Computing Portfolio 1

Basics

Hello World

A C++ file can be created in (cmd) terminal using `code filename.cpp`, which creates and opens the C++ file in Visual Studio Code. For the hello world program, we simply want to output the string `"Hello World"`. As this requires an output, we must include the `iostream` library as it defines the standard input and output stream objects; this allows for inputs and outputs in the program. Libraries can be included in a script using the `#include<library_to_include>` syntax.

C++ programs always include a `main()` function which returns an integer. If the program runs successfully, then the program will return a 0 using `return 0;`, , noting the inclusion of `;` at the end of the line to tell C++ where the code lines end. If the program runs unsuccessfully, it will return a different integer; this integer can be related to a thrown error to help debug the program. When defining functions in C++, we state the type of variable to be returned, in our case it needs to be `int`. We define the scope of the function using curly brackets: `{}`.

Next, we want to output our string. To do this, we need to use the standard output stream, `cout`, which is a part of the standard library (`std`) in C++. This means that we need to specify that we are getting `cout` from the standard library with `std::cout`. Next, we need to insert our string into the output stream. We do this using the insertion operator, `<<`. This means that a string can be output using `std::cout << "string_to_output";`. After our string is output, we also want to create a new line by inserting `std::endl;` into the output stream.

Putting this all together, our hello world program looks like this.

```
// Example: Hello World

#include <iostream>

int main()
{
    std::cout << "Hello World" << std::endl;

    return 0;
}
```

Variable Types

When declaring variables in C++, the variable type must be stated (for example to declare an integer variable: `int var_name`). Some common variable examples are:

- **int**: Integer
- **float**: Floating point number with a precision of 6-7 decimal points
- **double**: Floating point number with a precision 15-16 decimal points

- **bool**: Binary/logical variable (true/false, or 1/0)
- **char**: Single characters (surrounded in quotes `''`)
- **std::string**: String (requires `#include <string>`).

In addition to these, a variable can be declared using `auto` in place of a variable type which tells C++ to work out which type the variable should be. A variable can be initialised using `=` or by `{}`.

```
int var1 = 10;
int var2{10};
```

Conditions and `for` Loops

C++ has `if` and `else` statements and `for` loops similar to other popular languages; it's syntax has the condition in brackets after the conditional and the scope of the condition is given in curly brackets, `{}`.

```
// Output the 5 times table up to 30

#include <iostream>

int main()
{
    // Output all but last value
    for (int i{1}; i < 30; i++) {
        if (i % 5 == 0) {
            std::cout << i << ",";
        }
    }
    // Add the last value and end the line
    std::cout << 30 << std::endl;
}

// OUTPUT: 5,10,15,20,25,30
```

Functions

Similar to declaring variables, when declaring a function, you must declare the type that the function will return. Functions must be declared before (above) the `main()`, otherwise the compiler will say that the function has not been declared.

```
// Example: Function definitions

int sum( int a, int b )
{
    int c = a + b;
    return c;
}

// A function of type void will not return anything
void print_hello()
{
    std::cout << "Hello World" << std::endl;
}
```

Header files

Header files are useful for code organisation for larger projects. They allow for declarations to be made in a separate script to keep the main script easy to read and understand. Header files have the file handle `.h` and have header guards: `#ifndef _HEADER_NAME_H`, `#define _HEADER_NAME_H` at the top of the script and `#endif` at the bottom. Functions, variables and classes that are declared in this script can then be called another script by including the header file at the top, e.g. `#include "header_name.h"`.

```
// Header file containing timestable function

#ifndef _TIMESTABLE_H
#define _TIMESTABLE_H

#include<iostream>

void timestable(int times, int max_num)
{
    int initial_value{times};
    int current_num{initial_value};
    while (current_num < max_num) {
        std::cout << current_num << ", ";
        current_num += times;
    }
    std::cout << std::endl;
}
#endif
```

```
// A different script that uses the timestable function defined in the header file.

#include <iostream>
#include "timestable.h"

int main()
{
    std::cout << "Five times table" << std::endl;
    timestable(5,100);

    std::cout << "Twelve times table" << std::endl;
    timestable(12, 100);

    return 0;
}

// OUTPUT:
//   Five times table
//   5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90, 95,
//   Twelve times table
//   12, 24, 36, 48, 60, 72, 84, 96,
```

Numerical Type Conversion

In C++, you cannot re-declare variables that already exist (in the local scope). Variables can be assigned to a new variable that has a different type, but there are forms of 'safe' conversion and 'unsafe'. A 'safe'

conversion may be going from an `int` variable to `float` as the float is more precise than `int` and therefore no data will be lost. However, an 'unsafe' conversion would be converting from a `double` to a float as a `double` holds more precision than a float and therefore would have to be truncated, leading to a loss of data.

Variable Scope

As stated previously, curly brackets (`{}`) define scope in C++. Unlike other interpreted languages, variables in C++ exist in a specific scope. If a variable is declared between curly brackets, then that variable exists only until the end of the close bracket; until the end of the variable's scope.

```cpp
// Example: Scope

#include <oistream>

int main()
{
  // Declare an iteration variable within a for loop scope
  for (int i{0}; i < 3; i++) {
    std::cout << "iteration: " << i << std::endl;
  }
  // This variable will not exist outside the for loop scope
  std::cout << i << std::endl;

  return 0;
}

// OUTPUT:
//   error: 'i' was not declared in this scope
```

```cpp
#include <iostream>

int main()
{
  // Declare an iteration variable outside the for loop scope
    int iter{0};
    for (int i{0}; i < 3; i++) {
        std::cout << "iteration: " << iter << std::endl;
        iter++;
    }
    // This variable will not exist outside the for loop scope
    std::cout << "max iteration: " << iter << std::endl;

    return 0;
}

// OUTPUT
//   iteration: 1
//   iteration: 2
//   iteration: 3
//   max iteration: 3
```