A BRIEF INTRODUCTION TO PROGRAMMING LANGUAGES AND C

Programming Languages

Generations

There are well over 2500 programming languages and their number continues to increase. The different generations of programming languages include:

- **Machine languages:** These are the **native languages** that the CPU processes. Each manufacturer of a CPU provides the instruction set for its CPU.
- **Assembly languages:** These are **readable versions** of the native machine languages. Assembly languages simplify coding considerably. Each manufacturer provides the assembly language for its CPU.
- **Third-generation languages:** These languages are procedural: they identify the instructions or procedures involved in reaching a result. The instructions are **NOT tied to any particular machine**. Examples include **C, C++, and Java.**
- **Fourth-generation languages:** These languages describe what is to be done without specifying how it is to be done. These instructions are NOT tied to any particular machine. Examples include **SQL, Prolog, and Matlab.**
- **Fifth-generation languages:** We use these languages for **artificial intelligence, fuzzy sets, and neural networks.**

The **third, fourth, and fifth-generation languages are high-level languages**. They exhibit **no direct connection to any machine language**. Their instructions are **more human-like** and **less machine-like**. A program written in a high-level language is relatively **easy to read** and relatively **easy to port across different platforms.**

Features of C

- C is English-like.
- C is quite compact - has a small number of keywords
- C is the lowest in level of the high-level languages
- C can be faster and more powerful than other high-level languages
- C programs that need to be maintained are large in number
- C is used extensively in high-performance computing
- UNIX, Linux, and Windows operating systems are written in C and C++

The C Compiler

THE C PROGRAM COMPILATION TAKES PLACE IN THE FOLLOWING WAY

- The code written in the C language (also known as **source code** - this is the input or program instructions for the compiler) is converted into **machine-level equivalents (or Binary Code**, this is the output of the compiler) an **exe file** is generated by the compiler
- The OS loads the Binary Code (given by the compiler) into the RAM and starts executing the code based on the inputs given by the user and generates the output for it.

SUMMARY

C PROGRAM -------> COMPILER -------> MACHINE LANGUAGE PROGRAM -------> USER

C COMPILER FOR LINUX

gcc (GNU Compiler Collection)

- Command to create the Binary code version of our Source Code – **gcc hello.c** [assuming our source code is written in a file named hello.c]
- The executable/output file produced by the gcc compiler by default will be – **a.out** [This contains all the machine language instructions needed to execute the program]
- To produce an executable/output file with a name of your choice (other than the a.out), we have to replace **gcc hello.c** with **gcc hello.c -o hello**

C COMPILER FOR WINDOWS

cl (Clang)

- Command to create the Binary code version of our Source Code **– cl hello.c**
- The executable/output file produced by the gcc compiler by default will be – **hello.exe** [This contains all the machine language instructions needed to execute the program]
- To produce an executable/output file with a name of your choice (other than the a.out), we have to replace **cl hello.c** with **cl hello.c -o myprogram**