Introduction to C++

What is C++?

C++ is a High Level, General Purpose programming language, created by Sir Bjarne Stroustrup. It was originally created as an extension of C Programming Language, to include OOP style to C which Sir Bjarne Stroustrup, discovered when reading about a language which was gaining popularity back then, called Smalltalk. As such C++ was originally called C with classes.

Originally C with classes would just transpile to C and then compile to machine code and used to function just as a superset of C. However, later on, when Sir Stroustrup realised the potential the language had, he decided to create C with classes into C++ which was more than just a superset of C, though it was still 99% compatible with it.

C++ program structure

Every C++ program has a similar structure. At the top of the file you have preprocessor directives, header files, namespace declarations, etc. After that you have declarations of variables, functions, classes etc. After that you have the main function, which is the entry point of a C++ program. Then you have the definitions.

The main function of a program has the following structure:

```
int main() {
  // body of main
  return 0;
}
```

Following is the hello world program in C++:

```
#include <iostream>

using std::cout;
using std::endl;

int main() {
  cout << "Hello, World!" << endl;
  return 0;
}
```

Let's analyze the program

Starting from the top, we have `#include <iostream>` as the first line. In C++ whenever you see a `#` it means a preprocessor directive. A preprocessor directive is code which is executed before compilation of the code even begins.

In this program we're using the `include` preprocessor directive. The `include` preprocessor directive makes the compiler copy the contents of the file passed to it (in this case the `iostream` standard header file) in place of the directive itself.

Looking further we see that we write `iostream` in angular brackets (<>). The angular brackets indicate that the file provided is in the included path.

`iostream` in this case is a standard C++ header file which ships with every C++ compiler.

If you want to use a header file whose path is not in the included path for your compiler, you can use the double quotes around the file path to include it. OR you can use the -I flag with your compiler to pass the file path to the compiler.

Now, following the preprocessor directive, we see the `using` directive. `using` is a C++ command which allow you to use functions/classes etc from a foreign namespace. A namespace is a collection of classes, functions, variables which are related to each other. In this program we're using `cout`, `endl` and `string` from the `std` (standard) namespace.

Following the `using` we have our main function. The main function starts as `int main()`

`int` in the name of the main function indicates that the function returns an integer. The concept of returning is a slightly more detailed concept but for the main function, if the main function reaches the end with no issues, it returns 0, which means the program ran successfully. If something happens in the program which makes it fail, an error code is instead returned, depending on the error that occurred.

After that we have the `cout << "Hello, World!" << endl;` statement.

`cout` is a variable in `std` namespace which stores the information required to print to the standard output console. The `<<` is the overloaded left shift operator whenever you have something like `cout << variable`, (so long as the logic for `<<` operator is defined for the particular type) it will be printed to the console.

`endl` is another variable from `std` namespace which basically ends the line and flushes the buffer of `cout` (more on that later).

At the end of the main function we have the `return 0;` statement which tells the compiler that if the program reaches this point, it can exit successfully, with no errors.

Object Oriented Programming

Object Oriented Programming is a style of programming that works by coupling data and behaviour. It uses objects to do so, hence the name. To create objects we require schema which has their data and expected behaviour. This schema is called a class.

A class is like a blueprint for creating an object. It mentions all the variables(state) and methods(behaviour) that the object will have. Let's take an example class Contact. An actual Contact would have a Person's name, their phone number, their email and their residential address. A Contact class would mention these as object / instance variables along with there types.

An object is a concrete instance of class. In the above example, an object would be an actual concrete contact, for example contact of John Doe, whose phone number is +1 111 1111 111 and email johndoe@domain.extension who lives at 123 Main Street.

In C++ to create a class we use the keyword `class` followed by the class name. The syntax for creating the contact class would be as follows:

```
#include <iostream>
#include <string>

using std::cout;
using std::cin;
using std::endl;
using std::string;

class contact {
private:
  string name;
  string contact_number;
  string email_id;
  string residential_address;

public:
  contact(string const &name, string const &contact_number, string const &email_id,
          string const &residential_address) {
    this->name = name;
    this->contact_number = contact_number;
    this->email_id = email_id;
    this->residential_address = residential_address;
  }

  string get_name() { return name; }
  string get_contact_number() { return contact_number; }
  string get_email_id() { return email_id; }
  string get_residential_address() { return residential_address; }
  void call(contact const &recipient) {
    // calls the recipient
  }
};

int main() {
```

```
  contact john_doe("John Doe", "+1 111 1111 111", "johndoe@domain.extension",
                   "123 Main Street");
  cout << john_doe.get_name() << endl;
  cout << john_doe.get_contact_number() << endl;
  cout << john_doe.get_email_id() << endl;
  cout << john_doe.get_residential_address() << endl;
}
```