << myVar6 << endl;

}
```

Operations on Ints

Most of the operations from the other languages are the same (i.e. +,-,*,/, ++, +=, etc.) But here's three fun things to note:

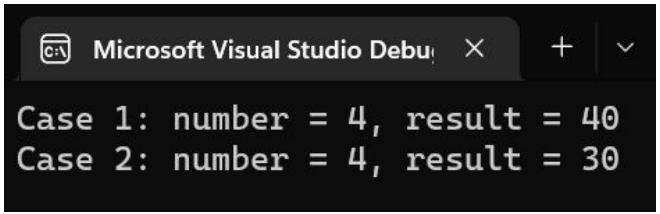
1. **Order of Operations**

- Is like BODMAS from elementary school. Brackets then Division, Multiplication, Addition and Subtraction. At equal priority, they are performed left to right.
- So 'x + 3 * 10' will not be the same as '(x + 3) * 10' or 'x + (3 * 10)'
- I didn't mean to treat everyone like an elementary schooler here, sorry, but I've made this mistake as a 20-something last week, so it doesn't hurt to reiterate. I'm embarrassed and I'm projecting.

2. **Prefix/Suffix Increment/Decrement**

- Here's two of the many ways of incrementing the same variable by one. Either x++ or ++x.
- If you perform ++x, it increments the variable before the larger expression is resolved. So in this example, in case 1, result will be equal to 40.
- But x++ performs the increment after. So even though number will be equal to 4 in both cases when we output it, result will be equal to 30 in case 2.
- So be careful with ++ in larger expression. Stick to x = (x+1) * 10…. Unless you know exactly what you're doing

```cpp
#include <iostream>
using namespace std;

int main()
{
    int number = 3;
    int result = ++number * 10;
    cout << "Case 1: " << "number = " << number << ", result = " << result << endl;

    number = 3;
    result = number++ * 10;
    cout << "Case 2: " << "number = " << number << ", result = " << result << endl;

}
```



3. Using Modulo (%) and rand() together.

- Here's a neat trick with the modulo operator!
- The code generates a random number and then uses '%6' to divide each by 6 and give the remainder within a range of 0 to 5.
- The +min at the end adjusts the range to 0-6.
- rand() is part of the STL. It's found in the cstdlib that's included at the top.
- You need to seed it!

```cpp
#include <iostream>
#include <cstdlib>

int rollDie()
{
    int roll;
    int min = 1;
    int max = 6;

    roll = rand() % (max - min + 1) + min;

    return roll;
}

int main()
{
    srand(time(0));
    for (int i = 0; i < 10; i++)
    {
        std::cout << rollDie() << std::endl;
    }
}
```

Maths

- C++ lets you perform max(x,y) and min(x,y) by default.
- Other maths functions need the library <cmath>.
- You can find all other cmath functions here.

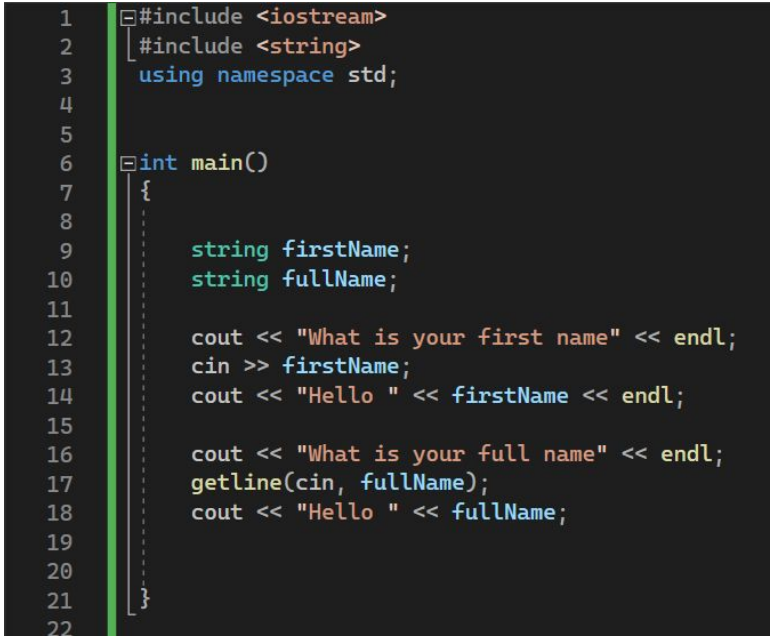
```cpp
#include <iostream>
#include <cmath>

using namespace std;


int main()
{

    int x = 54356;
    int y = 23523;

    cout << max(x, y) << endl;
    cout << min(2, 5) << endl;
    cout << sqrt(64) << endl;
    cout << round(2.6) << endl;
    cout << log(2) << endl;

}
```

Operations on Strings

```cpp
#include <iostream>
#include <string>
using namespace std;


int main()
{

    string a = "Hi! ";      // Three ways to declare strings
    string b("I am ");
    string c{ "Advait" };
    string exclaim(3, '!');  // Creating string with the same character repeated

    cout << a + b + c + exclaim << endl;    // Concatenation
    cout << a.append(c) << endl;            // Appending
    cout << c.find("v") << endl;            // Finding position of a character
    c.erase(c.find("v"), 3);                // Finding v and erasing the 3 letters after that.
    cout << c << endl;

    c = "Advait";                           // Modifying string

    string d = a + b + c;
    cout << d.length() << endl;   // Calculating the length of the string
    cout << d[10] << endl;        // Accessing individual characters in strings


    string l1 = "Line 1's String";
    string l2 = "Line 2";
    string l3 = "\nLine 2";       // Using \n for next line. Use \t for tab

    cout << l1 << l2 << endl;
    cout << l1 << l3 << endl;


}
```

User Input Strings

- You can ask the user to input a string using cin. Remember when using cin, the >> point the opposite way.
- But cin only takes one word and terminates at any whitespace in the input.
- If you need more than that, use getline(cin, variableName);



```
#include <iostream>
#include <string>
using namespace std;


int main()
{

    string firstName;
    string fullName;

    cout << "What is your first name" << endl;
    cin >> firstName;
    cout << "Hello " << firstName << endl;

    cout << "What is your full name" << endl;
    getline(cin, fullName);
    cout << "Hello " << fullName;



}
```

enums

- Enums are a special type of data structure that allows one to create an array of words that are each associated with a number.
- This is used a lot when you want to create a machine state programme as switch statements can only take numbers.
- You can declare individual elements in an enum with or without a value.
- It will start from 0 as a default and if a value in the middle is not indicated it will default to the last element's value + 1

```cpp
#include <iostream>
#include <string>
using namespace std;


int main()
{

    enum Difficulty { easy, medium, hard, sadism };  // Default values starting from 0
    cout << easy << " " << medium << " " << hard << " " << sadism << " " << endl;


    enum PlayerSkillLevel {     // Setting Values
        beginner = 0,
        novice = 4,
        expert = 8,
        god = 12
    };
    cout << beginner << " " << novice << " " << expert << " " << god << " " << endl;


    enum EnemyLevel { easiest, mediocre = 13, tougher, toughest = 30 };
    cout << easiest << " " << mediocre << " " << tougher << " " << toughest << " " << endl;
    //How values are auto assigned

}
```