Day 2: High Level, Low Level Language, Compilers / Assembler

Day 1 Recap Session

- Computer building blocks
- 4 basic operations how CPU works

What is a program?

- A set of instructions to the computer that perform a specified task in a specified manner.
- A program is like a recipe
- The task of writing a functional, error-free and cohesive set of instructions is called programming.
- Two major components of Programming:
  - Logic – What are my set of instructions?
  - Syntax – How do I convey them to the CPU?

Logic

- Programming logic: breaks down a complex real-world problem into a set of rules or languages for the computer, in order to perform a task.
- Divide a complex task into a series of simpler tasks in orderly sequence.
- Arrange simple tasks in the most efficient order to accomplish complex task.

Syntax

- Predetermined set of rules in which the instructions need to be provided.
- Usually unique to each programming language.
- Once the logic is established, convert the instructions into the syntax prescribed.

Programming languages

- A computer's CPU can only understand instructions that are written in machine language
- Assembly language was created in the early days as an alternative to machine languages
- Assembly language was difficult
- New generation of programming languages known as high-level languages
- Allow programmers to create powerful and complex programs without knowing how the CPU works

Programming languages

- Three types of programming languages:
  - Machine language (Low-level language)
  - Assembly language (Low-level language)
  - High-level language
- Low-level languages are closer to the language used by a computer, while high-level languages are closer to human languages

Low level & High level languages

- Programming languages range from low level (close to binary) to high level (close to human language).
- Order of languages:

  High-level --> Assembly --> Machine/Binary

- Low-level languages are extremely hard to learn and lack portability, but make optimal use of hardware.
- High-level languages are easier to learn and generalize, but are poorer at using hardware resources.

High-level languages

- Allow us to write computer code using instructions resembling everyday spoken language (for example: ***print***, ***if***, ***while***)
- Need to be translated into machine language before they can be executed
- Some use a **compiler** to perform this translation and others use an **interpreter**

Low level language: Machine language

- Made up of instructions and data that are written in binary numbers
- Difficult to write in, understand and debug

169 1 160 0 153 0 128 153 0 129 153 130 153 0 131

200 208 241 96

Low level language: Assembler language

- Consists of a series of instructions mnemonics that correspond to a stream of executable instructions
- Uses an **assembler**

mov a1, #061h

- Move the hexadecimal value 61 (97 decimal) into the processor register named "a1"
- Bridge between human language and machine language

Low level language: Assembler language

Compiled vs Interpreted

- Some languages have a **compiler** that converts your program (Source code) into a binary, executable file (Distributed code). E.g., C, C++, Fortran, Java etc.
- Other languages maintain the source code as is until the time of execution. Upon execution, the interpreter converts source code to machine code on-the-fly. E.g., Python, Perl, Ruby, Javascript etc.

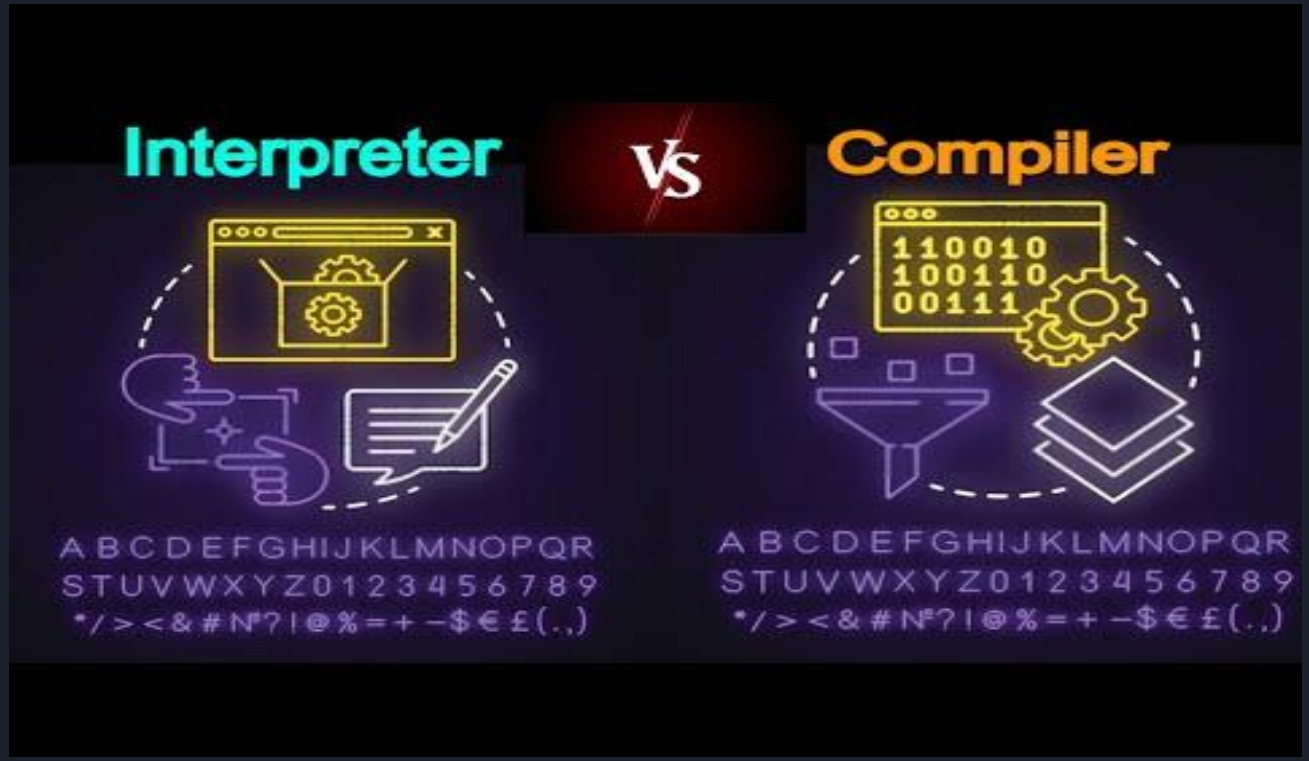
Compiler

- Read and analyze entire program and translates it as a whole into machine code of the specific machine
- The generated code would not necessarily work on other computer
- Generate an executable file
- A compiled code runs faster
- Display all errors after compilation
- Security of source code
- Eg. C and C++

Interpreter

- Reads through the code one line at a time and converts instructions in that line to machine code.
- Lacks the optimization step included in compilers – the translated code is executed as is
- May involve repeated analysis of some statements (loops, functions)
- Display error of each line
- Eg. Javascript, Python

Compiler vs Interpreter



Machine code is the ultimate result

All source code HAS to be eventually converted into machine code

Stages of programming

- Develop the logic
- Write code
- Compile/Interpret
- Test with data
- Debug code

Pseudocode

- A high level description of the problem and the programmatic solution that helps develop the overall logic and the subdivision of tasks.
- Informal way of writing code that does not worry about the syntax of the programming language.
- Follows a loose set of rules that allow logical grouping of functions.

Pseudocode Example

- Problem: Calculate the average length of all protein sequences.
- PseudoCode (Simplest):
  - Read all Protein sequences
  - Calculate length of each sequence
  - Calculate the mean of all lengths

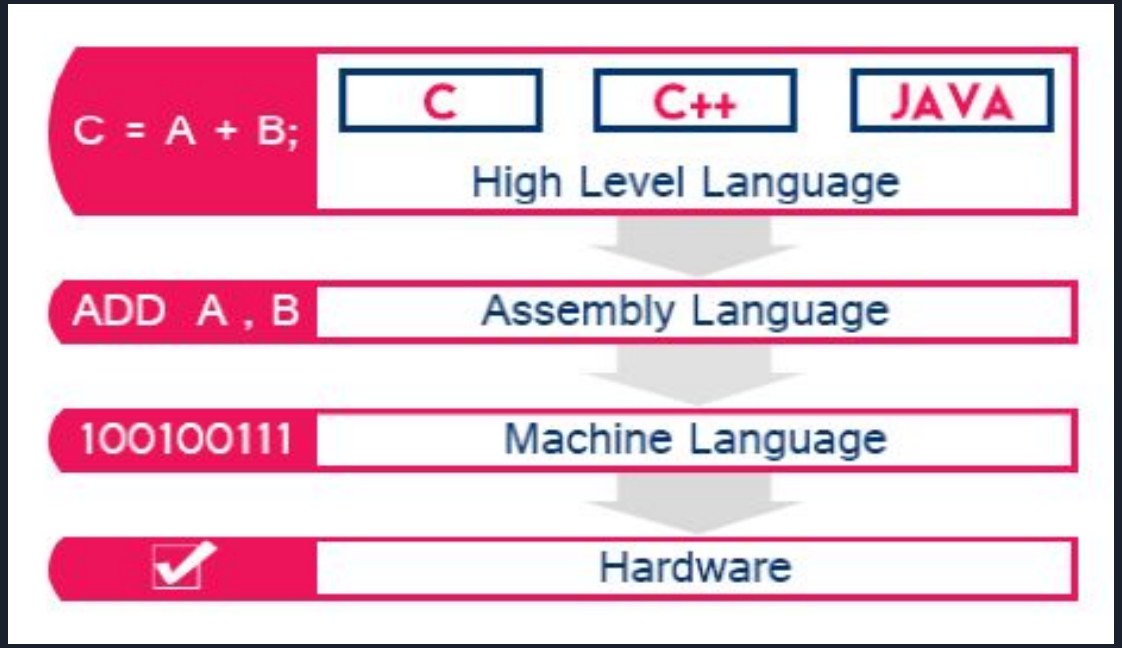
Pseudocode Example

- Problem: Calculate the average length of all protein sequences.
- PseudoCode (Simple):
  - Open protein sequence file
  - Loop over protein sequences
    - Read each sequence
    - Calculate length of current sequence
  - Add length to sum variable
  - Calculate the mean length by dividing sum/ no. of sequences

Pseudocode Example

- PseudoCode (Detailed):
  - Create mean length variable; set to 0
  - Create No. of sequences variable; set to 0
  - Create sum variable; set to 0
  - Ask user input for name of sequence file
  - Open protein sequence file (if exists)
  - Loop over protein sequences
    - Read each sequence
    - Calculate length of current sequence
    - Add sequence length to sum variable
    - Increment No. of sequences by 1
  - Calculate value of mean as Sum/No. of sequences.
  - Output value of mean

Recap


C = A + B;
C
C++
JAVA
High Level Language
ADD A , B
Assembly Language
100100111
Machine Language
Hardware

Recap

- Low level language:
  - the only language which can be understood by the computer
  - known as Machine Language
  - contains only two symbols, 1 & 0
- Assembler:
  - Translate human language to machine language
  - An output of any programming language
  - To improve the translated code before the processor executes it
- High level Language
  - can be understood by the users
  - needs to be converted into the low-level language to make it understandable by the computer
  - Eg: Javascript, Python, C, C++ etc.