❷ *BASICS of PROGRAMMING*

- You write computer instructions in a computer **programming language** such as Visual Basic, C#, C++, or Java.
- The instructions you write using a programming language are called **program code**; when you write instructions, you are **coding the program**.
- Every programming language has rules governing its word usage and punctuation. These rules are called the language's **syntax**. Mistakes in a language's usage are **syntax errors**. Unless the syntax is perfect, the computer cannot interpret the programming language instruction at all.
- After a computer program is typed using programming language statements and stored in memory, it must be translated to **machine language.** Machine language is also called **binary language**, and is represented as a series of **0**s and **1**s.
  - Your programming language statements are called **source code**, and the translated machine language statements are **object code**.
  - Each programming language uses a piece of software, called a **compiler** or an **interpreter**, to translate your source code into machine language.
  - Although there are differences in how compilers and interpreters work, their basic function is the same—to translate your programming statements into code the computer can use. When you use a **compiler**, an entire program is translated before it can execute; when you use an interpreter, each instruction is translated just prior to execution. Usually, you do not choose which type of translation to use—it depends on the programming language.
  - After a program's source code is successfully translated to machine language, the computer can carry out the program instructions. When instructions are carried out, a program **runs**, or **executes**. In a typical program, some input will be accepted, some processing will occur, and results will be output.
  - With all programming languages, each instruction must be translated to machine language before it can execute.

TYPES of PROGRAMMING LANGUAGES

- **LOW LEVEL LANGUAGES** (Machine Code, Assembly Language, etc):
  - The higher the level, the more abstraction from the hardware. If a language has higher abstraction – it is further away from machine language (1's and 0's)
  - Low level languages have almost no **abstraction** from the hardware.
  - Two types of low level languages are:
    - **Machine Code**
    - **Assembly Language**
  - **Machine code** is understood directly by the CPU. An example is below:

8B542408 83FA0077 06B80000 0000C383 FA027706 B8010000 00C353BB 01000000 B9010000 008D0419 83FA0376 078BD98B C84AEBF1 5BC3

Obviously, it takes specialized knowledge to program in machine code.

- **Assembly Language :** One level of abstraction from machine code. The program given above can be written in assembly language as:

```
fib:
    mov edx, [esp+8]
    cmp edx, 0
    ja @f
    mov eax, 0
    ret

    @@:
    cmp edx, 2
    ja @f
    mov eax, 1
    ret

    @@:
    push ebx
    mov ebx, 1
    mov ecx, 1

    @@:
        lea eax, [ebx+ecx]
        cmp edx, 3
        jbe @f
        mov ebx, ecx
        mov ecx, eax
        dec edx
    jmp @b

    @@:
    pop ebx
    ret
```

- **HIGH LEVEL LANGUAGES** (C, Visual Basic, Pascal, Fortran, Phyton, etc):
  - **A high level language** provides strong abstraction from the hardware.
  - This allows a program to be written in a language that can run on multiple types of computers (running the same operating system).
  - **Examples** of high level languages are:
    - **Visual Basic:**
      - Basic is an old language that has been updated over the years and adapted by Microsoft for use for writing Microsoft Windows and Web applications.
      - Basic first appeared in 1964 and was designed by John George Kemeny and - Thomas Eugene Kurtz at Dartmouth University.
      - The current version of Visual Basic is the 9th version from Microsoft. (Visual Basic 2010)
      - Microsoft first released VB in 1991. This moved the BASIC language to an event driven and **object-oriented programming** (OOP) language.
    - **C:**
      - C is a procedural programming language.
      - It was initially developed by Dennis Ritchie in the year 1972. It was mainly developed as a system programming language to write an operating system.
      - The main features of C language include low-level access to memory, a simple set of keywords, and clean style, these features make C language suitable for system programmings like an operating system or compiler development.

- Many later languages have borrowed syntax/features directly or indirectly from C language.
- Like syntax of Java, PHP, JavaScript, and many other languages are mainly based on C language.
- C++ is nearly a superset of C language (There are few programs that may compile in C, but not in C++).

- **C#**
- **C++**
- **Java**
- **Fortran**
- **Pascal**
- **Python**…etc

SOME HISTORICAL NOTES

- The first computer program was written by Ada Lovelace Byron in 1842.
- Modern programming is said to of started in the 1940s.
- The first "modern" language was Plankalkül which was described in 1943, but not implemented until 1998. It was designed by Konrad Zuse.
- The 1950s and 1960s languages still used today:

  FORTRAN- John Backus et al. (1955)

  LISP- John McCarthy et al.(1958)

  COBOL- Grace Hopper et al. (1959)

  BASIC- 1964 (as noted previously)
- Late 1960s and 1970s. Most of the languages used today were invented or are derived from one of the languages invented in this time period:

  1969- B (forerunner to C)

  1970- Pascal (Java borrows from Pascal)

  1972- C (C++, Java, C#, and many others are based on C)

  1973- ML (F# is based on ML, C++ borrows from ML too)

  1978- SQL (databases)
- The Internet Age 1990s. During the early/mid 1990s many Internet languages were developed:

  1991-Python

  1995- Java

  1995- Javascript (not related to Java)

  1995- PHP

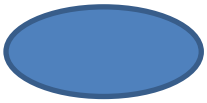
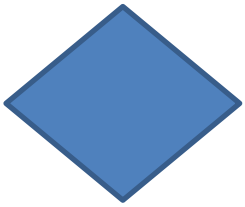
  1995- Delphi (Object Pascal)
- The main change during the evolution of languages is more abstraction as described above. Early programs were bound to specific hardware- current programs are not.

ALGORITHMS and FLOWCHARTS

- An **algorithm** is a plan, a logical step-by-step process for solving a problem. Algorithms are normally written as a **flowchart** or in **pseudocode**.
- Understanding the problem

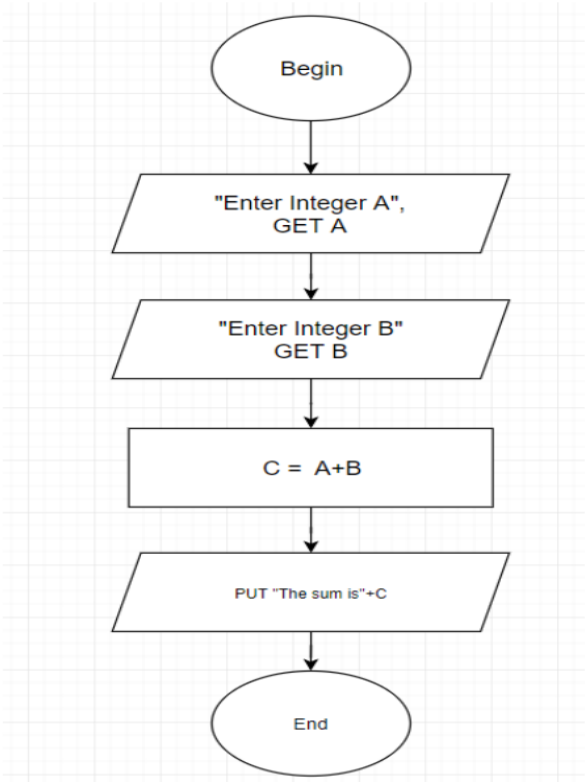
  Before an algorithm can be designed, it is important to check that the problem is completely understood. There are a number of basic things to know in order to really understand the problem: What are the inputs into the problem? What will be the outputs of the problem? In what order do instructions need to be carried out? What decisions need to be made in the problem? Are any areas of the problem repeated?
- **Pseudocode**: Most programs are developed using programming languages. These languages have specific syntax that must be used so that the program will run properly. Pseudocode is not a programming language, it is a simple way of describing a set of instructions that does not have to use specific syntax.
- A **flowchart** is a diagram that represents a set of instructions. **Flowcharts normally use standard symbols to represent the different types of instructions**. These symbols are used to construct the flowchart and show the step-by-step solution to the problem

Some common flowchart symbols:

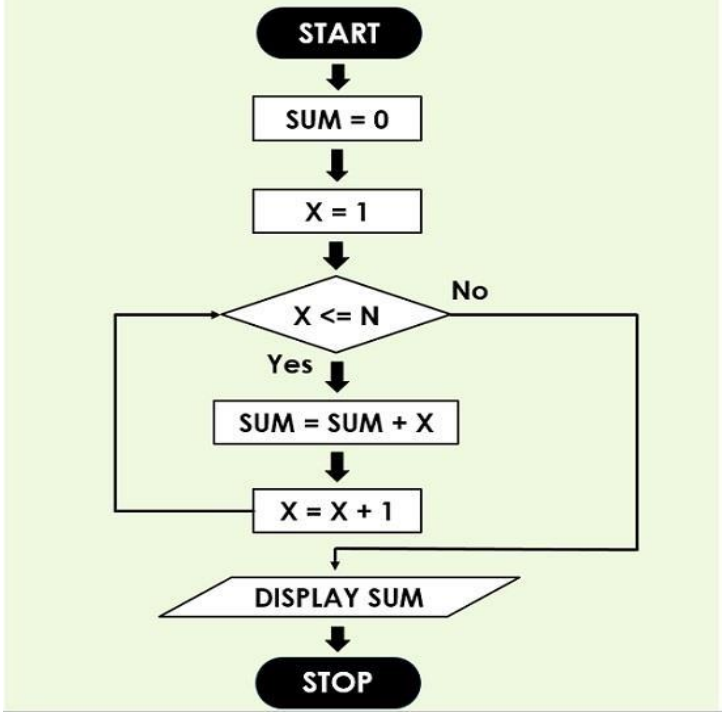
Name	Symbol	Usage
Start or Stop	Oval 	Indicates the starting or ending of the program or the process
Input/Output	Parallelogram 	Used for any Input/Output (I/O) operation. Indicates that the program is to obtain data or output results
Decision	Diamond 	Indicates that the program will decide on one of the two alternative paths
Process	Rectangle 	Used to indicate value assignment and arithmetic operations
Direction of flow	Arrow 	Connects the symbols. The arrow shows the direction of flow of the instructions

- **Simple Flowchart Examples:**

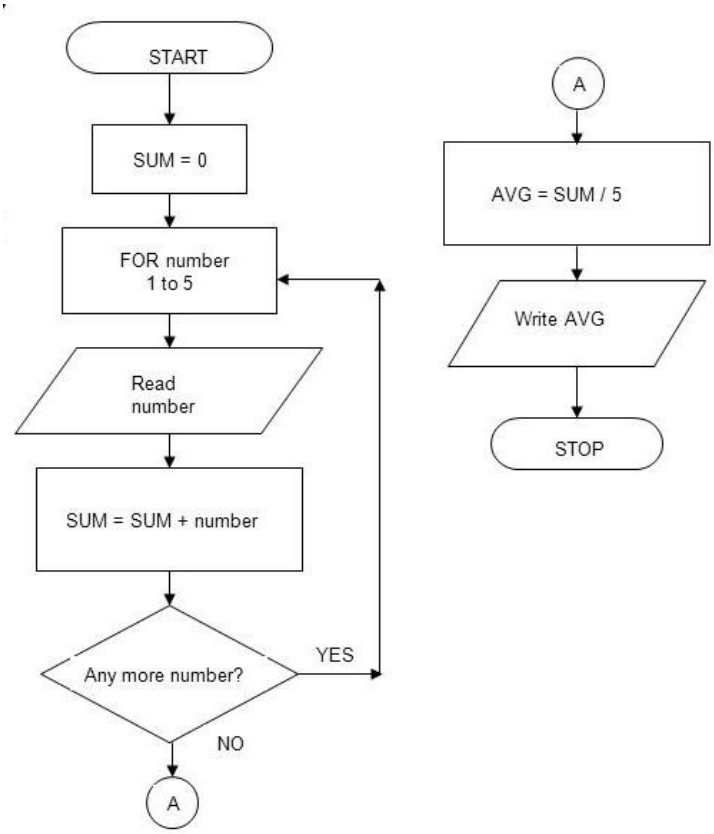
Example 1 - Flowchart for addition of two numbers:



Example 2 - Flowchart for adding integers from 1 to N:



Example 3 - Flowchart for finding the average of 5 numbers:



Example 4 - Flowchart for calculating final score using exam scores and deciding whether the student passes or fails: