Programming Fundamentals

Lecture 01

1

What is a Computer Program ?

- Computers don't read minds (yet), and thus we have to provide instructions to computers so they can perform these tasks.
- Computers don't speak natural languages (yet)—they only understand binary code.
- **Binary code** is unreadable by humans.

```
0110101101101011 1100000000110101 1011110100100100
1010010100100100 0010100100010011 1110100100010101
1110100100010101 0001110110000000 1110000111100000
0000100000000001 0100101011110100 0000001000101011
0010100101110000 0101001001001001 1010100110101000
```


2

What is a Computer Program ?

- A **computer program** is a sequence or set of instructions in a programming language for a computer to execute.
- It is one component of software, which also includes documentation and other intangible components.
- A computer program in its human-readable form is called **source code**.

3

Software Vs Program

- **Software** is a set of programs that enables the hardware to perform a specific task.
- Software is a generic term used to describe computer programs.
- Software can be categorized into two categories.
  - **System software**
  - **Application software**

4

Software in Real World

5

Programming Languages

- A **programming language** is a computer language engineered to communicate instructions to a machine.
- Programs are created through programming languages.
- They support to control the behavior and output of a machine through accurate **algorithms**, similar to the human communication process.

6

Machine Language

- **Machine language** (machine code) is a low-level computer language that is designed to be directly understandable by a computer.
- It is the language into which all programs must be converted before they can be run.
- All instructions, memory locations, numbers and characters are represented in 0's & 1's.

  00000100 10000000

7

Machine Language

Pros

- Can run and execute very fast as the code.

Cons

- Impossible for humans to use.
- Programs are hard to maintain and debug.
- No mathematical functions.
- Memory management needs to be done manually.

8

Assembly Language

- **Assembly language** replaces the instructions represented by patterns of 0's and 1's with alphanumeric symbols also called as **mnemonics**.
- It make it easier to remember and work with them including reducing the chances of making errors.

  ADD 3, 5, result

  SUB 1, 2, result

9

Assembly Language

- Because of alphanumeric symbols, assembly language is also known as **Symbolic Programming Language**.
- **Assembler** is a program that is used to convert the alphanumeric symbols written in assembly language to machine language
- This machine language can be directly executed on the computer.

10

Assembly Language

```
.equ STDOUT, 1
.equ SVC_WRITE, 64
.equ SVC_EXIT, 93

.text
.global _start

_start:
    stp x29, x30, [sp, -16]!
    mov x0, #STDOUT
    ldr x1, =msg
    mov x2, 13
    mov x8, #SVC_WRITE
    mov x29, sp
    svc #0 // write(stdout, msg, 13);
    ldp x29, x30, [sp], 16
    mov x0, #0
    mov x8, #SVC_EXIT
    svc #0 // exit(0);

msg:    .ascii "Hello World!\n"
.align 4
```

"Hello World!" program in assembly language (for ARM64 architecture)

11

Assembly Language

Cons

- There are no symbolic names for memory locations.
- It is difficult to read.
- Assembly language is machine-dependent making it difficult for portability

12

High-Level Language

- High-level languages are written in a form that is close to our human language, enabling programmers to just focus on the problem being solved.
- High-level languages are **platform independent**.
- That is programs written in a high-level language can be executed on different types of machines.

13

High-Level Language

- A program written in the high-level language is called **source program** or **source code**.
- It is any collection of human-readable computer instructions.
- Hello World program in C++:

```
#include <iostream>

int main () {
  std::cout << "Hello World!" << std::endl;
}
```


14

High-Level Language

- In Python, the same program is even more succinct:

```
print('Hello World!')
```

- For a computer to understand and execute a source program written in high-level language, it must be translated into machine language.
- This translation is done using either **compiler** or **interpreter**.

15

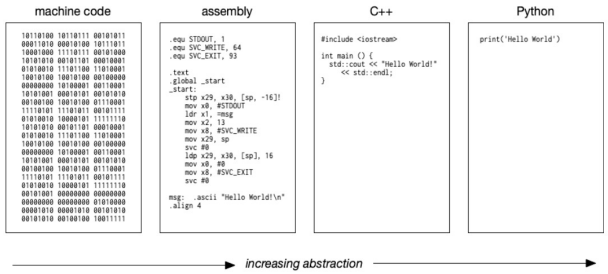
High-Level Language

Pros

- Easier to modify, faster to write code and debug as it uses English like statements.
- Portable code, as it is designed to run on multiple machines.

16

Abstraction in Languages



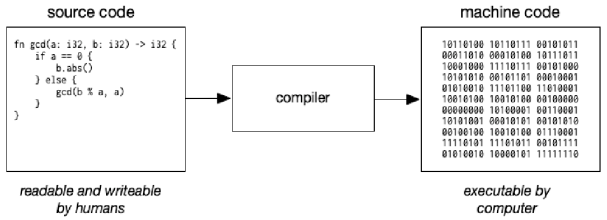
17

Compilation

- A compiler is a system software program that transforms high-level source code written by a software developer in a high-level programming language into a low-level machine language.
- Process of converting high-level programming language into machine language is known as **compilation**.

18

Compilation



19

Compiler

- Compilers translate source code all at once and the computer then executes the machine language that the compiler produced.
- The generated machine language can be later executed many times against different data each time.
- Programming languages like C, C++, C# and Java use compilers.

20

Compiler

- Compilers can be classified into **native-code** compilers and **cross** compilers.
- A compiler that is intended to produce machine language to run on the same platform that the compiler itself runs on is called a native-code compiler.
- A cross compiler produces machine language that is intended to run on a different platform than it runs on.

21

Interpreter

- An **interpreter** is a program that reads source code one statement at a time, translates the statement into machine language, executes the machine language statement, then continues with the next statement.
- Programming languages like Python, Ruby and Perl the source code is frequently executed directly using an interpreter.

22

Interpreter

- Generally compiled code run faster than to run a program under an interpreter.
- However, it can take less time to interpret source code than the total time needed to both compile and run it.
- Thus, interpreting is frequently used when developing and testing source code for new programs.

23

Programming Paradigms

- A programming paradigm is a style, or “way” of programming.
- Major programming paradigms are,
  - Imperative
  - Logical
  - Functional
  - Object-Oriented

24

Imperative Programming

- In Imperative programming the program describes a sequence of steps that change the state of the computer as each one is executed in turn.
- It explicitly tells the computer "how" to accomplish a certain goal.
- **Structured** programming is a subset of Imperative programming.
- It remove the reliance on the GOTO statement by introducing looping structures.

25

Imperative Programming

- **Procedural programming**, is another subset of Imperative programming.
- It use procedures to describe the commands the computer should perform.
- Procedural programming refers to the ability to combine a sequence of instructions into a procedure so that these instructions can be invoked from many places without resorting to duplicating the same instructions.

26

Imperative Programming

- The difference between procedure and function is that functions return a value, and procedures do not.
- An assembly language is an imperative language which is NOT structured or procedural.
- Popular programming language like C is imperative and structured in nature.

27

Logical Programming

- Logical paradigm fits exceptionally well when applied to problem domains that deal with the extraction of knowledge from basic facts and rules.
- Rules are written as logical clauses with a head and a body.

  Y is true if X1, X2, and X3 are true

28

Logical Programming

- Facts are expressed similar to rules, but without a body.

  Y is true.

- Idea in logical programming is that instead of telling the computer how to calculate things, you tell it what things are.
- Example: PROLOG

29

Functional Programming

- In functional programming languages, functions are treated as first-class objects.
- In other words, you can pass a function as an argument to another function, or a function may return another function.
- Examples of functional programming languages are F#, LISP, Scheme, and Haskel.

30

Object-Oriented Programming

- Object-oriented is the term used to describe a programming approach based on **objects** and **classes**.
- This paradigm allows to organize software as a collection of objects that consist of both data and behavior.
- Introduces concepts like **encapsulation, inheritance, and polymorphism** required in writing large and complex programs.

31

Object-Oriented Programming

- Object-oriented paradigm provides key benefits of reusable code and code extensibility.
- Examples of object-oriented languages are Python, C++, Java and C#.

32

Software Development

- Software development is a process by which stand-alone or individual software is created using a specific programming language.
- It involves writing a series of interrelated programming code, which provides the functionality of the developed software.
- Software development may also be called application development.

33

Software Development Life Cycle

- Process of software development goes through a series of stages in stepwise fashion known as the **Software Development Life Cycle (SDLC)**.
- It is a systematic approach to develop software.

34

Software Development Life Cycle

- SDLC creates a structure for the developer to design, create and deliver high-quality software according to the requirements of the customer.
- It also provides a methodology for improving the quality of the desired product.
- Purpose of the SDLC process is to provide help in producing a product that is cost effective and of high quality.

35