Hi CPPBuddies !

INSPIRE CLUB, MANIT BHOPAL

BRINGS

# C++ Complete Bootcamp

Learn How To Apply Problem Solving Skills

Hello CPPBudddies

**DAY 14**

---

Welcome

To

# C++ COMPLETE BOOTCAMP

Your Guide To A Solid Foundataion in C++

Let us begin

# C++ PROGRAMMING LANGUAGE

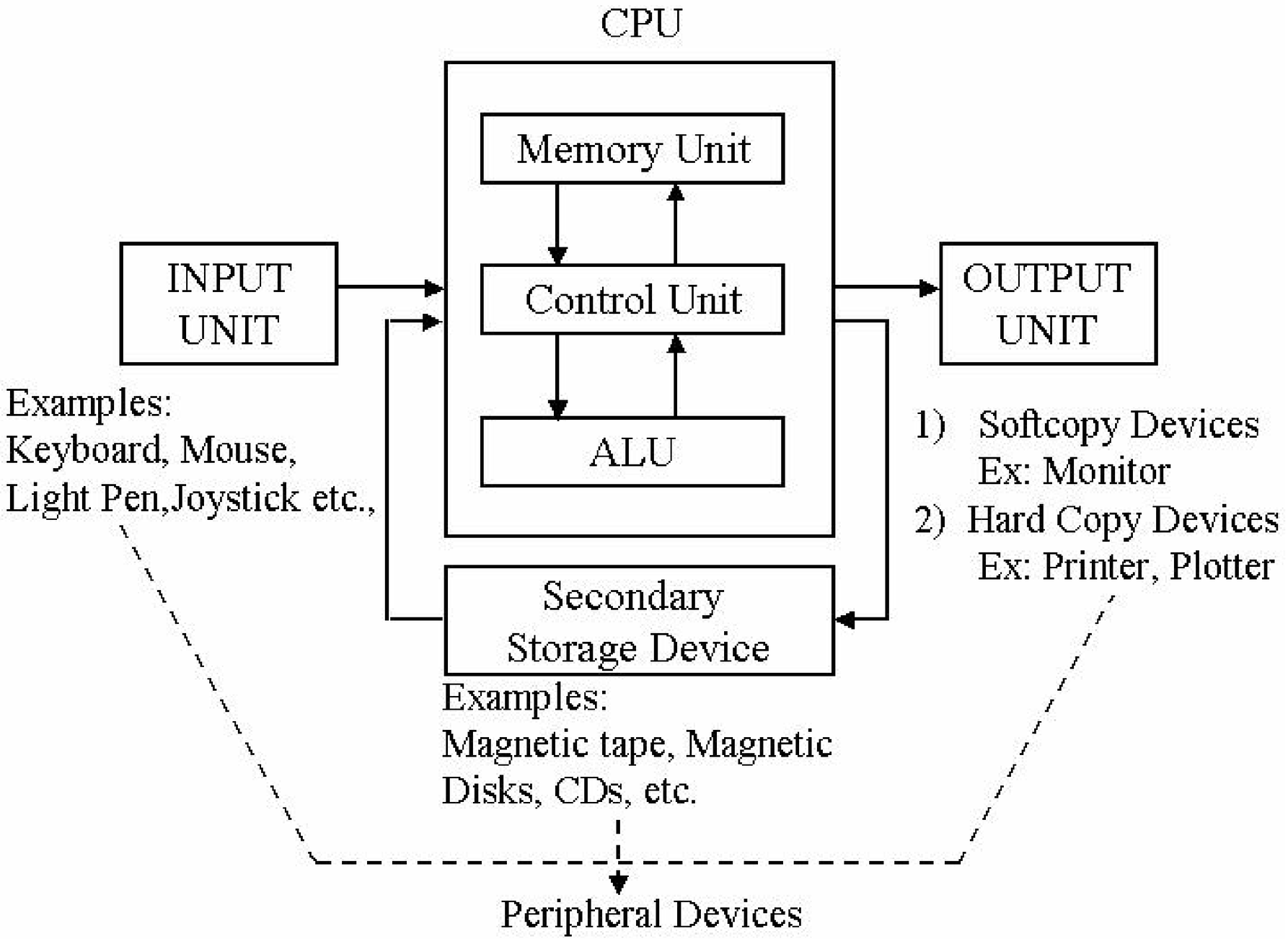
The best language for learning programming and DSA

ACTION TIME

# C++ COMPLETE BOOTCAMP

# Computer

A computer is an electronic device that manipulates information, or data.
It has the ability to store, retrieve, and process data


CPU
Memory Unit
Control Unit
ALU
INPUT UNIT
OUTPUT UNIT
Secondary Storage Device
Examples:
Keyboard, Mouse,
Light Pen,Joystick etc.,
1) Softcopy Devices
Ex: Monitor
2) Hard Copy Devices
Ex: Printer, Plotter
Examples:
Magnetic tape, Magnetic
Disks, CDs, etc.
Peripheral Devices

Computer are programmable machines

They are following the concept of
**STORED PROGRAM COMPUTERS**

This means they can store the programs and data, and operate on them to produce more data

Computers are very fast and very accurate

**They can perform billions of instructions per second**

Computers can store huge amount of data

**They can give you access to data very fast**

But they are dumb machines

We need to program them to get tasks done

# Introduction to programming languages

Modern computers are incredibly fast,
and getting faster all the time.
However, computers also have some significant constraints:
they only natively understand a limited set of commands,
and must be told exactly what to do.

Computers are not so smart => So we need to tell them exactly what to do and how to do

# What is Programming?

A computer program (also commonly called an application)
is a set of instructions that the computer can perform
in order to perform some task.

**The process of creating a program is called programming.**

Programmers typically create programs by
producing source code (commonly shortened to code),
which is a list of commands typed into one or more text files.

# Hardware vs Software

The collection of physical computer parts that make up a computer and execute programs is called the hardware.
When a computer program is loaded into memory and the hardware sequentially executes each instruction, this is called running or executing the program.

Software is a collection of instructions and data that tell a computer how to work.
This is in contrast to physical hardware, from which the system is built and actually performs the work.

# Hardware vs Software

Hardware => electrical and electronics engineer

Software => computer and IT engineers

**Hardware is tangible, but not software**


hardware executes

software tells what and how to execute

# Types of programming languages

1. machine language programming
2. assembly language programming
3. high-level language programming
4. very high-level language programming

# Machine Language

**A computer's CPU is incapable of speaking C++.**
The limited set of instructions that a CPU can understand
directly is called machine code
(or machine language or an instruction set).

Here is a sample machine language instruction: 10110000 01100001

Back when computers were first invented,
programmers had to write programs directly in machine language,
which was a very difficult and time consuming thing to do.

# Assembly Language

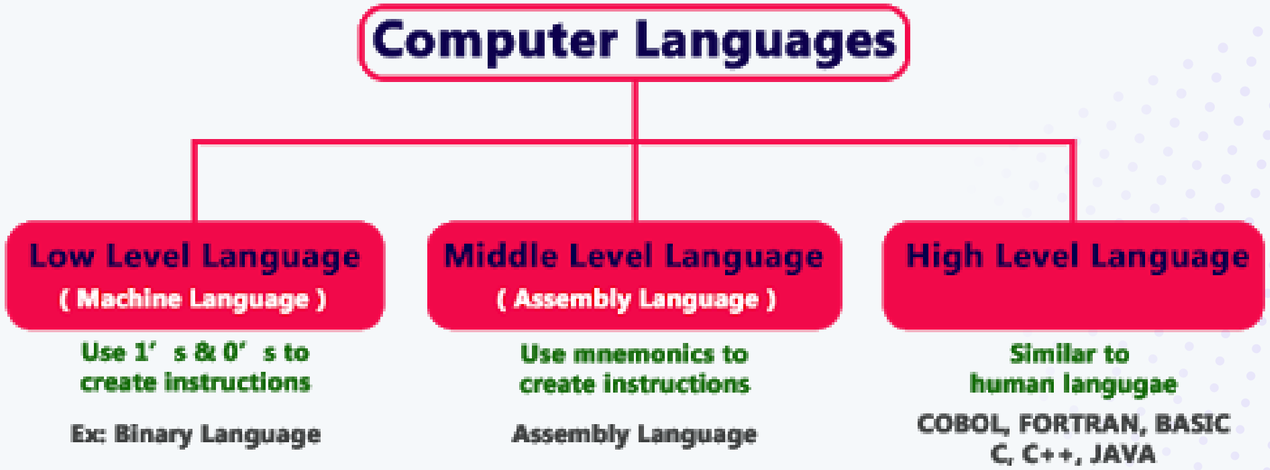
Because machine language is so hard for humans
to read and understand, assembly language was invented.
In an assembly language, each instruction is identified
by a short abbreviation (rather than a set of bits),
and names and other numbers can be used.
They are called mnemonics.

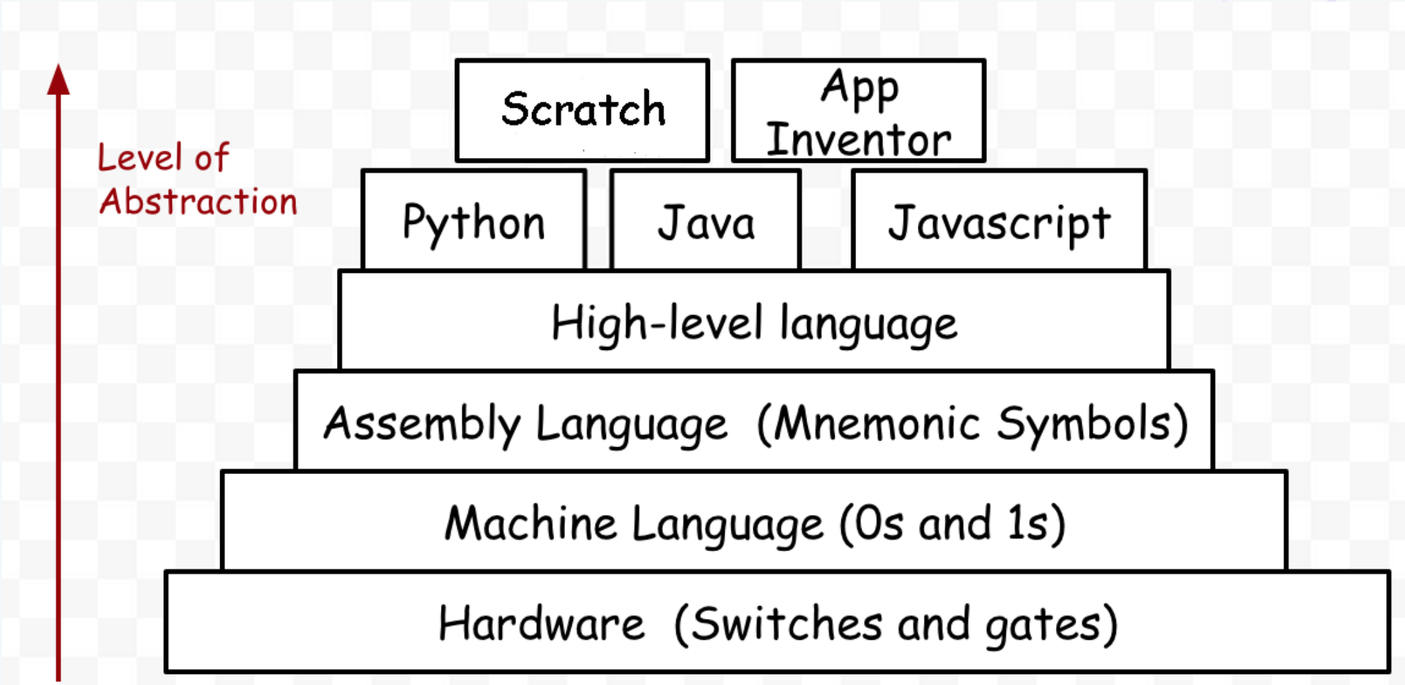
Here is the same instruction as above in assembly language: mov al, 061h

# High-level Languages

To address the readability and portability concerns,
new programming languages such as C, C++, Pascal
(and later, languages such as Java, Javascript, and Perl) were developed.
These languages are called high level languages,
as they are designed to allow the programmer to write programs
without having to be as concerned about
what kind of computer the program will be run on.

**They are easy to learn**


Computer Languages
Low Level Language
( Machine Language )
Use 1' s & 0' s to create instructions
Ex: Binary Language
Middle Level Language
( Assembly Language )
Use mnemonics to create instructions
Assembly Language
High Level Language
Similar to human langugae
COBOL, FORTRAN, BASIC
C, C++, JAVA


Level of Abstraction
Scratch
App Inventor
Python
Java
Javascript
High-level language
Assembly Language (Mnemonic Symbols)
Machine Language (0s and 1s)
Hardware (Switches and gates)

# Translators

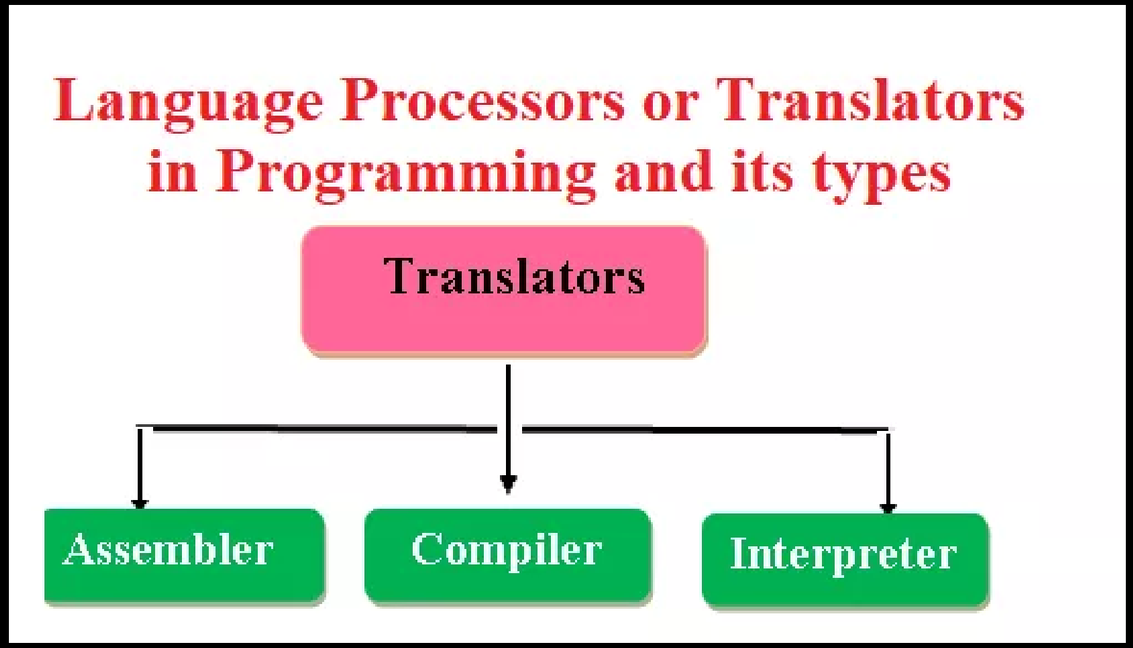
A translator is a programming language processor that converts a computer program from one language to another.

It takes a program written in source code and converts it into machine code.

It discovers and identifies the error during translation.

# Purpose

It translates **high-level language program** into a **machine language program** that the **central processing unit** (CPU) can understand



# COMPILER

A compiler is a translator used to convert **high-level programming language** to **low-level programming language.**

It converts the whole program in one session and reports errors detected after the conversion.

Compiler takes time to do its work as it translates high-level code to lower-level code all at once and then saves it to memory.

- **Microsoft Visual Studio**
- **GNU Compiler Collection (GCC)**
- **Common Business Oriented Language (COBOL)**

# INTERPRETER

Just like a compiler, is a translator used to convert **high-level programming language** to **low-level programming language.**

It converts the program **one at a time** and reports errors detected at once, while doing the conversion.

With this, it is easier to detect errors than in a compiler as it does **LINE BY LINE EXECUTION.**

- **OCaml**
- **List Processing (LISP)**
- **Python**

How to manually run C++ files.

**We have to use terminal**

g++ filename.cpp

g++ filename.cpp -o executable.cpp

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

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

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

AKASH MAJI@Akash320 MINGW64 ~/Desktop/CPP
$ g++ -v
Using built-in specs.
COLLECT_GCC=C:\MinGW\bin\g++.exe
COLLECT_LTO_WRAPPER=c:/mingw/bin/../libexec/gcc
Target: mingw32
Configured with: ../src/gcc-6.3.0/configure --b
mpc=/mingw --with-isl=/mingw --prefix=/mingw --
jc,obj-c++,fortran,ada --with-pkgversion='MinGW
isable-sjlj-exceptions --enable-version-specifi
ibstdcxx-debug --with-tune=generic --enable-lib
Thread model: win32
gcc version 6.3.0 (MinGW.org GCC-6.3.0-1)

AKASH MAJI@Akash320 MINGW64 ~/Desktop/CPP
$ git --version
git version 2.30.1.windows.1

AKASH MAJI@Akash320 MINGW64 ~/Desktop/CPP
$
```

# Hello World Program

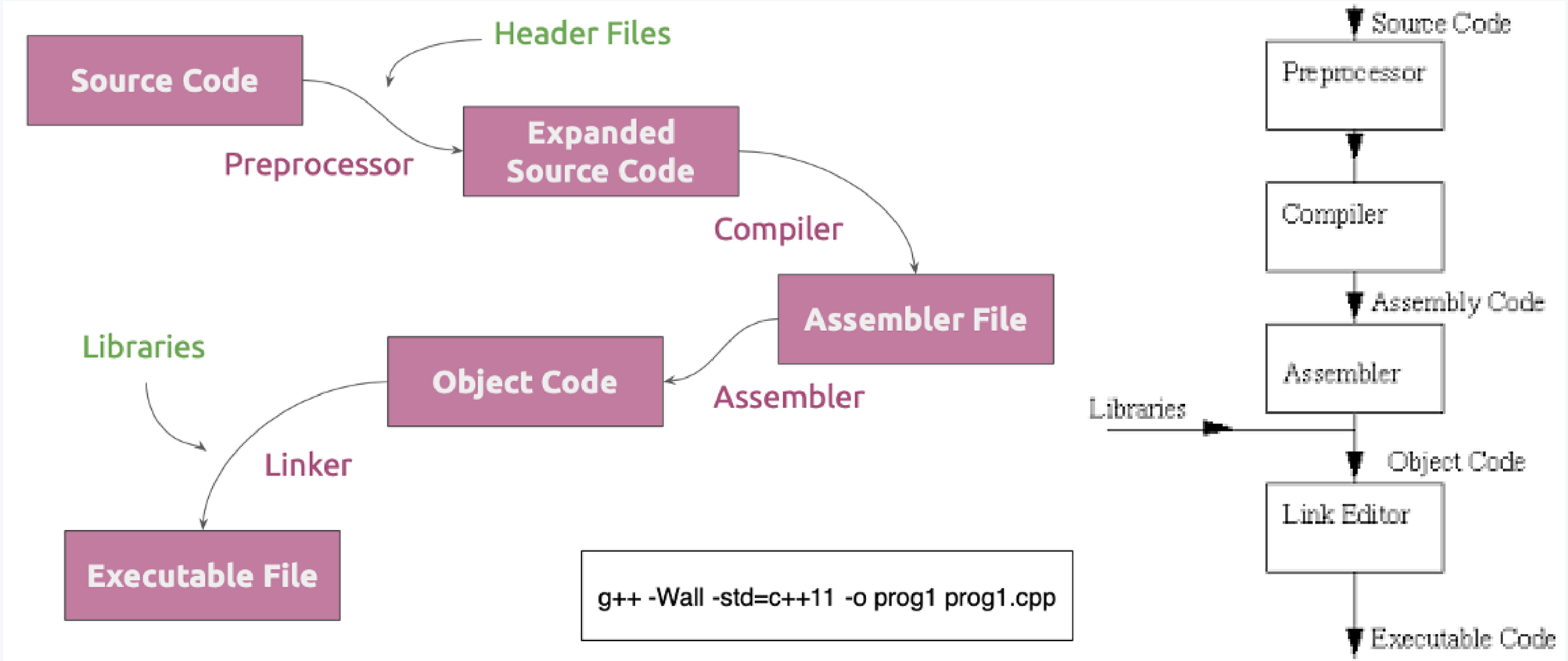
Basic Program
for
Beginners

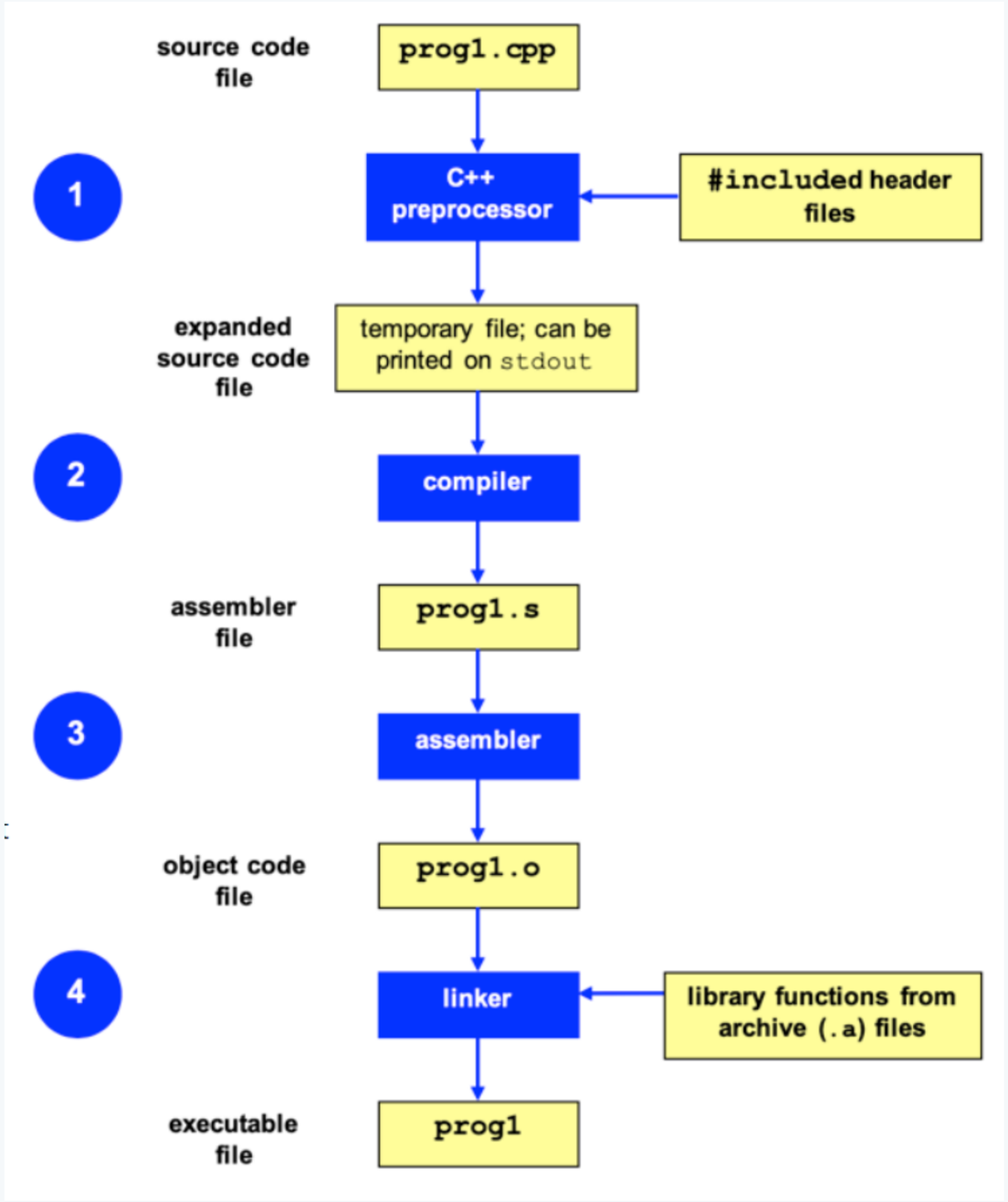
<====== check g++ compiler version

How to compile
$ **g++ main.cpp -o main.exe**
How to execute
$ **./main.exe**

<====== check git version

# C++ is a COMPILED language






# The C++ Build Process

Building an executable file from
a C++ source code file is a multi-step process.

For example, if you have a C++ source code file
named prog1.cpp and you execute the command

**g++  -o prog1 prog1.cpp**

Another detailed approach
**g++ -Wall -Werror -std=c++11 -o prog1 prog1.cpp**

## The build process looks like this:

1. The C++ preprocessor copies the contents of the included header files into the source code file, generates macro code, and replaces symbolic constants defined using #define with their values.
2. The expanded source code file produced by the C++ preprocessor is compiled into the assembly language for the platform.
3. The assembler code generated by the compiler is assembled into the object code for the platform.
4. The object code file generated by the assembler combined with functions from the standard library files by the linker to produce an executable file.

**By default, this executable file is named a.out.**

Here, we have used the **-o** option to specify the name of the executable file as **prog1**.

# By using appropriate compiler options, we can stop this process at any stage.

**To stop the process after the preprocessor step, you can use the -E option:**

**g++  -E prog1.cpp**

The expanded source code file will be printed on standard output (the screen by default); you can redirect the output to a file if you wish.

**To stop the process after the compile step, you can use the -S option:**

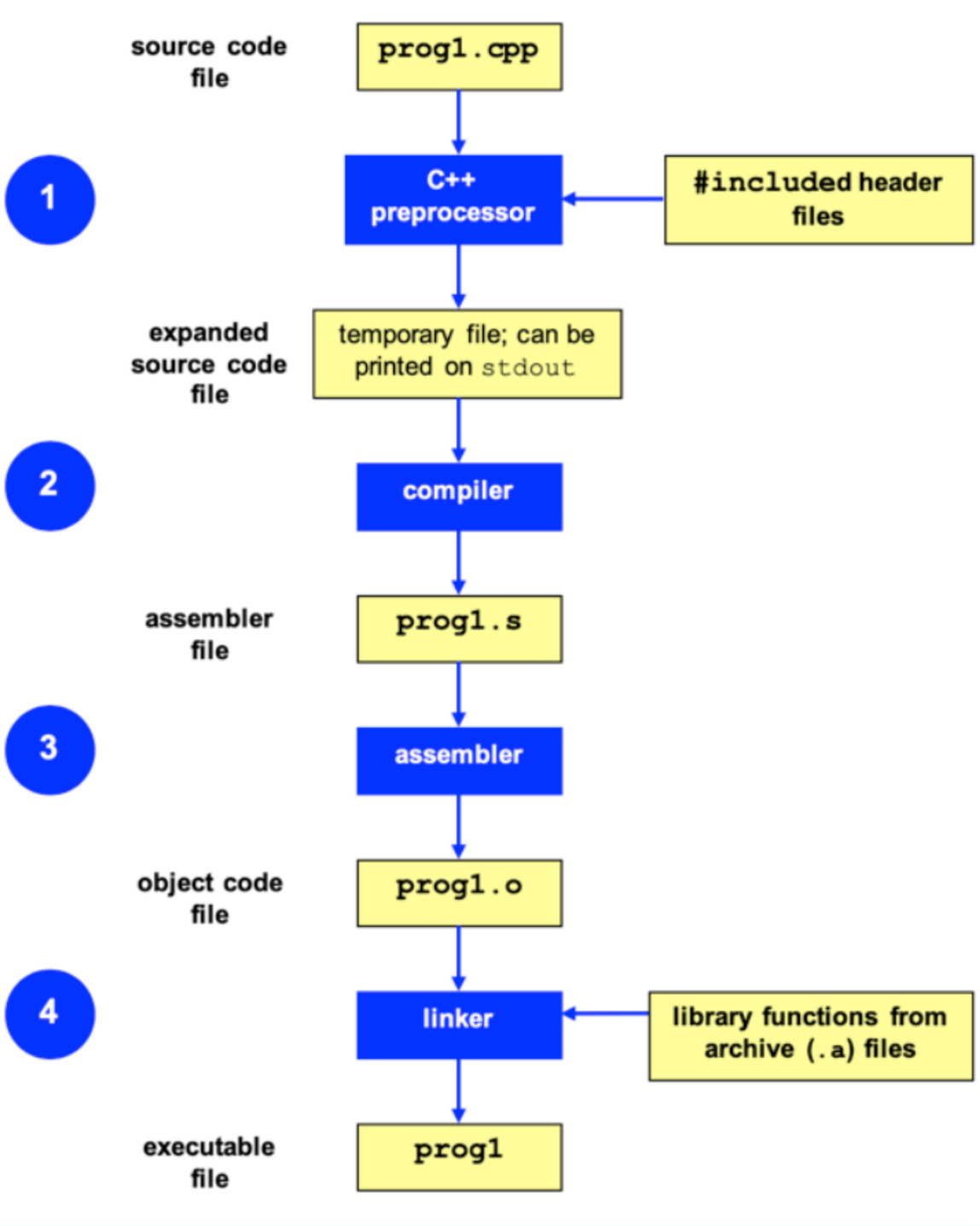
**g++  -S prog1.cpp**

By default, the assembler code for a source file named filename.cpp will be placed in a file named filename.s

**To stop the process after the assembly step, you can use the -c option:**

**g++  -c prog1.cpp**

By default, the object code for a source file named filename.cpp will be placed in a file named filename.o




C++ Pre-Processor

**Source Code => Extended Source Code**

**g++ -E prog1.cpp**

Compiler

**Extended Source Code => Assembly Code**

**g++ -S prog1.cpp**

Assembler

**Assembly Code => Object Code**

**g++ -c prog1.cpp**

Linker

**Object Code => Executable Code**

**g++ prog1.cpp -o prog1.exe**


# C++ BUILD PROCESS

# ASK YOUR DOUBTS

do not let doubts bother you

# THANK YOU

keep calm,
wear mask,
and
study hard

ISSUED IN PUBLIC INTEREST

whoami

**AKASH MAJI**

Your Mentor