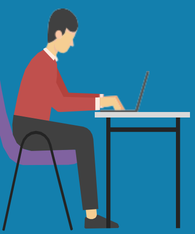
MALLA REDDY
UNIVERSITY

PYTHON PROGRAMMING

UNIT- I

INTRODUCTION TO COMPUTING



Contents
Introduction to Computing

Computer Systems

Computing Environment

Computer Languages

Algorithms and Flowcharts

Steps for Creating and Running programs

COMPUTER SYSTEMS

A Computer is an electronic device that stores, retrieves and manipulates the data. We can also refer computer as a fast electronic device having communication capabilities and vast memory capacity. It processes data to produce information. A System is a group of several objects within a process.

For Example: Educational System involves teachers and students (objects). A Teacher teaches subject to students and Teaching can be considered a process. Similarly, a computer system can also have objects and processes.

The following are the objects of computer System

1) User (A person who uses the computer)
2) Hardware
3) Software

Computer is made up of two major components called "hardware" and "software".

Hardware: All physical components of computer come under hardware like all input and output devices and CPU etc. (that are visible, that can be touched and felt).

The hardware of a computer system can be classified as

1) Input Devices(Input Unit)
2) Output Devices (Output Unit)
3) Central Processing Unit (CPU)

Input Unit: It provides man to machine communication. It accepts data (**Input**) from the user with the help of input devices in human readable form converts it into machine-readable form and sends it to CPU. Key Board and mouse are commonly used input devices. Other in-put devices are punched cards, optical scanners, joy stick etc.

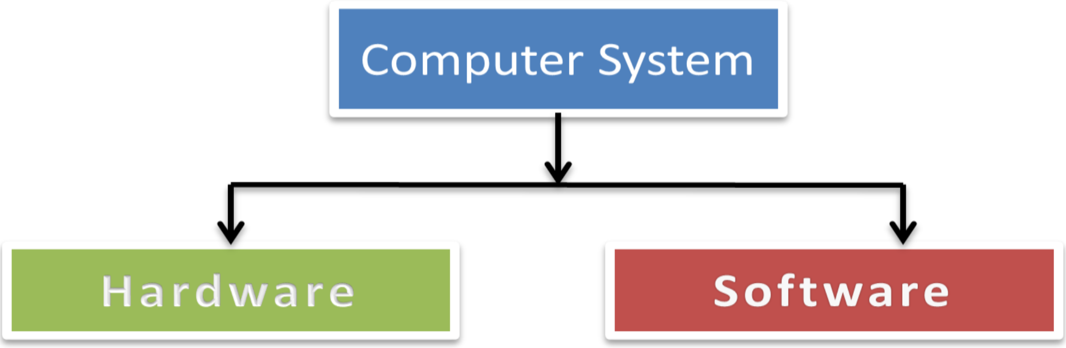


Figure 1.1: Computer System

Output Unit: It provides machine to man communication. It receives the data (**Output**) from CPU in machine readable form, and presents it to human readable form. Visual Display Unit (VDU), Monitor, Printers are most commonly used output devices.

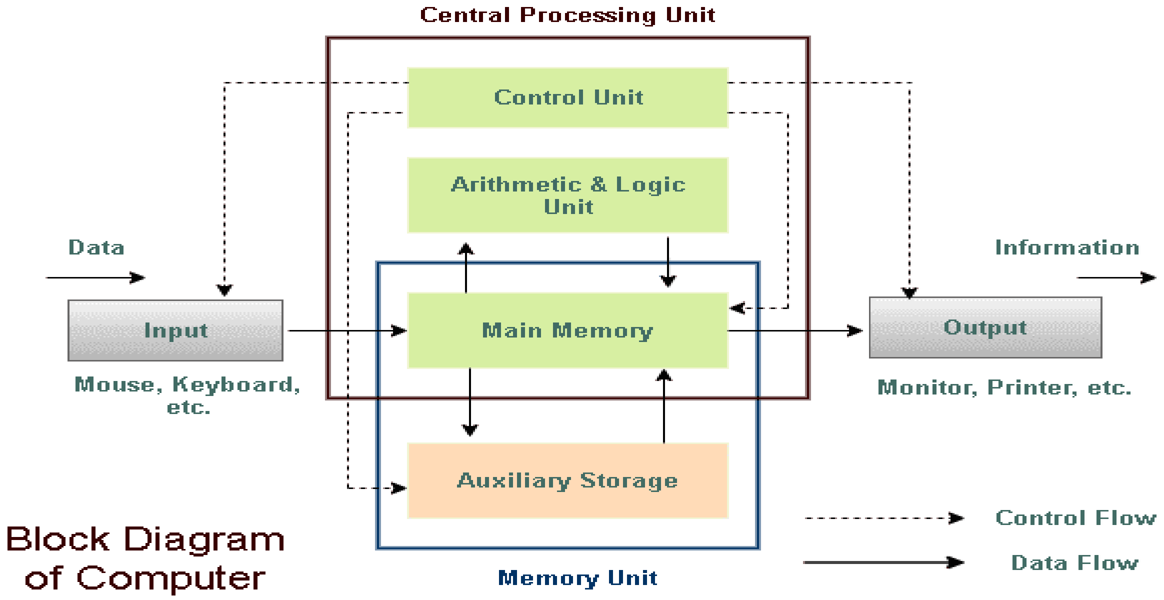


Figure 1.2: Block Diagram of Computer System

If the output is shown on the monitor, we say that we have a **soft copy**. If it is printed by the printer, we say that we have a **hard copy.**

CPU: The CPU Controls the sequence of operations based on stored instructions and it passes the commands to all the parts of the computer for process the data and sends results to output devices. The CPU consists of three major parts

a. **Control Unit:** Every Operation such as storing, computing and retrieving the data should be governed by the control unit.

b. **ALU:** It performs the Arithmetic and Logical Operations such as

   i. **+,-,*,/** (Arithmetic Operators) &&, ||! (Logical Operators)

c. **Memory Unit:** The Memory unit is used for storing the data. The Memory unit is classified into two

   1. Primary Memory or Main Memory or Immediate access memory
   2. Secondary Memory or Auxiliary Memory

1. **Primary memory:** It is part of CPU and is used to store the primary data when the

   A. **Read Only Memory (ROM):** It represents Read Only Memory that stores data and instructions even when the computer is turned off. The Contents in the ROM can't be modified once if they are written. It is used to store the BIOS information.

   B. **Random Access Memory (RAM):** It represents Random Access Memory that stores data and instructions when the computer is turned on. The contents in the RAM can be modified any no. of times by instructions. It is used to store the programs under execution.

C. **Cache Memory:** It is used to store the data and instructions referred by processor.

2. **Secondary Memory:** Which is external to the CPU? The following are the different kinds of memories which are treated as Secondary.

   A. **Magnetic Storage:** The Magnetic Storage devices store information that can be read, erased and rewritten a number of times.

   Example: Floppy Disks, Hard Disks, Magnetic Tapes

   B. **Optical Storage:** The optical storage devices that use laser beams to read and write stored data.

   Example: CD (Compact Disk), DVD (Digital Versatile Disk)

Computer Software: The software is collection of programs that allows the hardware to do its job. Computer software is divided into two broad categories such as.

1. **System Software:** The System software manages the computer resources. It provides the interface between the hardware and the users. It is divided into three classes: the operating system programs, System support programs, and System development programs.

   A. Operating system software: It provides services such as a user interface, file and database access, and interfaces to communication systems such as Internet protocols. The primary purpose of this software is to keep the system operating in an efficient manner while allowing the users access to the system.

   B. System support software: It provides system utilities and other operating services. Examples of system utilities are sort programs and disk format programs. Operating services consists of programs that provide performance statistics for the operational staff and security monitors to protect the system and data.

   C. System development software: It includes the language translators that convert programs into machine language for execution, debugging tools to ensure that the programs are error free and computer –assisted software engineering (CASE) systems.

   D. General purpose software: It is purchased from a software developer and can be used for more than one application. Examples of general-purpose software include word processors, database management systems, and computer aided design systems. They are labeled general purpose because they can solve a variety of user computing problems.

2. **Application Software:** The Application software is directly responsible for helping users solve their problems. Application software is broken in to two classes: general-purpose software and application – specific software.

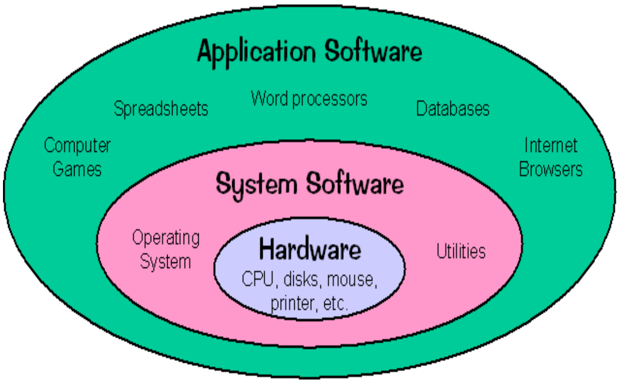


Figure 1.3: Block Diagram for Relationship between System and Application Software

Application Specific software: It can be used for its intended purpose. A general ledger system used by an accountant is an example for application specific software. The relationship between system and application software is shown below. In this figure, each circle represents an interface point. The inner core is hard ware. The user is represented by the out layer. To work with the system, the typical user uses some form of application software. The application software in turn interacts with the operating system, which is a part of the system software layer. The system software provides the direct interaction with the hard ware. The opening at the bottom of the figure is the path followed by the user who interacts directly with the operating system when necessary.

COMPUTING ENVIRONMENTS

The word computer ‘is used to refer to the process of converting information to data. The advent of several new kinds of computers created a need to have different computing environments. The following are the different kinds of computing environments available

1. Personal Computing Environment
2. Time Sharing Environment
3. Client/Server Environment
4. Distributed Computing Environment

1. Personal Computing Environment

In 1971, Marcian E. Hoff, working for INTEL combined the basic elements of the central processing unit into the microprocessor. If we are using a personal computer then all the computer hardware components are tied together. This kind of computing is used to satisfy the needs of a single user, who uses the computer for the personal tasks.

Example: Personal Computer

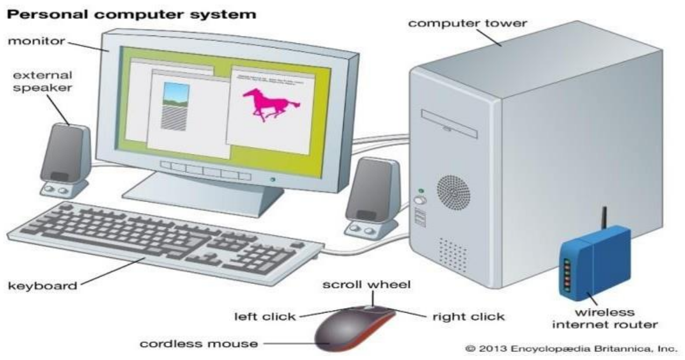


Figure 1.4: Personal Computing Environment

2. Time-Sharing Environment

The concept of time-sharing computing is to share the processing of the computer basing on the criteria time. In this environment all the computing must be done by the central computer.

The complete processing is done by the central computer. The computers which ask for processing are only dumb terminals.

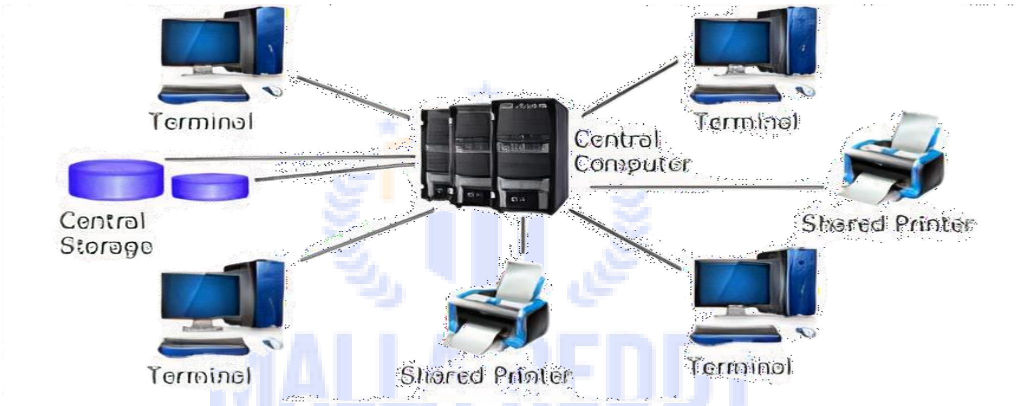


Figure 1.5: Time-Sharing Environment

3. Client/Server Environment

A Client/Server Computing involves the processing between two machines. A client Machine is the one which requests processing. Server Machine is the one which offers the processing. Hence the client is Capable enough to do processing. A portion of processing is done by client and the core(important) processing is done by Server.

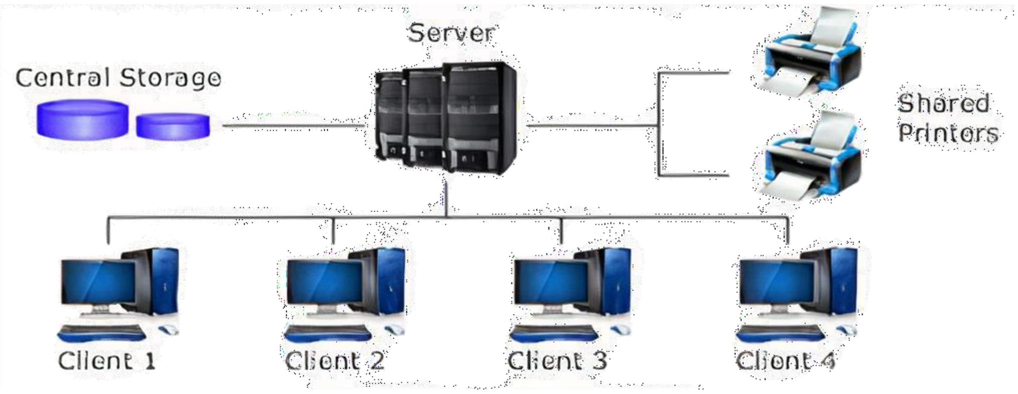


Figure 1.6: Client/Server Environment

4. Distributed Computing

In distributed computing environment the computing complexity will be distributed among multiple servers and clients, which are having connectivity. This environment is reliable and scalable. All the machines Clients/Servers share the processing task.

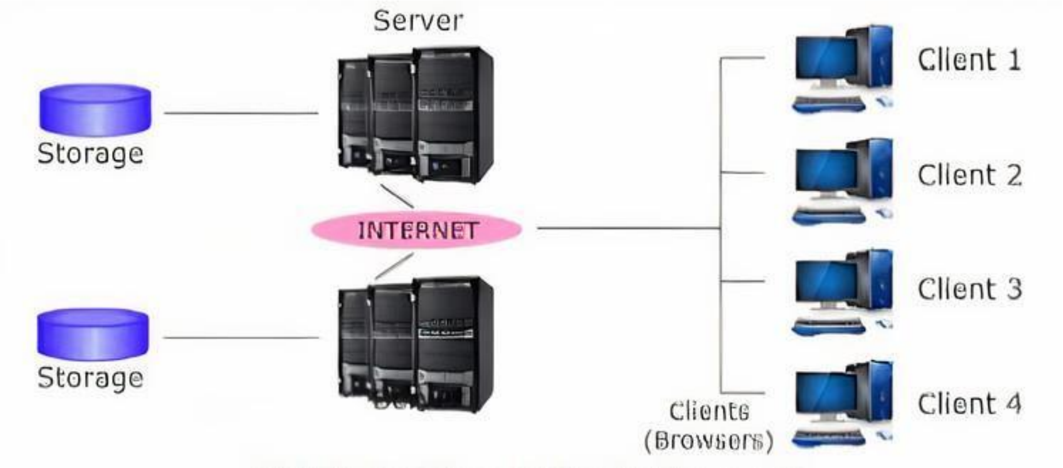


Figure 1.7: Distributed Computing Environment

COMPUTER LANGUAGES

To write a program (tells what to do) for a computer, we must use a computer language.

Generally humans are communicating to the computer with the help of programming languages, these languages are mainly divided into two types such as

1. Low level language
2. High level language

The low level language is again subdivided in to two types such as

1. Machine level Language (Binary level language)
2. Assembly level language (Symbolic language)

Over the year's computer languages have evolved from machine languages to natural languages. The following is the summary of computer languages

1940's	**--**	**Machine level languages**
1950's	**--**	**Symbolic Languages**
1960's	**--**	**High Level Languages**

Machine level Language

In the earliest days of computers, the only programming languages available were machine languages. Each computer has its own machine language which is made of streams of 0's and 1's. The instructions in machine language must be in streams of 0's and 1's. This is also referred as binary digits. These are so named as the machine can directly understood the programs and Machine level language is machine dependent

Advantages:

1) High speed execution
2) The computer can understand instructions immediately
3) No translation is needed.

Disadvantages:

1) Machine dependent
2) Programming is very difficult
3) Difficult to understand
4) Difficult to write bug free programs
5) Difficult to isolate an error

Example Addition of two numbers

```
          2             ⍰ 0 0 1 0
    +     3             ⍰ 0 0 1 1
         ---            -------------
          5      ⍰        0 1 0 1
         ----           -------------
```

Symbolic Languages (or) Assembly Language:

In the early 1950‘s Admiral Grace Hopper, a mathematician and naval officer, developed the concept of a special computer program that would convert programs into machine language. These early programming languages simply mirrored the machine languages using symbols or mnemonics to represent the various language instructions. These languages were known as symbolic languages. Because a computer does not understand symbolic language it must be translated into the machine language. A special program called an **Assembler** translates symbolic code into the machine language. Hence, they are called as Assembly language.

Advantages:

1) Easy to understand and use
2) Easy to modify and isolate error
3) High efficiency
4) More control on hardware

Disadvantages:

1) Machine Dependent Language
2) Requires translator
3) Difficult to learn and write programs
4) Slow development time
5) Less efficient

Example:

```
          2                    PUSH2,A
          3                    PUSH3,B
            +                 ADDA,B
      =============
          5                   PRINT B
```

High-Level Languages

The symbolic languages greatly improved programming efficiency they still required programmers to concentrate on the hardware that they were using working with symbolic languages was also very tedious because each machine instruction had to be individually coded. The desire to improve programmer efficiency and to change the focus from the computer to the problems being solved led to the development of high-level languages.

High-level languages are portable to many different computers allowing the programmer to concentrate on the application problem at hand rather than the intricacies of the computer.

C	A systems implementation Language
C++	C with object-oriented enhancements
JAVA	Object oriented language for internet and general applications using basic C syntax

Advantages:

1) Easy to write and understand
2) Easy to isolate an error
3) Machine independent language
4) Easy to maintain
5) Better readability
6) Low Development cost
7) Easier to document
8) Portable

Disadvantages:

1) Needs translator
2) Requires high execution time
3) Poor control on hardware
4) Less efficient

Example: C language

```
#include<stdio.h>

void main()
{
int a,b,c;
 scanf("%d%d%",&a,&b);
c=a+b;
printf("%d",c);
}
```

Difference between Machine, Assembly, High Level Languages

Feature	Machine	Assembly	High Level
Form	0‘s and 1‘s	Mnemonic codes	Normal English
Machine Dependent	Dependent	Dependent	Independent
Translator	Not Needed	Needed (Assembler)	Needed (Compiler)
Execution Time	Less	Less	High
Languages	Only one	Different Manufacturers	Different Languages
Nature	Difficult	Difficult	Easy
Memory Space	Less	Less	More

Language Translators

These are the programs which are used for converting the programs in one language into machine language instructions, so that they can be executed by the computer.

1) **Compiler:** It is a program which is used to convert the high-level language programs into machine language
2) **Assembler:** It is a program which is used to convert the assembly level language programs into machine language
3) **Interpreter**: It is a program, it takes one statement of a high-level language program, translates it into machine language instruction and then immediately executes the resulting machine language instruction and soon.

Comparison between a Compiler and Interpreter

COMPILER	INTERPRETER
A Compiler is used to compile an entire program and an executable program is generated through the object program	An interpreter is used to translate each line of the program code immediately as it is entered
The executable program is stored in a disk for future use or to run it in another computer	The executable program is generated in RAM and the interpreter is required for each run of the program
The compiled programs run faster	The Interpreted programs run slower
Most of the Languages use compiler	A very few languages use interpreters.

CREATING AND RUNNING PROGRAMS

The procedure for running a program written in C into machine Language. The process is presented in a straightforward, linear fashion but you should recognize that these steps are repeated many times during development to correct errors and make improvements to the code.

It is the job of the programmer to write and test the program. There are four steps in this process.

1) Writing and Editing the program
2) Compiling the program
3) Linking the program with the required modules
4) Executing the program

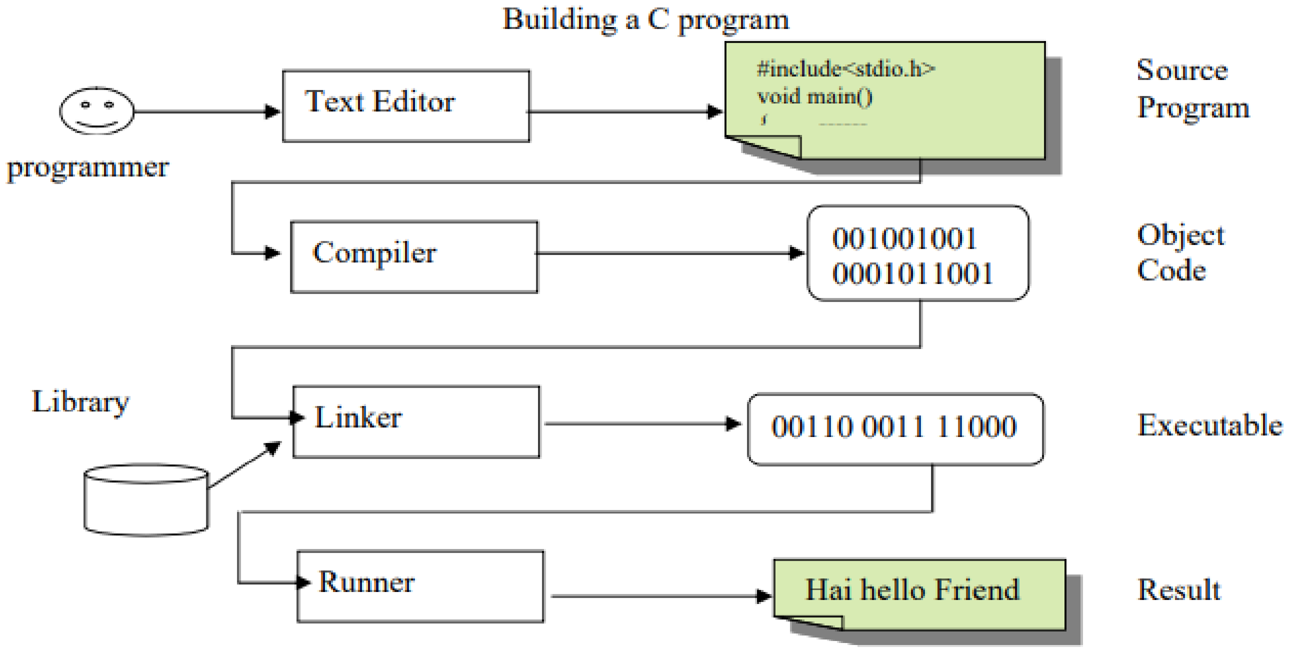


Figure 1.8: Steps for Creating and Running Programs

Table: List of Phases with Tools and file extension

S. No.	Phase	Name of Code	Tools	File Extension
1	Text Editor	Source Code	C Compilers Edit, Notepad etc.	.C
2	Compiler	Object Code	C Compiler	.OBJ
3	Linker	Executable Code	C Compiler	.EXE
4	Runner	Executable Code	C Compiler	.EXE

Writing and Editing Programs

The software used to write programs is known as a text editor. A text editor helps us enter, change and store character data. Once we write the program in the text editor we save it using a filename stored with an extension of.C. This file is referred as source code file.

Compiling Programs

The code in a source file stored on the disk must be translated into machine language. This is the job of the compiler. The Compiler is a computer program that translates the source code written in a high-level language into the corresponding object code of the low-level language. This translation process is called compilation. The entire high-level program is converted into the executable machine code file. The Compiler which executes C programs is called as C Compiler. Example Turbo C, Borland C, GC etc.,

The C Compiler is actually two separate programs:

1. **ThePreprocessor**
2. **The Translator**

The Preprocessor reads the source code and prepares it for the translator. While preparing the code, it scans for special instructions known as preprocessor commands. These commands tell the preprocessor to look for special code libraries. The result of preprocessing is called the translation unit.

After the preprocessor has prepared the code for compilation, the translator does the actual work of converting the program into machine language. The translator reads the translation unit and writes the resulting object module to a file that can then be combined with other precompiled units to form the final program. An object module is the code in the machine language.

Linking Programs

The Linker assembles all functions, the program‘s functions and system‘s functions into one executable program.

Executing Programs

To execute a program, we use an operating system command, such as run, to load the program into primary memory and execute it. Getting the program into memory is the function ofanoperatingsystemprogramknownastheloader.Itlocatestheexecutableprogramandreads it into memory. When everything is loaded the program takes control and it begin execution.

ALGORITHM

Algorithm is a step by step procedure for solving a particular problem or Algorithm is a finite sequence of instructions, each of which has a clear meaning. We represent an algorithm using a pseudo language that is a combination of the constructs of a programming language together with informal English statements.

The ordered set of instructions required to solve a problem is known as an *algorithm*. The characteristics of a good algorithm are:

- **Input –** the algorithm receives input.
- **Output –** the algorithm produces output.
- **Definiteness -** Each instruction must be clear, well defined and precise. There should not be any ambiguity.
- **Finiteness -** It should be a sequence of finite instructions.
- **Effectiveness**- The operations must be simple and must be completed in a finite time.

Algorithm Notation: (Using pseudo code)

- Name of the algorithm: It specifies the problem to be solved
- Step number: Identification number or tag of an instruction it should be a positive value.
- Explanatory Comments: It follows the step number and describes or explains the operation to be performed. It should be written with in a pair of square brackets.
- Termination: It indicates the end of the algorithm. It is generally a STOP statement, and it is the last instruction in the algorithm.

Example:

Q. Write an algorithm to find sum of two numbers?

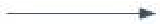
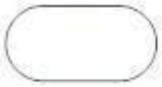
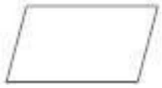
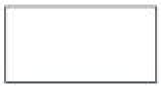
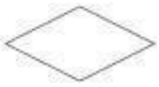
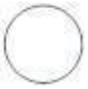
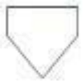
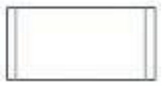
```
Step 1: Start
Step 2: [Read A and B]
        Read a,b
Step 3: [Compute]
        Sum=A+B
Step 4: [Write Sum]
        Print Sum
Step 5: [end] Stop.
```

FLOWCHART

Flowchart is a diagrammatic representation of an algorithm. Flowchart is very helpful in writing program and explaining program to others.Flow chart is constructed by using geometrical figures, which indicates particular operation. While drawing the flow chart, operation must be written inside the geometric figures.

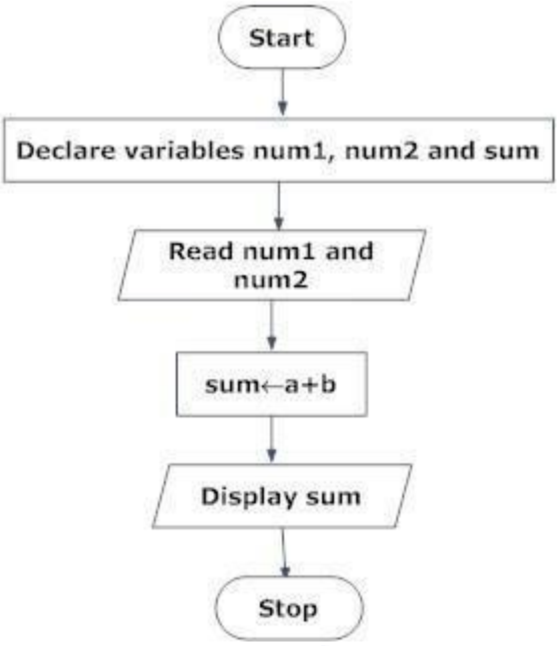
Symbols Used in Flowchart

Different symbols are used for different states in flowchart, for example: Input/Output and decision making has different symbols. The table below describes all the symbols that are used in making flowchart

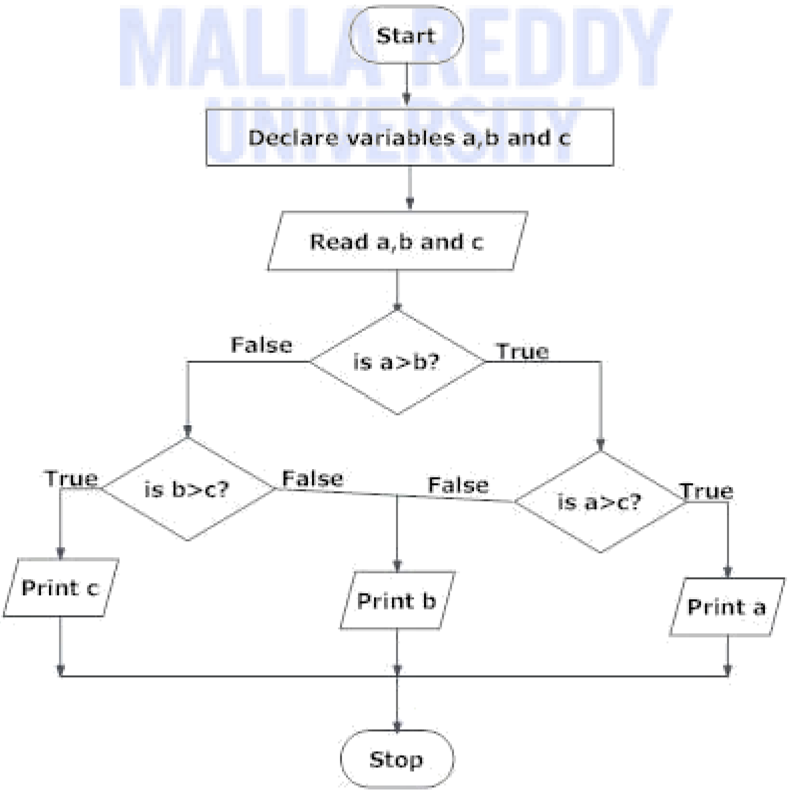
Symbol	Symbol Name	Purpose	Description
	Arrow	Flow line	Used to indicate the flow of logic by connecting symbols.
	Oval	Terminal (Stop/Start)	Used to represent start and end of flowchart.
	Parallelogram	Input/output	Used for input and output operation.
	Rectangle	Processing	Used for arithmetic operations and data- manipulations.
	Rhombus	Decision	Used to represent the operation in which there are two alternatives, true and false.
	Circle	On-page Connector	Used to join different flowline
		Off-page Connector	Used to connect flowchart portion on different page.
		Predefined Process/Function	Used to represent a group of statements performing one processing task.

Examples of flowcharts in programming

Example-1: Draw a flowchart for adding of two numbers.



Example: 2 draw a flowchart for finding the largest among three different numbers.



PROGRAM DEVELOPMENT STEPS

The following are the important steps to consider while writing programs.

Understand the problem: Unless the problem is carefully understood we cannot begin to solve it. Identify the required information that is omitted.

Develop the solution: Once the problem is fully understood, there is a need to develop the solution. Three tools will help in this task.

(i) structure charts (ii) pseudo code (iii) flowchart

The structure chart is used to design the whole program. Pseudo code and flowcharts are used to design individual parts of the program.

Write the program: Write the program by starting with the top box on the structure chart and work out to the bottom. This is known as top-down implementation.

Test the Program: Program testing can be a very time-consuming part of program development. It is the programmer's responsibility to completely test the program.

Black box testing: Testing the program without knowing what is inside it – without Knowing how it works. This is done by looking requirements statement. Assumes that tester knows nothing about the program.

ii) **White box testing:** Each and every This testing assumes that the tester knows everything about the program.

What is a Programming Language?

1. Programming language is a set of written symbols that instructs the computer hardware to perform specific tasks.

2. Typically, a programming language consists of a vocabulary and a set of rules (called syntax) that the programmer must learn".

1st generation of programming languages

Machine language is the only programming language that the computer can understand directly without translation. It is a language made up of entirely 1s and 0s. There is not, however, one universal machine language because the language must be written in accordance with the special characteristics of a given processor.

Each type or family of processor requires its own machine language. For this reason, machine language is said to be machine-dependent (also called hardware-dependent).

In the computer's first generation, programmers had to use machine language because no other option was available. Machine language programs have the advantage of very fast execution speeds and efficient use of primary memory.

Use of machine language is very tedious, difficult and time-consuming method of programming. Machine language is low-level language. Since the programmer must specify every detail of an operation, a low-level language requires that the programmer have detailed knowledge of how the computer works. Programmers had to know a great deal about the computer's design and how it functioned.

To make programming simpler, other easier-to-use programming languages have been developed. These languages, however must ultimately be translated into machine language before the computer can understand and use them.

2nd Generation of programming languages

The first step in making software development easier and more efficient was the creation of Assembly languages. They are also classified as low-level languages because detailed knowledge of hardware is still required.

They were developed in 1950s. Assembly languages use mnemonic operation codes and symbolic addresses in place of 1s and 0s to represent the operation codes. A mnemonic is an alphabetical abbreviation used as memory aid.

This means a programmer can use abbreviation instead of having to remember lengthy binary instruction codes. For example, it is much easier to remember L for Load, A for Add, B for Branch, and C for Compare than the binary equivalents i-e different combinations of 0s and 1s.

Assembly language uses symbolic addressing capabilities that simplify the programming process because the programmer does not need to know or remember the exact storage locations of instructions or data.

Symbolic addressing is the ability to express an address in terms of symbols chosen by the programmer rather than in terms of the absolute numerical location. Therefore, it is not necessary to assign and remember a number that identifies the address of a piece of data.

Although assembly languages represented an improvement, they had obvious limitations. Only computer specialists familiar with the architecture of the computer being used can use them. And because they are also machine dependent, assembly languages are not easily converted to run on other types of computers.

Before they can be used by the computer, assembly languages must be translated into machine language. A language translator program called an assembler does this conversion. Assembly languages provide an easier and more efficient way to program than machine languages while still maintaining control over the internal functions of a computer at the most basic level. The advantages of programming with assembly languages are that they produce programs that are efficient, use less storage, and execute much faster than programs designed using high-level languages.

3rd Generation of programming languages

Third generation languages, also known as high-level languages, are very much like everyday text and mathematical formulas in appearance. They are designed to run on a number of different computers with few or no changes.

Objectives of high-level languages

- To relieve the programmer of the detailed and tedious task of writing programs in machine language and assembly languages.
- To provide programs that can be used on more than one type of machine with very few changes.
- To allow the programmer more time to focus on understanding the user's needs and designing the software required meeting those needs.

Most high-level languages are considered to be procedure-oriented, or Procedural languages, because the program instructions comprise lists of steps, procedures, that tell the computer not only what to do but how to do it.

High-level language statements generate, when translated, a comparatively greater number of assembly language instructions and even more machine language instructions.

The programmer spends less time developing software with a high-level language than with assembly or machine language because fewer instructions have to be created.

A language translator is required to convert a high-level language program into machine language. Two types of language translators are used with high level languages: compilers and interpreters.

4th Generation of programming languages

Fourth generation languages are also known as very high-level languages. They are non-procedural languages, so named because they allow programmers and users to specify what the computer is supposed to do without having to specify how the computer is supposed to do it.

Consequently, fourth generation languages need approximately one tenth the number of statements that a high-level language needs to achieve the same results.

Because they are so much easier to use than third generation languages, fourth generation languages allow users, or non-computer professionals, to develop software.

Objectives of fourth generation languages

1. Increasing the speed of developing programs.
2. Minimizing user effort to obtain information from computer.
3. Decreasing the skill level required of users so that they can concentrate on the application rather than the intricacies of coding, and thus solve their own problems without the aid of a professional programmer.
4. Minimizing maintenance by reducing errors and making programs that are easy to change.
5. Depending on the language, the sophistication of fourth generation languages varies widely. These languages are usually used in conjunction with a database and its data dictionary.

Five basic types of language tools fall into the fourth-generation language category.

1. Query languages
2. Report generators.
3. Applications generators.
4. Some microcomputer application software.

Query languages

Query languages allow the user to ask questions about, or retrieve information from database files by forming requests in normal human language statements (such as English). The difference between the definitions for query language and for database management systems software is so slight that most people consider the definitions to be the same. Query languages do have a specific grammar vocabulary, and syntax that must be mastered, but this is usually a simple task for both users and programmers.

Report generators

Report generators are similar to query languages in that they allow users to ask questions from a database and retrieve information from it for a report (the output); however, in case of a report generator, the user is unable to alter the contents of the database file. And with a report generator, the user has much greater control over what the output will look like. The user of a report generator can specify that the software automatically determine how the output should look or can create his or her own customized output reports using special report generator command instructions.

Application generators

Application generators allow the user to reduce the time it takes to design an entire software application that accepts input, ensures data has been input accurately, performs complex calculations and processing logic, and outputs information in the form of reports. The user must key into computer-useable form the specification for what the program is supposed to do. The resulting file is input to the applications generator, which determine how to perform the tasks and which then produces the necessary instructions for the software program.

5th Generation of programming languages

Natural Languages represent the next step in the development of programming languages, i-e fifth generation languages. The text of a natural language statement very closely resembles human speech. In fact, one could word a statement in several ways perhaps even misspelling some words or changing the order of the words and get the same result.

These languages are also designed to make the computer "smarter". Natural languages already available for microcomputers include Clout, Q&A, and Savvy Retriever (for use with databases) and HAL (Human Access Language).

The use of natural language touches on expert systems, computerized collection of the knowledge of many human experts in a given field, and artificial intelligence, independently smart computer systems.

INTRODUCTION TO C LANGUAGE