1 C++: Lecture 1 on 17 May 2020

1.1 Preprocessors

What is a preprocessor? These are lines of code that will run before the compilation occurs. These lines will not be present when we run the actual program, it just is used to simplify code, such as by including libraries.

All preprocessor statements are called "directives" and they start by using a #, for example.

```
#include <iostream> //input output stream
#include <string>//examples
#include <vector>
#include <fstream>
#include <math.h>
```

1.2 Comments

Comments are used to document and markup code for later usage, these comments are never evaluated.

```
//this is a comment
/*
this is a comment
*/

/*comments are very useful. by not being considreed by the compiler, they
    ↪ can be usued to record notes, wirte down thought processes, and
    ↪ leave messges for collaborations*/
```

1.3 Programming Fundamentals

Every single C++ program has a few charecteristics. Every program will have a single *int main*() function which is the **starting point** of execution. This will typically always return 0 but not always required.

```
int main() {
    // code goes here
    return 0;
}
```

1.4 Outputting

The most essential part of programming is printing to a terminal, you can do it in C++ by writing into a stream using the following syntax.

```
std::cout << "Hello world\n";
```

In this case, *cout* is a standard output stream object, and pushes content to the console output. The usage of double arrows such as << are insertion operators.

cout can be used to print anything to the console. The words must be wrapped in quotations (") and end with a semicolon. You can use a newline by simply using $\backslash n$ or by additionally inserting $std : endl$.

```
std::cout << "Hello world\n";
std::cout << "Hello " << "World" << "\n";
std::cout << "My name is Rob\n";
std::cout << "My name is Rob" << std::endl;

std::cout << "Hello\n"
           "Hello\n";

std::cout << "Hello\nHello\n";
```

1.4.1 Task: Simulate Conversation

Utilize *cout* to simulate a converstaion between two people, take about 5 minutes to solve this problem. You can make the conversation messages anything you would like.

On top of that, *cout* can be used for more complex situations, such as performing mathematical operations.

```
using namespace std;

cout << 8 + 9 << "\n";
cout << 67*9124 << "\n";
```

Note: We are using *using namespace std*; in this code so that we do not have to say *std* :: *cout*, rather we can assume using *std* with the using namespace statement. This means we can just write *cout* instead.

Something of importance, if we were to wrap those with quotes, it would literally print out "8 + 9" rather than 17, as intended.

We can also incorporate strings and computation within the same *cout* statement, for example:

```
cout << "8 + 9 is: " << 8+9 << endl; // 8 + 9 is: 17
```

1.5 Operators

In C++, there are a few types of operations.

- Addition using +
- Subtraction using −
- Multiplication using ∗
- Division using \
- Modulus (Remainder) using %

1.5.1 Task: Series of Operations

Write some code that prints out some initial number (of your choice) by 7, divide it by 2, subtract 7, and finally take the remainder when the number is divided by 10. You will be using all the operators as listed above.

Take about 5 minutes to solve this.

1.6 Variables

Variables are imporatnt tools in programming and allow for ease of acces etc. For now we will only focus on integers.

Integers - any number positive or negative that doesnt have a decimal component
In order to declare variables, you can do it like so:

```
int a;
int b;
int c;

// Alternatively...
int a, b, c;
```

Declaring variables is one thing, however, we can also define the value of these variables as well, when we declare, like so:

```
# Defining variables. A variable would be useless without having a value
    ↪ assigned to it.
a = 7;
b = 8;
c = 10;

# You can also declare and define varaibles at the same time.
int a = 5;

#Redefining is also useful. For example:
int a = 5;
a = 7;
```

Here's a more complex example utilizing variables and mathematical operations through time, run this code and see what you get.

```
int a = 22;

a = a + 1;
cout << a << endl;
a = a - 1;
cout << a << endl;
a = a/2; //truncated "round down" b/c int
cout << a << endl;
a = a * 2;
cout << a << endl;
a = a % 10;
cout << a << endl;
a += 10;
cout << a << endl;
a -= 10;
cout << a << endl;
a *= 10;
cout << a << endl;
a /= 10;
cout << a << endl;
//post increment/decrement
a++;
cout << a << endl;
a--;
cout << a << endl;
//pre increment/decrement
++a;
cout << a << endl;
--a;
cout << a << endl;
```

1.6.1 Task: Operations on Variables

In this task, we will be having you define an integer and take it through a series of operations.

- Increase this number 7
- Then, multiply this new value by 5
- Divide the new value by 10
- Finally, multiply this value by itself

At the end, print the number out to the terminal. You can also print throughout the operations to see how it transforms. Take about 7 minutes to solve this question.

1.7 Input

Rather than just ouputting, we can also use code to take an input from the user and perform certain operations with it. This structure is extremely similar to *cout*, but we will be using *cin*.

```
int MY_INT;
MY_INT = 5;
cin >> MY_INT;
cout << MY_INT << endl;

//another example would be
int b,c,x,y,z;
cin >> a >> b >> c >> x >> y >> z;
```

Remember, integers are used for whole numbers. If you wanted to work with inputting strings, you would have to specify the *string* type when initializing the variable, like so:

```
string My_Name;
My_Name = "Bob\n";
cout << My_Name;

//now we can use cin with strings
string name;
cin >> name;
cout << "Your name is: " << name << endl;
```

1.7.1 Task: Age Calculator

Using *cin* and *cout* statements, take the user's name and age as inputs (make sure to use the proper datatypes for this) and output their name and their age after 18 years have passed. Take at most 7 minutes to solve this problem.