Compilation program summary:

\*\*\*We must have a compiler which can be possible for use on operating system

\*\*\*Firstly : we have to create a c/c++ program and save the file as filename.c or filename.cpp

We have 4 phrase inside the compilation process:

1. Pre-processing :

The preprocessed output is stored in the **filename.i**.

* Removal of Comments

-Comments are stripped off.

* Expansion of Macros

To simplify and reduce the amount of repetitive coding

To reduce errors caused by repetitive coding

To make an assembly program more readable

* Expansion of the included files
* Conditional compilation

1. Compilation

The next step is to compile filename.i and produce an; intermediate compiled output file **filename.s**. This file is in assembly level instructions.

1. Assembly

In this phase the filename.s is taken as input and turned into **filename.o** by assembler. This file contain machine level instructions. At this phase, only existing code is converted into machine language.

1. Linking

This is the final phase in which all the linking of function calls with their definitions are done. Linker knows where all these functions are implemented. Linker does some extra work also, it adds some extra code to our program which is required when the program starts and ends.

THE GCC COMPILER :

- GCC support for C or C++ programs.

-First of all, GCC is a portable compiler--it runs on most platforms available today, and can produce output for many types of processors. In addition to the processors used in personal computers, it also supports microcontrollers, DSPs and 64-bit CPUs.

-GCC is not only a native compiler--it can also *cross-compile* any program, producing executable files for a different system from the one used by GCC itself. This allows software to be compiled for embedded systems which are not capable of running a compiler. GCC is written in C with a strong focus on portability, and can compile itself, so it can be adapted to new systems easily.

-GCC has multiple language *frontends*, for parsing different languages. Programs in each language can be compiled, or cross-compiled, for any architecture. For example, an ADA program can be compiled for a microcontroller, or a C program for a supercomputer.

-GCC has a modular design, allowing support for new languages and architectures to be added. Adding a new language front-end to GCC enables the use of that language on any architecture, provided that the necessary run-time facilities (such as libraries) are available. Similarly, adding support for a new architecture makes it available to all languages.

-Finally, and most importantly, GCC is free software, distributed under the GNU General Public License (GNU GPL).[(1)](http://lampwww.epfl.ch/~fsalvi/docs/gcc/www.network-theory.co.uk/docs/gccintro/gccintro_foot.html" \l "FOOT1) This means you have the freedom to use and to modify GCC, as with all GNU software. If you need support for a new type of CPU, a new language, or a new feature you can add it yourself, or hire someone to enhance GCC for you. You can hire someone to fix a bug if it is important for your work.

-Furthermore, you have the freedom to share any enhancements you make to GCC. As a result of this freedom you can also make use of enhancements to GCC developed by others. The many features offered by GCC today show how this freedom to cooperate works to benefit you, and everyone else who uses GCC.

THE CLANG COMPILER

-The Clang ASTs and design are intended to be [easily understandable](https://clang.llvm.org/features.html#simplecode) by anyone who is familiar with the languages involved and who has a basic understanding of how a compiler works.

- [Clang's support for C++](https://clang.llvm.org/compatibility.html#cxx) is more compliant than GCC's in many ways.

- Clang is designed as an API from its inception, allowing it to be reused by source analysis tools, refactoring, IDEs (etc) as well as for code generation.

- Clang does not implicitly simplify code

- Clang can serialize its AST out to disk and read it back into another program, which is useful for whole program analysis.

- Clang is [much faster and uses far less memory](https://clang.llvm.org/features.html#performance) than GCC.

- Clang has been designed from the start to provide extremely clear and concise diagnostics (error and warning messages), and includes support for [expressive diagnostics](https://clang.llvm.org/diagnostics.html).

-  [Clang uses a BSD license,](https://clang.llvm.org/features.html#license) which allows it to be embedded in software that is not GPL-licensed.

- Clang inherits a number of features from its use of LLVM as a backend, including support for a bytecode representation for intermediate code, pluggable optimizers, link-time optimization support, Just-In-Time compilation, ability to link in multiple code generators, etc.

- Clang supports [many language extensions](https://clang.llvm.org/docs/LanguageExtensions.html), some of which are not implemented by GCC. For instance, Clang provides attributes for checking thread safety and extended vector types.