LECTURE ONE

C++ Introduction

C++ history

C++ programming language was developed in 1980 by Bjarne Stroustrup at bell laboratories of AT&T (American Telephone & Telegraph), located in U.S.A.

Bjarne Stroustrup is known as the **founder of C++ language.**

It was develop for adding a feature of **OOP (Object Oriented Programming)** in C without significantly changing the C component.

C++ programming is "relative" (called a superset) of C, it means any valid C program is also a valid C++ program.

Let's see the programming languages that were developed before C++ language.

Language	Year	Developed By
Algol	1960	International Group
BCPL	1967	Martin Richard
B	1970	Ken Thompson
Traditional C	1972	Dennis Ritchie
K & R C	1978	Kernighan & Dennis Ritchie
C++	1980	Bjarne Stroustrup

What is C++?

C++ is a cross-platform language that can be used to create high-performance applications.

C++ was developed by Bjarne Stroustrup, as an extension to the C language.

C++ gives programmers a high level of control over system resources and memory.

The language was updated 3 major times in 2011, 2014, and 2017 to C++11, C++14, and C++17.

Why Use C++

C++ is one of the world's most popular programming languages.

C++ can be found in today's operating systems, Graphical User Interfaces, and embedded systems.

C++ is an object-oriented programming language which gives a clear structure to programs and allows code to be reused, lowering development costs.

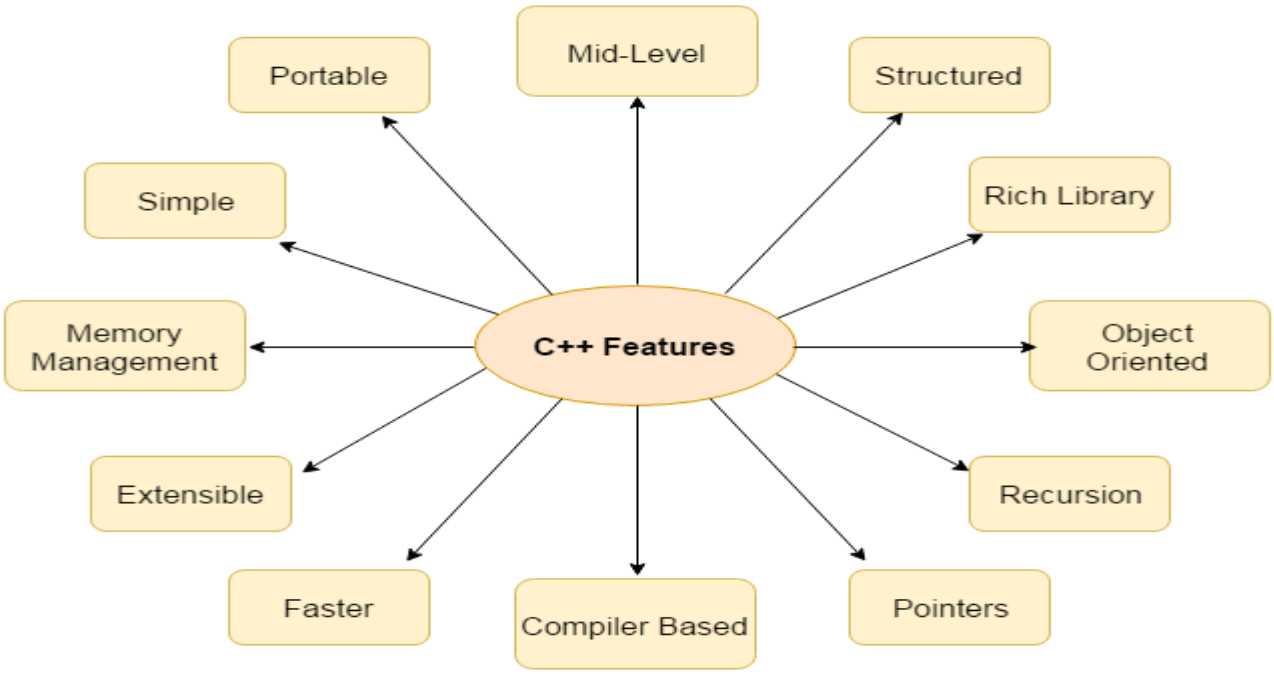
C++ is portable and can be used to develop applications that can be adapted to multiple platforms.

C++ is fun and easy to learn!

As C++ is close to C# and Java, it makes it easy for programmers to switch to C++ or vice versa

C++ Features

C++ is object oriented programming language. It provides a lot of **features** that are given below.



1. Simple
2. Machine Independent or Portable
3. Mid-level programming language
4. Structured programming language
5. Rich Library
6. Memory Management
7. Fast Speed
8. Pointers
9. Recursion
10. Extensible
11. Object Oriented
12. Compiler based

1) Simple

C++ is a simple language in the sense that it provides structured approach (to break the problem into parts), rich set of library functions, data types etc.

2) Machine Independent or Portable

Unlike assembly language, c programs can be executed in many machines with little bit or no change. But it is not platform-independent.

3) Mid-level programming language

C++ is also used to do low level programming. It is used to develop system applications such as kernel, driver etc. It also supports the feature of high level language. That is why it is known as mid-level language.

4) Structured programming language

C++ is a structured programming language in the sense that we can break the program into parts using functions. So, it is easy to understand and modify.

5) Rich Library

C++ provides a lot of inbuilt functions that makes the development fast.

6) Memory Management

It supports the feature of dynamic memory allocation. In C++ language, we can free the allocated memory at any time by calling the free() function.

7) Speed

The compilation and execution time of C++ language is fast.

8) Pointer

C++ provides the feature of pointers. We can directly interact with the memory by using the pointers. We can use pointers for memory, structures, functions, array etc.

9) Recursion

In C++, we can call the function within the function. It provides code reusability for every function.

10) Extensible

C++ language is extensible because it can easily adopt new features.

11) Object Oriented

C++ is object oriented programming language. OOPs makes development and maintenance easier where as in Procedure-oriented programming language it is not easy to manage if code grows as project size grows.

12) Compiler based

C++ is a compiler based programming language, it means without compilation no C++ program can be executed. First we need to compile our program using compiler and then we can execute our program.

Applications of C++ Programming

As mentioned before, C++ is one of the most widely used programming languages. It has it's presence in almost every area of software development.

- **Application Software Development** - C++ programming has been used in developing almost all the major Operating Systems like Windows, Mac OSX and Linux. Apart from the operating systems, the core part of many browsers like Mozilla Firefox and Chrome have been written using C++. C++ also has been used in developing the most popular database system called MySQL.
- **Programming Languages Development** - C++ has been used extensively in developing new programming languages like C#, Java, JavaScript, Perl, UNIX's C Shell, PHP and Python, and Verilog etc.
- **Computation Programming** - C++ is the best friends of scientists because of fast speed and computational efficiencies.
- **Games Development** - C++ is extremely fast which allows programmers to do procedural programming for CPU intensive functions and provides greater control over hardware, because of which it has been widely used in development of gaming engines.
- **Embedded System** - C++ is being heavily used in developing Medical and Engineering Applications like softwares for MRI machines, high-end CAD/CAM systems etc.

C++ supports the object-oriented programming, the four major pillar of object-oriented programming (OOPs) used in C++ are:

1. Inheritance

2. Polymorphism
3. Encapsulation
4. Abstraction

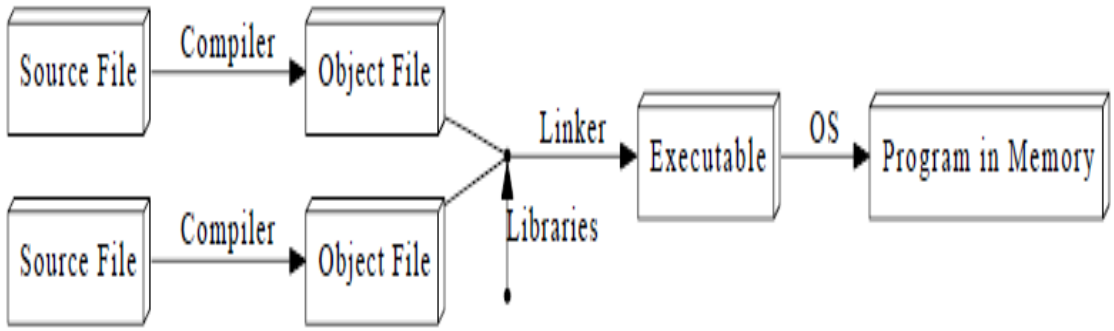
C++ Standard Libraries

Standard C++ programming is divided into three important parts:

- The **core library** includes the data types, variables and literals, etc.
- The **standard library** includes the set of functions manipulating strings, files, etc.
- The **Standard Template Library (STL)** includes the set of methods manipulating a data structure.

The Compilation Process

A program goes from text files (or source files) to processor instructions as follows:



The following **three steps** are required to create and translate a C++ program:

1. First, a text editor is used to save the C++ program in a text file. In other words, the *source code is saved to a source file. In larger projects the programmer will normally* use *modular programming. This means that the source code will be stored in* several source files that are edited and translated separately.

2. The source file is put through a *compiler for translation. If everything works as* planned, an object file made up of *machine code is created. The object file is also* referred to as a ***module***.

3. Finally, the *linker combines the object file with other modules to form an executable file. These further modules contain functions from standard libraries or* parts of the program that have been compiled previously.

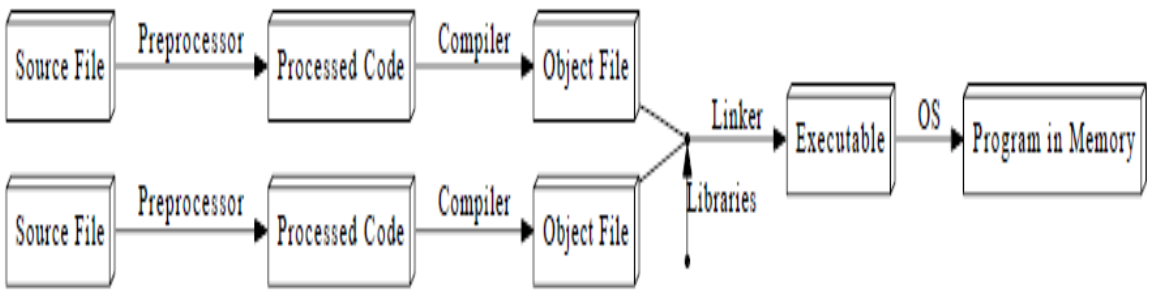
4. The compiler and linker are just regular programs. The step in the compilation process in which the compiler reads the file is called **parsing**.

5. If the source file contains just one *syntax error, the compiler will report an error. Additional error* messages may be shown if the compiler attempts to continue despite having found an error. So when you are troubleshooting a program, be sure to start with the first error shown.

6. In addition to error messages, the compiler will also issue *warnings. A warning does* not indicate a syntax error but merely draws your attention to a possible error in the program's logic, such as the use of a non-initialized variable.

7. In C++, all these steps are performed ahead of time, before you start running a program. In some languages, they are done during the execution process, which takes time. This is one of the reasons C++ code runs far faster than code in many more recent languages.

8. C++ actually adds an extra step to the compilation process: the code is run through a preprocessor, which applies some modifications to the source code, before being fed to the compiler. Thus, the modified diagram is:



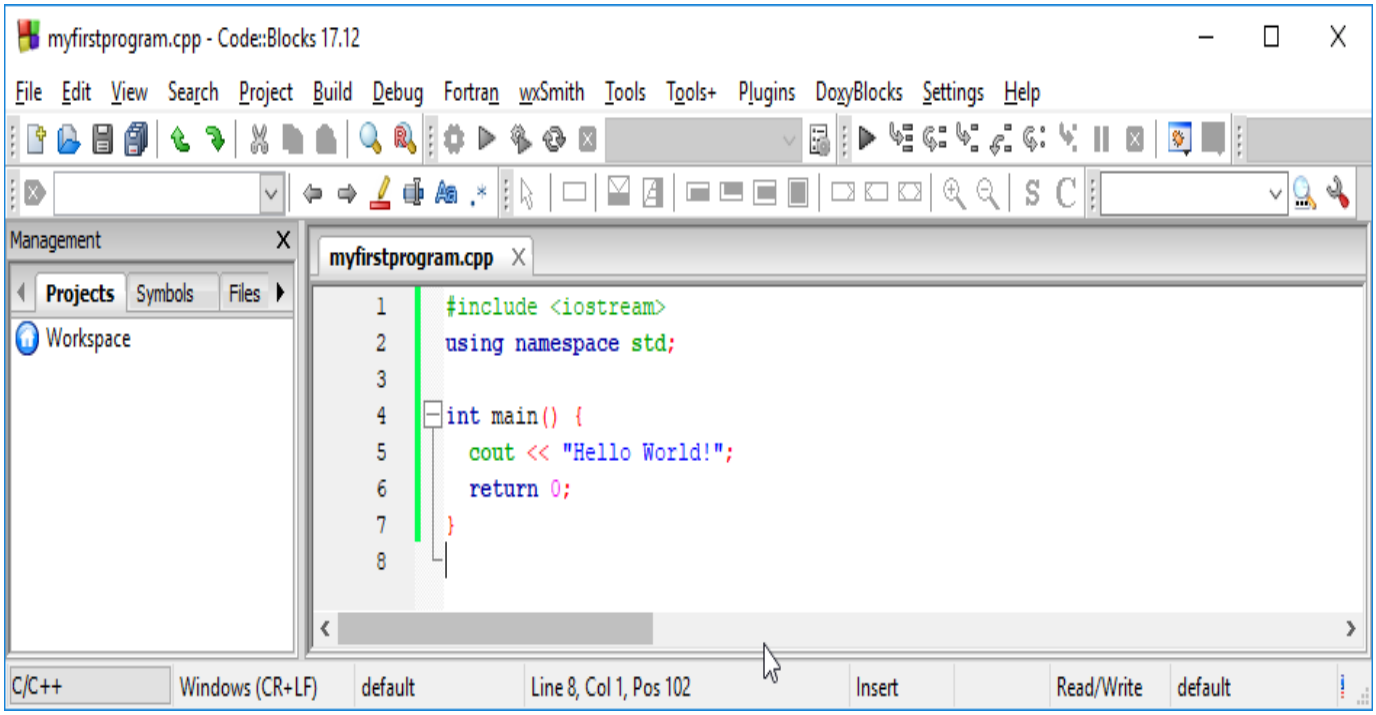
C++ Program

Let's create our first C++ file.

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

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

In Codeblocks, it should look like this:



Then, go to **Build > Build and Run** to run (execute) the program. The result will look something to this:

```
Hello World!
Process returned 0 (0x0) execution time : 0.011 s
Press any key to continue.
```

Example explained

Line 1: #include <iostream> is a **header file library** that lets us work with input and output objects, such as cout (used in line 5). Header files add functionality to C++ programs.

Line 2: using namespace std means that we can use names for objects and variables from the standard library.

Don't worry if you don't understand how #include <iostream> and using namespace std works. Just think of it as something that (almost) always appears in your program.

Line 3: A blank line. C++ ignores white space.

Line 4: Another thing that always appear in a C++ program, is int main(). This is called a **function**. Any code inside its curly brackets {} will be executed.

Line 5: cout (pronounced "see-out") is an **object** used together with the *insertion operator* (<<) to output/print text. In our example it will output "Hello World".

Note: Every C++ statement ends with a semicolon ;.

Note: The body of int main() could also been written as:
int main () { cout << "Hello World! "; return 0; }

Remember: The compiler ignores white spaces. However, multiple lines makes the code more readable.

Line 6: return 0 ends the main function.

Line 7: Do not forget to add the closing curly bracket } to actually end the main function.

Omitting Namespace

You might see some C++ programs that runs without the standard namespace library. The using namespace std line can be omitted and replaced with the std keyword, followed by the :: operator for some objects:

Example

```
#include <iostream>

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

C++ Output (Print Text)

The cout object, together with the << operator, is used to output values/print text:

Example

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

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

You can add as many cout objects as you want. However, note that it does not insert a new line at the end of the output:

Example

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

int main() {
  cout << "Hello World!";
  cout << "I am learning C++";
  return 0;
}
```

New Lines

To insert a new line, you can use the \n character:

Example

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

int main() {
  cout << "Hello World! \n";
  cout << "I am learning C++";
  return 0;
}
```

Tip: Two \n characters after each other will create a blank line:

Example

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

int main() {
  cout << "Hello World! \n\n";
  cout << "I am learning C++";
  return 0;
}
```

Another way to insert a new line, is with the endl manipulator:

Example

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

int main() {
  cout << "Hello World!" << endl;
  cout << "I am learning C++";
```

```
  return 0;
}
```

Both \n and endl are used to break lines. However, \n is used more often and is the preferred way.

C++ Comments

Comments can be used to explain C++ code, and to make it more readable. It can also be used to prevent execution when testing alternative code. Comments can be singled-lined or multi-lined.

Single-line comments start with two forward slashes (//).

Any text between // and the end of the line is ignored by the compiler (will not be executed).

This example uses a single-line comment before a line of code:

Example

```
// This is a comment
cout << "Hello World!";
```

This example uses a single-line comment at the end of a line of code:

Example

```
cout << "Hello World!"; // This is a comment
```

C++ Multi-line Comments

Multi-line comments start with /* and ends with */.

Any text between /* and */ will be ignored by the compiler:

Example

```
/* The code below will print the words Hello World!
to the screen, and it is amazing */
cout << "Hello World!";
```

Single or multi-line comments?

It is up to you which you want to use. Normally, we use // for short comments, and /* */ for longer.