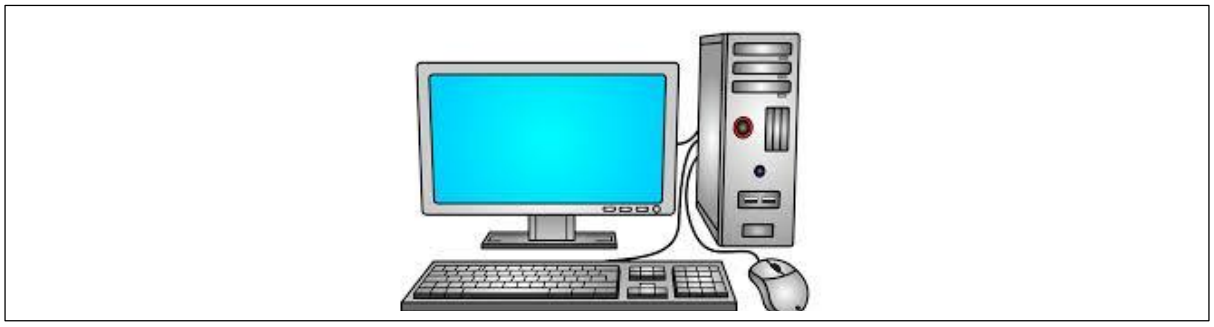
Lecture One: Introduction to Programming Language

Computer in its simplest form is a fast electronic calculating machine that accepts digitized information from the user, processes it according to a sequence of instructions stored in the internal storage, and provides the processed information to the user.



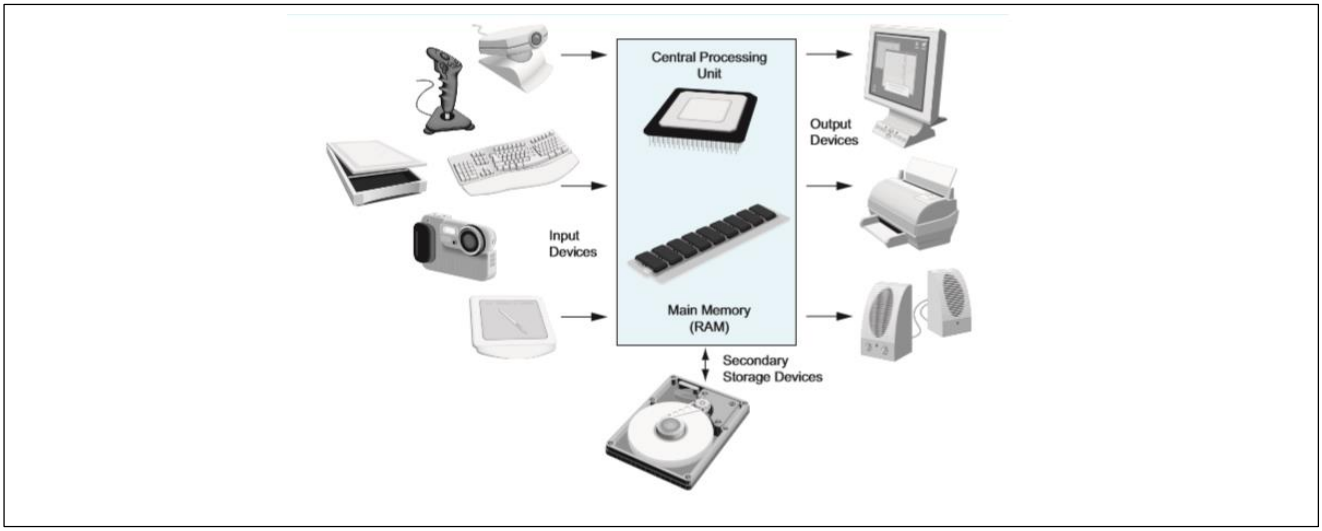
Computer process data under the control of instruction called **computer program**.

The uses of computers are almost limitless in our everyday lives. Computers can do such a wide variety of things because they can be programmed. This means that computers are not designed to do just one job, but to do any job that their programme tell them to do.

Program is a set of instructions that a computer follows to perform a task. Programs are commonly referred to as software.

1-Hardware and Software

The physical devices that a computer is made of are referred to as the computer's hardware. The programs that run on a computer are referred to as software. A computer is not one single device, but a system of devices that all work together.



There are two general categories of software: **system software** and **application software**. Most computer programs clearly fit into one of these two categories.

A-System Software

The programs that control and manage the basic operations of a computer are generally referred to as ***system software***.

B-Application Software

Programs that make a computer useful for everyday tasks are known as application software. An examples of a commonly used applications software are:

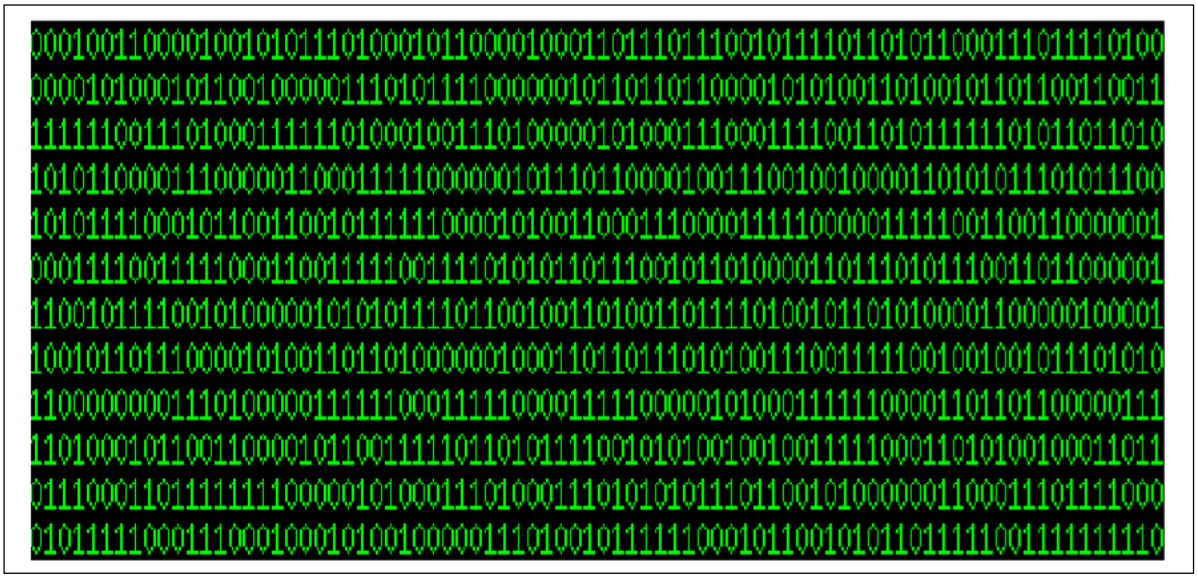
1. Microsoft word
2. Adobe Photoshop
3. Spreadsheet
4. Email programs
5. Web browsers
6. Game programs ……..etc.

2-How a Program Works

Computers can only understand instructions that are written in machine language. Because people find it very difficult to write entire programs in machine language, other programming languages have been invented. The CPU is the most important component in a computer because it is the part of the computer that runs programs. The CPU is designed to perform operations such as reading data from main memory, adding, subtracting, multiplying, and dividing two numbers, moving data from one memory location to another, and determining whether one value is equal to another value.

It can be seen that CPUs perform simple operations on data. CPU does nothing on its own, however. It has to be told what to do, and that's the purpose of a program.

A program is a list of instructions that cause the CPU to perform operations. Here's an example of an instruction that might appear in a program: ***10110000***.

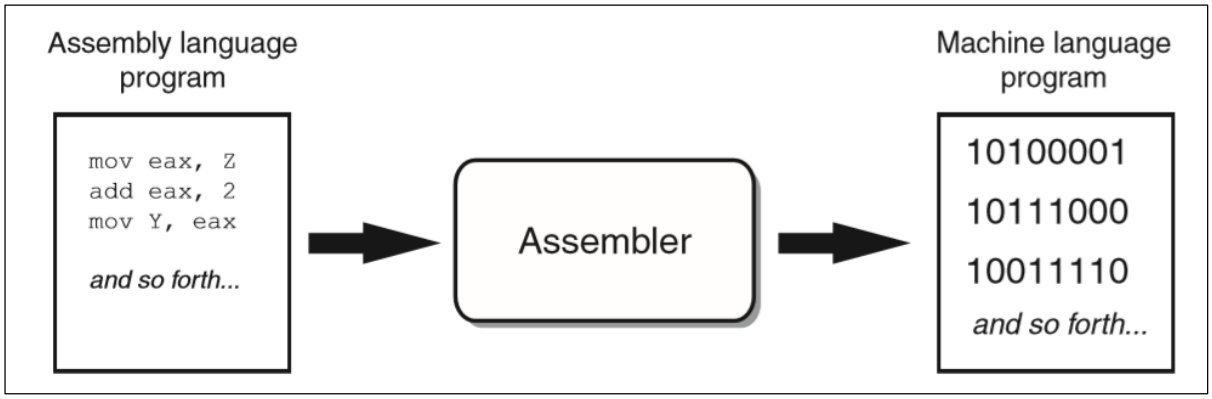


To you and me, this is only a series of 0s and 1s. To a CPU, however, this is an instruction to perform and operation. It is written in 0s and 1s because CPUs only understand instruction that are written in machine language, and machine language instructions always have an underlying binary structure.

Program can have thousands or even millions of binary instructions and writing such a program would be very tedious and time consuming. Programming in machine language would also be very difficult because putting a 0 or a 1 in the wrong place will cause an error. Hence, it is impractical for people to write programs in machine language. For this reason, assembly language was created in early days of computing as an alternative to machine language. Instead of binary numbers for instructions, assembly language use short words that are known as mnemonics.

For example, in assembly language, mnemonic **add** typically means to add numbers. Assembly language programs cannot be executed by the CPU, however. The CPU only understands machine language, so a special program known as an assembler is used to translate an assembly

language program to a machine language program.

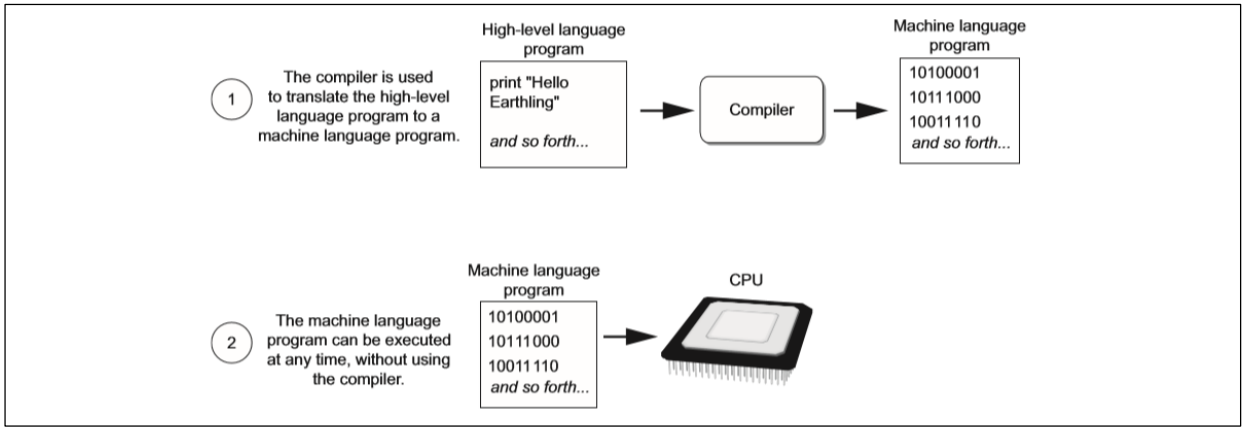


Although assembly language makes it unnecessary to write binary machine language instructions, it not without difficulties. Assembly language is primarily a direct substitute for machine language, and like machine language, it requires that you write a large number of Instruction for even the simplest program. Because assembly language is so close in nature to machine language, it referred to as ***low-level language***.

3-Compilers and Interprets

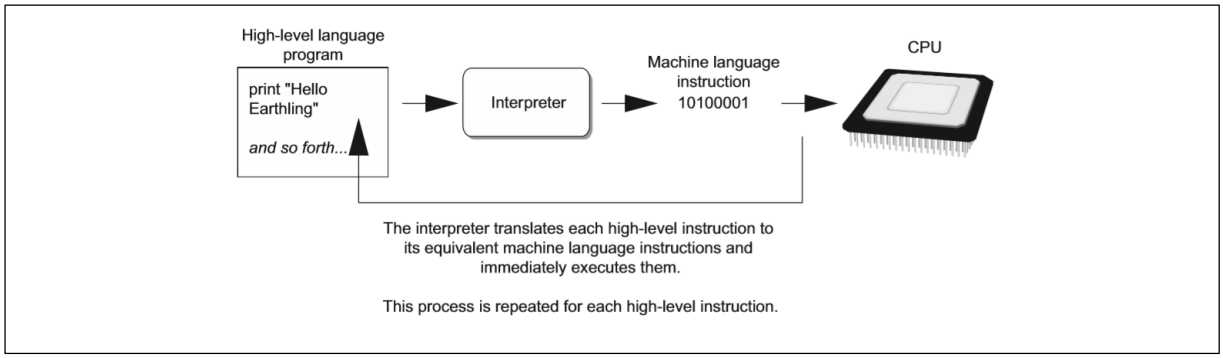
Because the CPU understands only machine language instructions, programs that are written in a high-level language must be translated into machine language. Depending on the language that a program has been written in, the programmer will use either a compiler or an **interpreter** to make the translation.

A compiler is a program that translates a high-level language program into a separate machine language program. The machine language program can then be executed at any time it is needed.



An interpreter is a program that both translates and executes the instructions in high-level language program. As the interpreter reads each individual instruction in the program, it

converts it to machine language instructions and then immediately executes them.

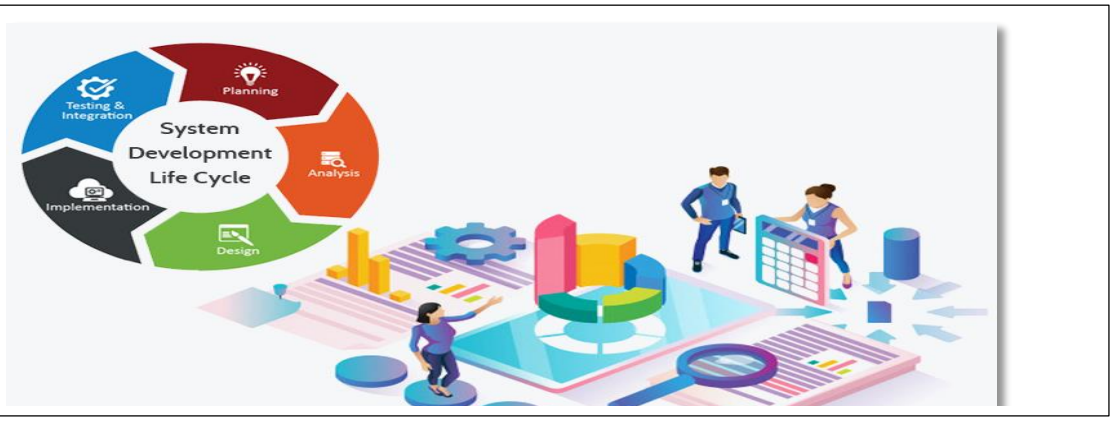


4-Integrated Development Environment (IDE)

An integrated development environment (IDE) is a software application that helps programmers develop software code efficiently. It increases developer productivity by combining capabilities such as software editing, building, testing, and packaging in an easy-to-use application. Just as writers use text editors and accountants use spreadsheets, software developers use IDEs to make their job easier. The IDE integrates all these things and more into one package so that you can edit, compile, run, and debug your code all in one program. Some IDEs, such as IntelliJ IDEA, Eclipse, and Lazarus, contain the necessary compiler, interpreter, or both; others, such as SharpDevelop and NetBeans, do not.

5-Systems Development Life Cycle (SDLC)

A systems development life cycle is composed of distinct work phases that are used by systems engineers and systems developers to deliver information systems. Like anything that is manufactured on an assembly line, an SDLC aims to produce high-quality systems that meet or exceed expectations, based on requirements, by delivering systems within scheduled time frames and cost estimates.



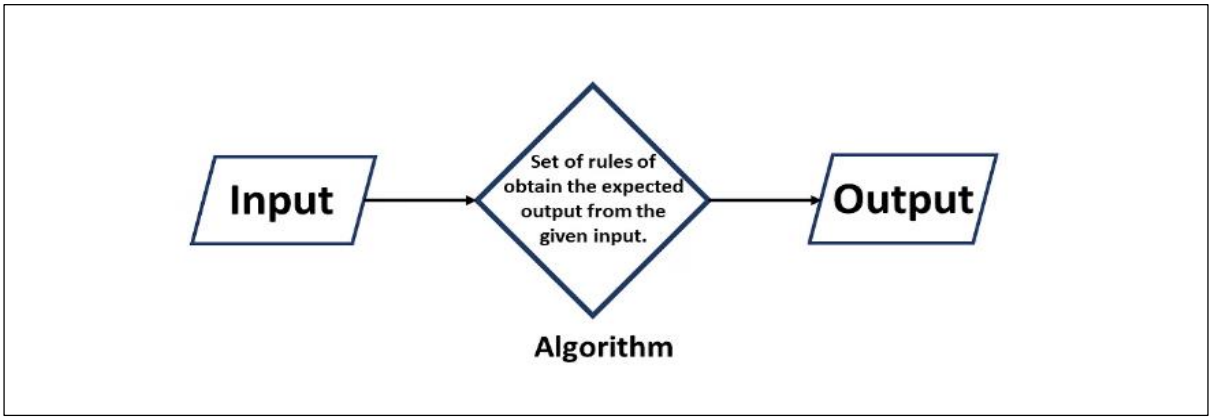
The major steps in creating an application include the following:

- **Initiation:** This stage involves preparing a formal project request to initiate all system development and integration activities including project objectives, users of the system as well as key time frames for project completion.
- **Analysis:** This stage primarily requires development teams to perform a feasibility study to determine whether the project request should be approved for development.

- **Rrequirements specifications:** In this stage, teams need to develop business, operational, and functional requirements specification documents to ensure that all the Requirements are clearly understood. Security teams make sure all security concerns are properly addressed and reflected in the requirements document.
- **Design:** This stage begins with developing detailed design specifications that translate all the functional specs into logical and physical design.
- **Development:** This is where the actual coding of the project begins. All the features of the system are developed in this stage. Extra emphasis is laid on configuring and enabling security features. Information security teams perform source code reviews for critical aspects of the system, including user authentication, and authorization.
- **Testing:** This stage requires developers to use automated testing tools to carry out unit as well as system testing. Security testing is an important aspect in this stage; teams need to continuously review the security tests being written and validate the results – so no security vulnerabilities go unnoticed.
- **Evaluation:** The review stage involves taking feedback from users on the overall effectiveness of the project and ensuring that the system is operating at a satisfactory level.

6-What is Algorithm?

An algorithm is a set of commands that must be followed for a computer to perform calculations or other problem-solving operations. According to its formal definition, an algorithm is a finite set of instructions carried out in a specific order to perform a particular task.



7-How do Algorithms Work?

Algorithms are step-by-step procedures designed to solve specific problems and perform tasks efficiently in the realm of computer science and mathematics. These powerful sets of instructions form the backbone of modern technology and govern everything from web searches to artificial intelligence. Here's how algorithms work:

- Input: Algorithms take input data, which can be in various formats, such as numbers, text, or images.
- Processing: The algorithm processes the input data through a series of logical and mathematical operations, manipulating and transforming it as needed.
- Output: After the processing is complete, the algorithm produces an output, which could be a result, a decision, or some other meaningful information.
- Efficiency: A key aspect of algorithms is their efficiency, aiming to accomplish tasks quickly and with minimal resources.
- Optimization: Algorithm designers constantly seek ways to optimize their algorithms, making them faster and more reliable.
- Implementation: Algorithms are implemented in various programming languages, enabling computers to execute them and produce desired outcomes.

Algorithm Example:

Example 1: Develop an algorithm that inputs a Two of number and output their Average.

A computer algorithm can only carry out simple instruction like

- Read a number
- Add a number to another number
- Output Number

Sol:

Step1:Start

Step2:Read a and b

Step3:avg=(a+b)/2

Step4:print avg

Step5:output

Example 2: Develop an algorithm that allows the user to enter the count of numbers in a list followed by these numbers.

The algorithm should find and output the minimum and the maximum numbers in the list. An algorithm for this might be:

- Initialize.
- Get count of numbers.
- Enter numbers and find maximum and minimum.
- Output result.

The user might enter zero for the count. To deal with this case the above general case can be extended as follows to be an algorithm:

1. Initialize the require variables.
2. Get count of numbers.

3. If count is zero then exit.

4. Otherwise begin.

5. Enter numbers.

6. Find max and min.

7. Output result.

8. End.

Algorithm it is not the entire program or code; it is simple logic to a problem represented as an informal description in the form of a **pseudocode or flowchart**.

A. **Pseudocode:** Pseudocode is one method of designing or planning a program. It uses English statement to describe what a program is to accomplish. Pseudocode is used for documenting the program or module design (also known as the algorithm). The following example outline of a simple program illustrates pseudocode, we want to enter the ages of two people and have the computer calculate their average and display the answer.

```
Input
        Display a message asking the user to enter the first age
        Get the first age from the keyboard
        Display a message asking the user to enter the second age
        Get the second age from the keyboard
Processing
        Calculate the answer by adding two ages together and dividing by two
Output
        Display the answer on the screen
        Pause so the user can see the answer
```