## Early History of C++

## C++ History Review

* C with Classes, created by Bjarne Stroustrup in 1980s
* C front compiler, which was similar to a traditional preprocessor, converted C++ to C and use C compiler for assembly
* Core language:
* Object Oriented: class, derived class, abstract classes, virtual function
* Generic programming: template
* Operator overloading, references, name spaces, exception handling
* Standard library: Original STL implementation by Stepanov and Lee. 1994
* Containers, algorithms, iterators and functions6

## C++ 98

* The first international standard for C++ ISO
* Core language: cast operator, template instantiation, member templates
* Standard library: STL, auto\_ptr, I/O stream (std::cout, std::cin)

## C++ 03

* Minor revision of the C++98 standard, primarily bug fix release to address core language as well as library defects.

## C++ 11

* Modern C++ standard, bringing many major extensions and improvements
* Core language: auto, decltype, move constructors and move assignment operator, range-for, uniform initialization, lambda functions and expressions
* Standard library:
* Containers: <array>, <unordered\_map>, <unordered\_set>,
* Multi-threading: <thread>, <mutex>, <future>, <conditional\_variable>, <atomic>
* Time: <chrono>
* Functional: <functional>
* Smart pointers: <memory>, std::unique\_ptr, std::shared\_ptr cial Engineering (QF633) 5 / 50Evolution of C++

## C++ 14

* Minor revision after major C++11, mainly for minor improvements and defect fixes
* Core language: variable templates, generic lambdas, lambda init-capture
* Standard library: std::make\_unique

## C++ 17

* Major revision after major C++11
* Core language: constexpr if, if init-statement, structured bindings
* Standard library: Type erasure: std::any, std::variant
* Containers: std::string\_view +

## C++ 20

* Support from compilers are not yet completed
* The Big Four
* Concepts: write requirements for your templates to get improved error message instead of pages of cryptic error messages
* Range Library
* Co-routines: no C++20 concreate co-routines but only a framework for writing own co-routines Modules

## IDE Recommended

## C++ Basic concept, IDE, Compiler, Linker, CMake etc

* VS CODE for iOS, Visual Studio Code is a modern code editor.

1. Code editing.
2. Redefined.
3. Free. Built on open source.
4. Runs everywhere

More details to seup vs code on mac can be found in below link:

<https://code.visualstudio.com/docs/setup/mac>

* MS Visual Studio for Windows, Microsoft Visual Studio is a powerful tools for C++, including text editing, compiling and linking, debug, source code repository etc.

1. One stop solution and powerful.
2. Steep learning curve.
3. Recommend “community” version for study purpose

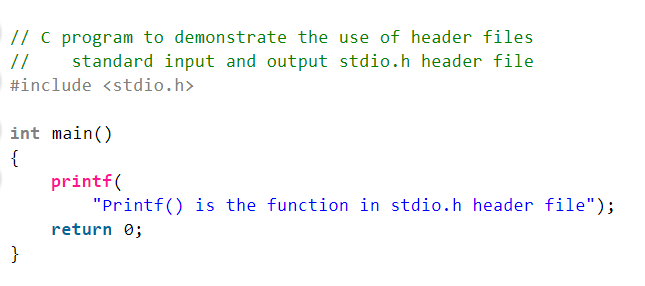
More details to seup visual studio can be found in below link:

<https://visualstudio.microsoft.com/downloads/>

## Structure of C++ Program: Components

The structure of a C++ program is made up of multiple source code files that cater to different components such as main function, member functions, class definition, headers/ standard headers, comments, variables, data types, namespaces, input/ output statements, etc.

Let’s see one example: suppose we have a file named **progl.cpp** as below



Header file

In C language, header files contain a set of predefined standard library functions. The .h is the extension of the header files in C and we request to use a header file in our program by including it with the C pre-processing directive “#include”.

C Header files offer the features like library functions, data types, macros, etc by importing them into the program with the help of a **pre-processor directive** “#include”. Syntax of Header Files in C: We can include header files in C by using one of the given two syntax whether it is a pre-defined or user-defined header file.

**#include <filename.h> // for files in system/default director**

**#include "filename.h" // for files in same directory as source file**

.cpp file

Files with CPP file extension are source code files for applications written in C++ programming language. A single C++ project may contain more than one CPP files as application source code.

Namespace Declarations

Namespace declarations are used in defining named scopes which help in organizing and grouping related code. It’s a way to group a set of identifiers under a common name, to avoid naming conflicts and improve readability. They are part of the actual C++ code and are processed by the C++ compiler during the compilation phase. Different users can create separate namespaces thus facilitating the use of similar names for entities. This further helps in avoiding possible compile-time errors which might arise due to identical-name conflicts.

## Functions in C++

Functions are a group of code statements also referred to as blocks of code/ source code. They are used to group a set of instructions that perform a specific task. Functions can be called from other parts of the program, making it easier to reuse code, implement modularity and improve the readability of the program. A coder can either use any of the many built-in functions already available or create specific user-defined functions, as they deem fit.

Main Function

The main function is the entry point or the startup function for every C++ program, i.e. this section defines the main() function of the program. This is where the program starts executing, hence this section is a mandatory part of the structure of the C++ program.

Besides the function declaration and program execution, the return type of the main function is also mentioned in this section. All in all, the main function coordinates the flow of the entire program by calling other functions, handles user input and output, and performs various other tasks.

Comments

Comments are used to provide explanations and documentation for the code only, that is, they are not executable statements. It’s considered good practice to always write comments for your code to improve the readability of the program. There are two types of comment styles supported in C++, namely- single-line comments and multiline comments. The symbol // is used for single-line comments (or single comments) while /\* and \*/ are used for writing multiline comments.

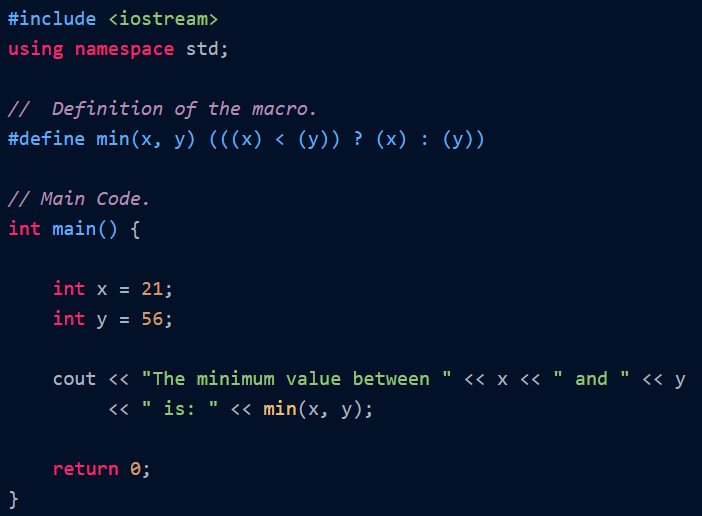


C++ Macros

A C++ macro is defined as a section of code that that particular macro value can replace. We can define the macro by using a #define directive. When the compiler goes to the macros while program compilation, the macro's name is replaced by the definition of the macro. The termination of the C++ Macros does not need a semi-colon (;).

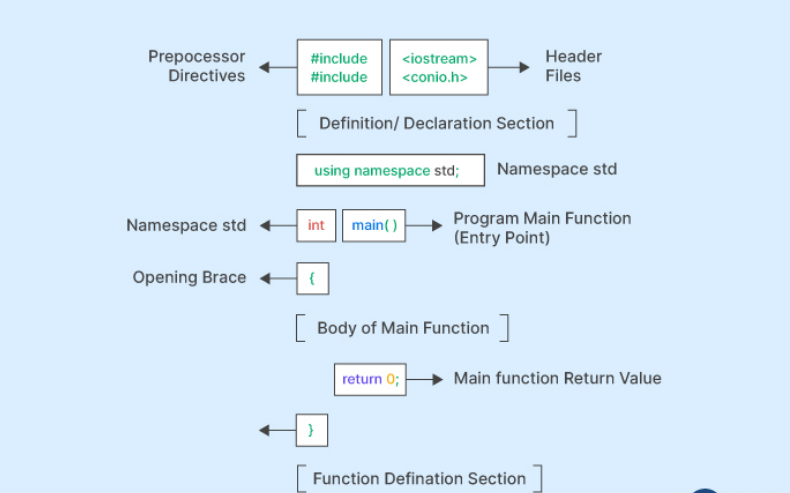






More details can be found in below link:

<https://www.scaler.com/topics/cpp-macro/>



## Step-by-Step Guide to Compile and Execute C++ Program

Compiling and executing a C++ program involves several steps, each of which is crucial to ensure the smooth and successful execution of the program. They are:

#### 1.  Writing the code:

The first step in the execution of a C++ program is to write the code/ lines of code for the specified purpose. This involves utilizing all the components we learned above as required in a logical manner.

#### 2.  Saving the code:

Once the code is written, we have to save the code in a file with a .cpp extension. This is the extension for source code files in C++. This file of source code contains all the code that will be compiled and then executed in the steps ahead.

#### 3.  Preprocessing:

Before the code is compiled, it goes through a preprocessor. At this stage, the preprocessor checks the code for any preprocessor directives such as include statements, #define statements, etc., and replaces them with the appropriate code. This ensures that all the necessary libraries, standard libraries, and header files are included in the code.

#### 4.  Compiling:

During the compilation process, the C++ compiler reads the code within the curly braces and checks for any syntax errors. It then compiles the code into object code (i.e. binary code or machine-readable instructions).

#### 5.  Linking:

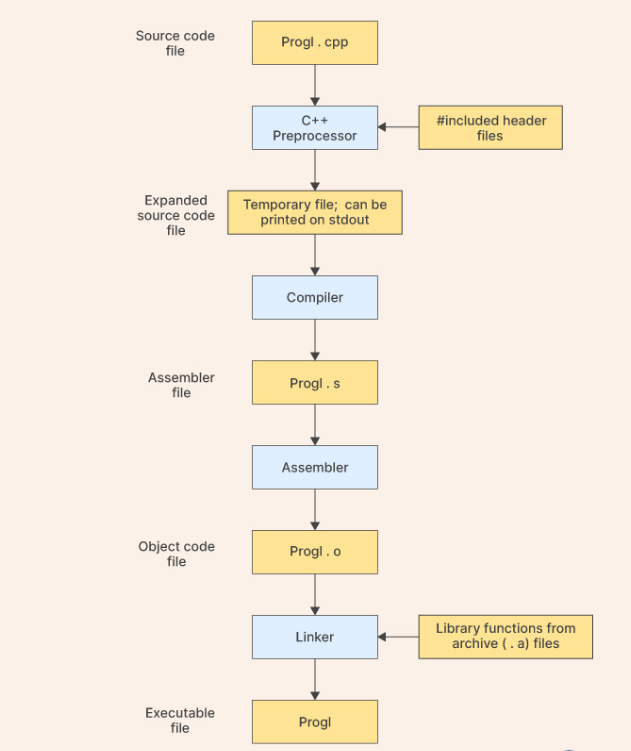
After the compilation of the code, it needs to be linked to the necessary libraries and header files that we have included in the program. This ensures that the program has access to all the functions and variables defined in the program. The linker also checks for any unresolved symbols in the object code and links the code to the appropriate libraries.

#### 6.  Generating Executable:

After the code is successfully compiled and linked, it is ready to be executed. This file is called an executable file. This file contains all the machine-readable instructions that will be executed by the computer.

#### 7.  Execution:

The final step is to execute the program. The executable file is loaded into memory, and the program is executed.



## Recap Preprocessing

Preprocessor directives are invoked to perform various pre-processing tasks, such as importing header files, declaring namespaces, defining constants, and other such operations that need to be done before the program starts executing.

* Header files usually contain function prototypes, class definitions, macro definitions, and other declarations that are necessary for the program to use a particular library or API.
* They are also used to define Macros and constants. Macros are defined using the #define directive and can take arguments that are replaced by their values when macros are expanded.

## Recap Compiling

Compilation The translation of C++ instructions into machine readable files is called compilation. This is done by a program called compiler.

###### The binary files produced by the compiler are object files (.obj in Windows and .o in Linux/Unix).

###### Syntax errors in C++ code are detected by the compiler — compilation error.

###### Compilation errors are pleasant (!) Easy to correct. The compiler messages are usually helpful and link you directly to the place of error.

## Compiler Recommended

###### VS Code: G++.exe build and debug active file

###### <https://code.visualstudio.com/docs/cpp/config-mingw>

###### MS VS: Embedded compiler

###### <https://learn.microsoft.com/en-us/cpp/get-started/?view=msvc-170>

## What is CMake

In software development, CMake is cross-platform free and open-source software for build automation, testing, packaging and installation of software by using a compiler-independent method. CMake is not a build system itself; it generates another system's build files. It supports directory hierarchies and applications that depend on multiple libraries. It can invoke native build environments such as Make, Qt Creator, Ninja, Android Studio, Apple's Xcode, and Microsoft Visual Studio. It has minimal dependencies, requiring only a C++ compiler on its own build system.

More details in <https://cmake.org/>

## First C++ Program – ‘Hello World’

## Type of program in Windows

###### Console application

A console application or command-line program is a computer program (applications or utilities) designed to be used via a text-only user interface, such as a text terminal, the command-line interface of some operating systems (Unix, DOS etc.) or the text-based interface included with most graphical user interface (GUI) operating systems, such as the Windows Console in Microsoft Windows, the Terminal in macOS, and xterm in Unix.

###### Dynamic Linked Library (dll)

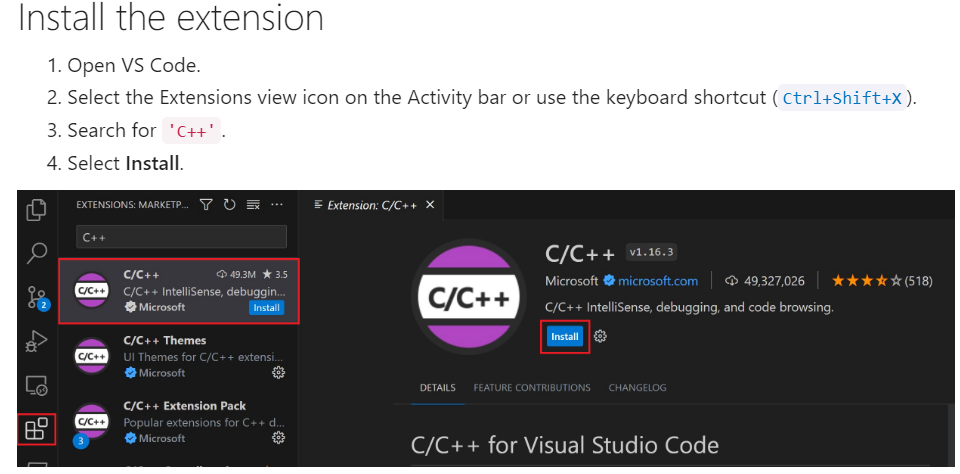
In Windows, a dynamic-link library (DLL) is a kind of executable file that acts as a shared library of functions and resources. Dynamic linking is an operating system capability. It enables an executable to call functions or use resources stored in a separate file

## Create your first console application

* Start VS code or Visual studio (create a project with application type as console application)
* Create a new file named helloworld.cpp
* Type below syntax
* Compile and run

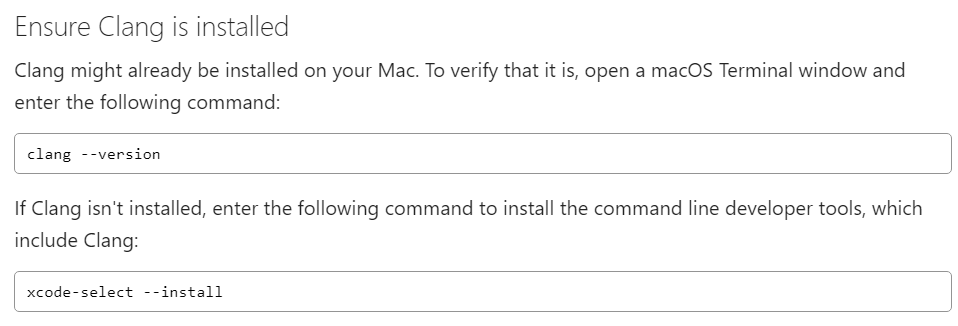
###### Step by step for Setup VS Code

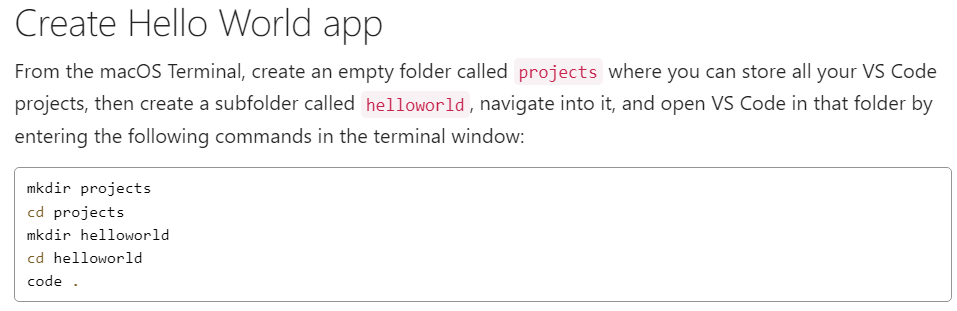
[https://code.visualstudio.com/docs/languages/cpp#\_install-the-extension](https://code.visualstudio.com/docs/languages/cpp" \l "_install-the-extension)

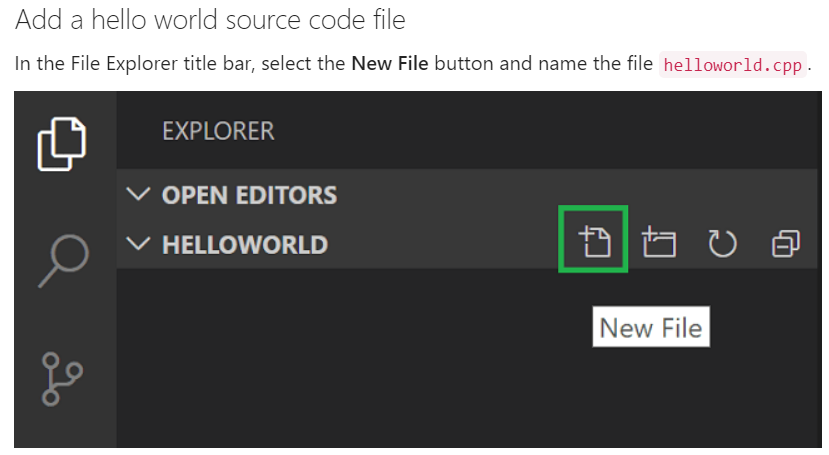


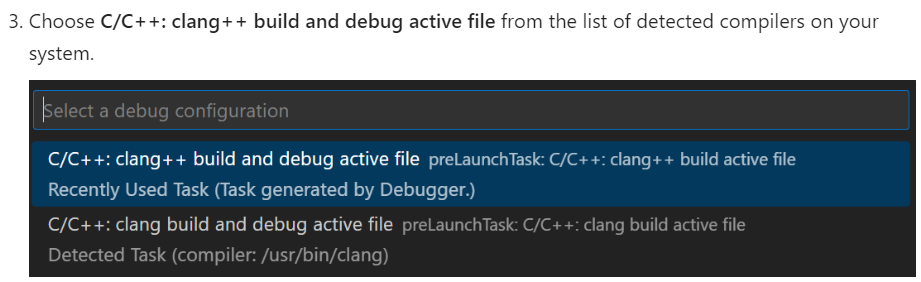
For Mac

[https://code.visualstudio.com/docs/cpp/config-clang-mac#\_prerequisites](https://code.visualstudio.com/docs/cpp/config-clang-mac" \l "_prerequisites)



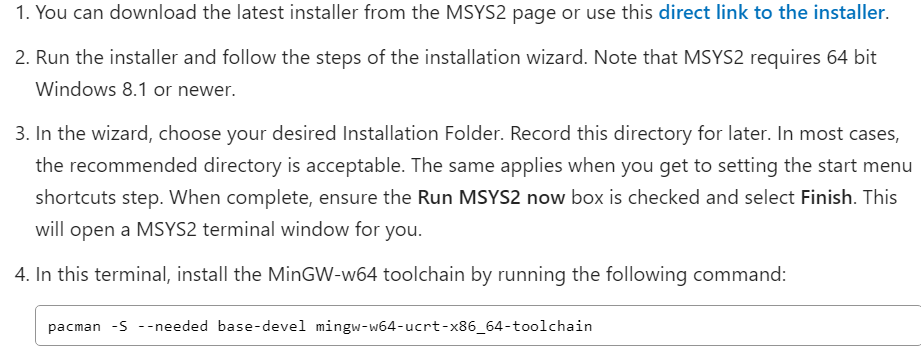


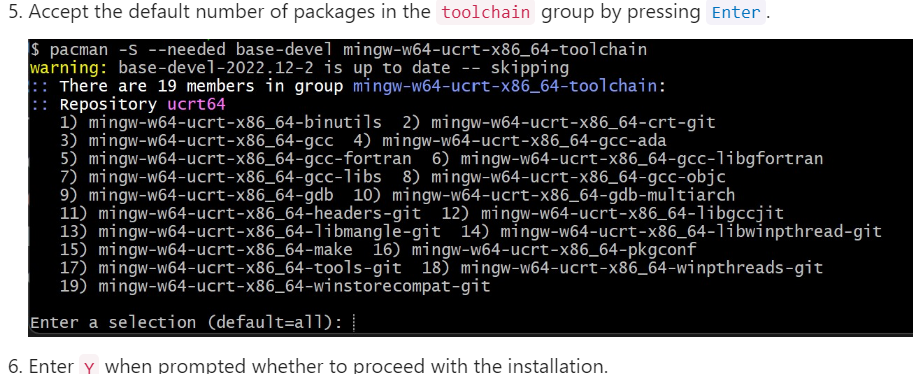


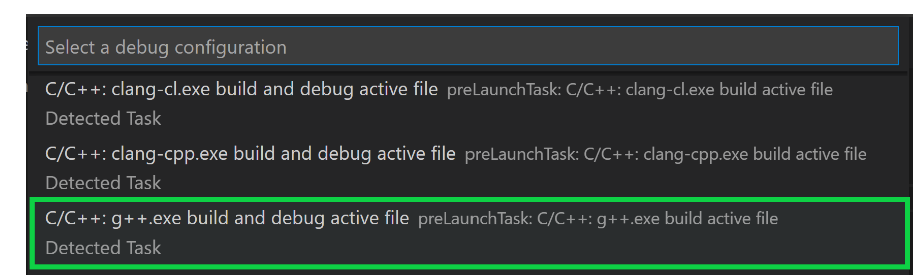
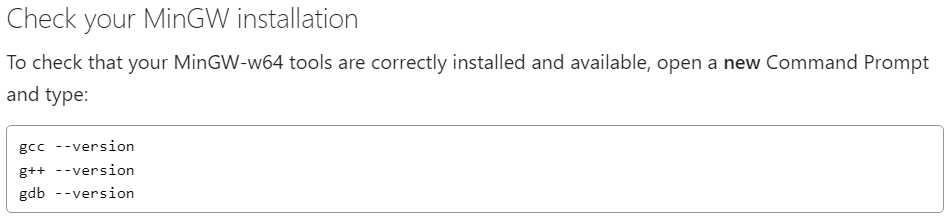
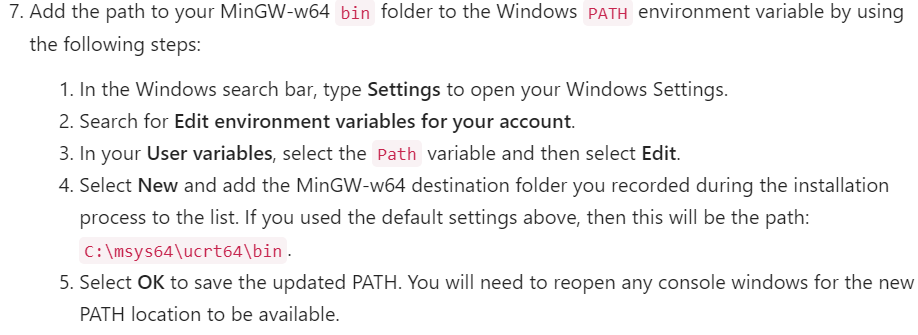


For windows:

[https://code.visualstudio.com/docs/cpp/config-mingw#\_prerequisites](https://code.visualstudio.com/docs/cpp/config-mingw" \l "_prerequisites)







Source code:

#include <iostream>

#include <vector>

#include <string>

using namespace std;

int main()

{

vector<string> msg {"Hello", "C++", "World", "from", "VS Code", "and the C++ extension!"};

for (const string& word : msg)

{

cout << word << " ";

}

cout << endl;

}

###### Assignment

1. Setup your working group IDE, install the application and packages.
2. Submit your working group source code of first C++ console application, and take screen shot of running result on console. In the program you have written, indicate the different component type.