CHAPTER - 1
INTRODUCTION TO COMPUTERS

1.1. INTRODUCTION TO COMPUTERS :

Computers have become all pervasive in our day to day life. The advent of computers can be treated as another revolution after industrial revolution. It has led to development of not only computation but also Communication, Internet, Multimedia, Games, Accounting, Artificial Intelligence and virtualisation in all fields of life facilitating non destructive testing, simulation and modelling, mechanization and use of Robots in hazardous fields.

Computers being inanimate and capable of understanding only machine language, consisting of ones and zeros, communication with the computers requires special efforts. *The computer can communicate with outside world only through interfaces between the machine and the person.* **The computer consists of main Central Processing Unit, Memory, Input devices and Output devices.** The basic block diagram of a computer is given in fig. 1.1.

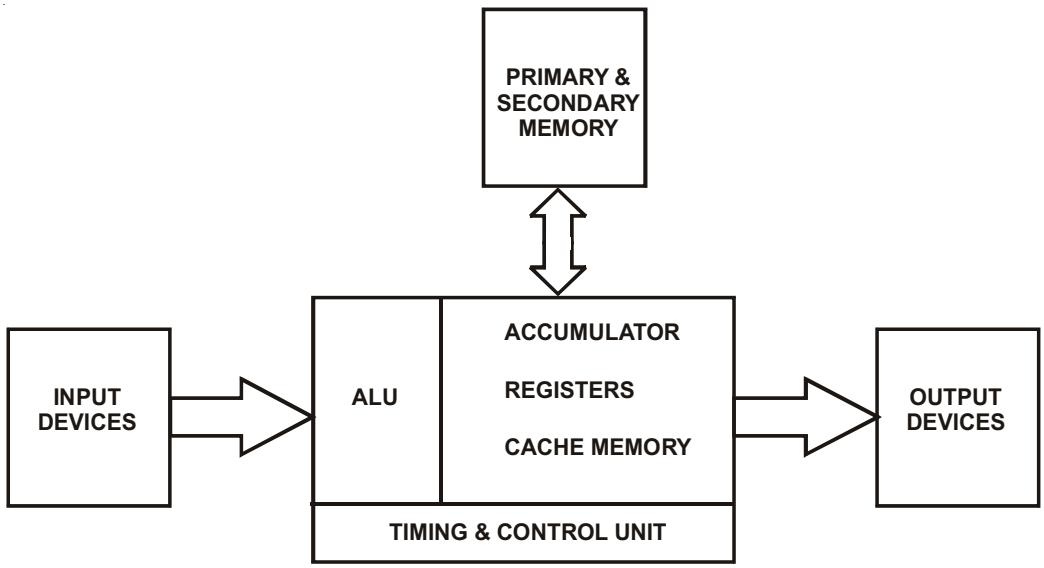

Fig. 1.1 : Block Diagram of a Computer

The **Central Processing Unit** is the Heart and Brain of the computer. It takes data and instructions from the outside world through Input devices, stores them in the memory, does the required processing taking them from memory and stores the results in the memory for future use and also gives the output to the outside world through output devices immediately after processing. The *CPU consists of Arithmetic logic unit,* which does the computations and logic

operations, accumulator which is a register used for taking the data and doing the computations, general and special purpose registers for storing the data and instructions inside the CPU on receipt from memory and a timing and control unit which generates timing and control signals required for the execution of instructions and also control the operation of other parts of CPU, Memory and Input, Output devices.

Memory stores the information like program, data results and any other required information. We have a large **Main memory or Primary memory** which is fast and accessible to the computer immediately. We have the **Secondary memory or auxiliary memory** which stores voluminous information including the Operating System. We have also got a small amount of **cache memory** inside the Microprocessor which is a part of the CPU for storing the immediate instructions and necessary data for faster access.

The **Input devices** like the Key board, Card or Tape readers pass on the Input to the CPU as electrical impulses. The **output devices** display the results either on the monitor or screen, or store in the Hard Disc, Floppy Disks, CDs or DVDs or Pen Drives or magnetic disks or magnetic tapes etc.,

The Microprocessor which forms a main component of the CPU, being an electronic device takes electrical impulses and hence it can understand only the machine language consisting of ones and zeros, one being the state of full voltage (5 Volts) and zero being no voltage. These states can also be described as high and low, up and down. This language consisting of only ones and zeros is a Low level language and to program in this **Low level language or machine language** is tedious and susceptible to making mistakes. Hence a slightly Higher level languages known as the **Assembly language** consisting of mnemonics is used in micro-processors. This consists of English abbreviations. To program in assembly language we have to keep track of the memory addresses, register names and their contents etc., which is again difficult. To make communication with the computer easier, **High level languages** like Basic, Fortran, Cobal, Pascal etc., have been designed.

C Language is an Intermediate or Middle level language between the high level language and the Low level language. A program written in C can be carried out at a considerably less time because of its features like few restrictions, compact set of key words, block structures, less documentation, rich set of operators and many built in functions. It consists of features like manipulations of bits, bytes and addresses as in Assembly language. C is highly portable. **Through its functions and structures C language has become very useful in the design of compilers and operating systems** which are essential with features like interface between the user and the computer, controls of the peripherals and other devices to be used by the computer. Compilers translate

the High level language program into Low level language understandable by the computer. *High level languages are easy to program by the user but it takes more execution time compared to low level language.*

1.2. HISTORY OF C & C++ :

The most popular programming language C is the result of a development process that started with an older language BASIC COMBINED PROGRAMMING LANGUAGE (BCPL). "Martin Richards" developed BCPL. Ken Thompson developed a language called B, during 1960s. B was modified by Dennis Ritchie which led to the development of C and it was implemented in Bell Laboratories in the 1970s. Since it was developed along with the Unix operating system, for many years C was strongly associated with UNIX. With Micro Computers gaining popularity C gained widespread implementations, under a number of operating systems, including MS-DOS. The source codes accepted by most of these operating systems are highly compatible. C++ is a superset of C.

C and C++ languages being closely related, a compiler for C++, can also work as compiler for C language. There are several types of compilers for C language like Turbo C, Ansy C, Borland C, Unix based C, DOS based C compilers. There are C compilers available for DOS Operating System, OS/2, for Macintosh and also in Microsoft Visual Studio 6.0. Turbo C++ compiler is one of the popular compilers which is used for some of the programs executed in this book. There may be some slight differences between Turbo C compiler and Turbo C++ compiler which will be pointed out in this book as and when required.

1.3. DESIGN, DEVELOPMENT AND EXECUTION OF A PROGRAM :

1. Before developing a program for solving a problem, *the problem has to be understood clearly* as to what are the Inputs, the specifications (and limitations) what is expected to be done and what is the required output and in what form is the output required.
2. *The logic required to meet the constrains and expectations of the problem has to be developed.*
3. After understanding the problem a step by step approach called **algorithm** for solving the problem has to be developed. *The algorithm is a step by step approach for solving the problem without getting into infinite loop.* Infinite loop is a set of steps or instructions that are repeated indefinitely.
4. In the place of Algorithm *a pictorial representation of Algorithms called Flow chart may be developed.*

5. Instead of Flow chart a **Pseudo code** may be developed. *A pseudo code is a program written in any High level language like statements without restricting itself to any particular language using the basic features of a Programming language.*

6. From the Algorithm or a Flow chart or pseudo code a program in the required programming language may be developed. *The program code written by the user in a programming language is called a* ***Source Code.*** This source program is given a file name in the format **filename .C** or **filename .cpp**, with the extension ".c or .cpp" indicating the compiler to be used for executing the program.

7. For Debugging and testing, this source code is loaded on the computer and *the compiler converts the source code into* **object code** *which is executable* after scanning through all the statements given in the program. ***A compiler is a program which translates all the source code into object code in one go.*** If there are any mistakes it lists out all the syntax errors i.e., errors in the format of the programming statements, errors in defining the identifiers etc., which are said to be **grammatical errors or syntax errors** in programming and displays the error code. After rectifying the mistakes or removing the errors, the program is again fed to the compiler. **The name of the file with a dot (.) extension indicates type of the file.** Ex : **File name .obj** will be the name of the object code file. *An executable file will have the extension* "**.exe**" Instead of using a compiler an interpreter may be used. **An interpreter** *is a program which reads and translates the source program into object code stepwise and any error in that step is pointed out immediately and further translation is stopped.*

8. *After successful compilation of the full program the system links the different library functions and the user define functions, creates a table of symbols assigns the values to the different variables that are defined and used in the program etc.,*

9. *After successful linking the program is executed and the results are displayed or given as outputs in the form required by the user as given in the program.*

10. After obtaining the results the programmer has to analyze the results and see whether he has got the output as desired. Otherwise the program logic will have to be redesigned or it has to be checked whether the inputs have been given proper values or whether any illegal operations like division by zero or overflow or underflow has occurred. In such cases the program has to be revised and fed again.

11. After successful execution of the said program it has to be checked whether the program gives correct results for different values of the data.

1.4 TESTING OF PROGRAMS : There are two types of testing known as **i) White box testing, ii) Black box testing.**

1.4.1 White Box Testing : White box testing is used during the early stages of testing the software to test the internal working of the program. Test cases are provided to verify the proper execution of all independent paths within a module, all logical decisions, and all loops and validate data structures.

1.4.2 Black Box Testing : Knowing the specified functions used in a program, tests can be conducted to demonstrate each function and its operations, simultaneously searching for errors. Black Box tests are used to demonstrate that the software functions are operational, input is correctly accepted, and output is produced correctly and attempts to find errors like, incorrect or missing functions, interface errors, Errors in data structures or data base access, performance errors, initialization/ termination errors.

1.4.3. Testing Bench Mark Problems: *In real time problems it will be a better practice to run the program for bench mark problems, if there are any* as in power systems. **Bench mark problems are standard problems designed by standard institutions like IEEE etc., for which the standard input and outputs are available.** If the program works for different bench mark programs for the same type of problem then it is reasonable to assume that the program developed will work for any other data or any other size of the problem.

1.5 POINTS TO BE CONSIDERED IN PROGRAMMING :

1. For real time problems collection of information, selection of solution, documentation are very important. *Documentation involves developing and writing supporting manuals and on-line assistance or help which the user will need to understand and use the finished program effectively.* Failing to use a good job at this stage can invalidate all the good work done up till now.
2. *It is always better to write a working program first* i.e., a program which works correctly and gives correct results. *Then the program can be refined* to make it more efficient with less coding and requiring less execution time and less memory.
3. **It is also better to write as generic a program as possible by using general variables instead of particular values, template functions** so that the program can work for any values of the variables and data types without a need for rewriting the program steps for different values.

1.6 DIFFERENT LEVELS OF PROGRAMMING LANGUAGES :

A "**programming language**" is a language designed to describe a set of consecutive actions to be executed by a computer. A programming language is therefore a practical way for us (humans) to give instructions to a computer.

On the other hand, the term "natural language" defines a means of communication shared by a group of individuals (for example: English or French)

Languages that computers use to communicate with each other, have nothing to do with programming languages, they are referred to as communication protocols, these are two very different concepts. A programming language is very strict:

EACH instruction corresponds to ONE processor action.

The language used by the processor is called **machine code**. The code that reaches the processor consists of a series of 0s and 1s known as (binary data).

Machine code is therefore difficult for humans to understand, which is why intermediary languages, which can be understood by humans, have been developed. The code written in this type of language is transformed into machine code so that the processor can process it.

The assembler was the first programming language ever used. This is very similar to machine code but can be understood by developers. Nonetheless, such a language is so similar to machine code that it strictly depends on the type of processor used (each processor type may have its own machine code). Thus a program developed for one machine may not be *ported* to another type of machine. The term "**portability**" describes the ability to use a software program on different types of machines. A software program written in assembler code, may sometimes have to be completely rewritten to work on another type of computer!

A programming language has therefore several advantages:

- It is much more understandable than machine code;
- It allows greater portability, i.e. can be easily adapted to run on different types of computers.

1.6.1 Imperative and functional programming languages

Programming languages are generally divided into two major groups according to how their commands are processed:

- Imperative languages;
- Functional languages.

1.6.1.1 Imperative programming language :

An imperative language programs using a series of commands, grouped into blocks and comprising of conditional statements which allow the program to return to a block of commands if the condition is met. These were the first programming languages in use, even today many modern languages still use this principle.

Structured imperative languages suffer, however, from lack of flexibility due to the sequentiality of instructions.

1.6.2 Functional programming language :

A **functional programming language** (often called *procedural language*) is a language which creates programs using functions, returning to a new output state and receiving as input the result of other functions. When a function invokes itself, we refer to this as recursion.

1.6.3 Interpretation and compilation :

Programming languages may be roughly divided into two categories:

- **interpreted languages**
- **compiled languages**

1.6.3.1. Interpreted language :

A programming language is by definition different to machine code This must therefore be translated so that the processor can understand the code. A program written in an interpreted language requires an extra program (the interpreter) which translates the programs commands as needed.

1.6.3.2 Compiled language :

A program written in a "**compiled**" language is translated by an additional program called a **compiler** which in turn creates a new stand-alone file which does not require any other program to execute itself, such a file is called an **executable**.

A program written in a compiled language has the advantage of not requiring an additional program to run it once it has been compiled. Furthermore, as the translation only needs to be done once, at compilation it executes much faster. However, it is not as flexible as a program written in an interpreted language, as each modification of the source file (the file understandable by humans: the file to be compiled) means that the program must be recompiled for the changes to take effect.

On the other hand, a compiled program has the advantage of guaranteeing the security of the source code. In effect, interpreted language, being a directly legible language, means that anyone can find out the secrets of a program and thus copy or even modified the program. There is therefore a risk of copyright violation. On the other hand, certain secure applications need code confidentiality to avoid illegal copying (bank transactions, on-line payments, secure communications...).

1.6.3.3. Intermediary languages :

Some languages belong to both categories (LISP, Java, Python...) as the program written in these languages may in certain cases undergo an intermediary compilation phase, into a file written in a language different to the source file and non-executable (requiring an interpreter). Java applets, small programs, often loaded in web pages, are compiled files, which can only be executed from within a web browser (these are files with the .class extension).

1.6.4. Some widely used languages :

Table 1.1. gives a non-exhaustive list of current programming languages

Table 1.1. List of some current programming languages

Language	Main application area	Compiled/interpreted
ADA	Real-time	Compiled language
BASIC	Programming for educational purposes	Interpreted language
C	System programming	Compiled language
C++	System object programming	Compiled language
Cobol	Management	Compiled language
Fortran	Calculation	Compiled language
Java	Internet oriented programming	Intermediary language
MATLAB	Mathematical calculations	Interpreted language
Mathematica	Mathematical calculations	Interpreted language
LISP	Artificial intelligence	Intermediary language
Pascal	Education	Compiled language
PHP	Dynamic website development	Interpreted language
Prolog	Artificial intelligence	Interpreted language
Perl	Processing character strings	Interpreted language

1.7. APPROACHES IN PROGRAMMING :

There are 3 approaches or methodologies followed in programming.

i) Stepwise Refinement ii) Modular programming,
iii) Structured programming.

i) **Stepwise Refinement :** In this method a problem is decomposed into smaller parts or smaller steps and *each step is programmed and analyzed for correctness and refined before going to the next part.*

ii) **Modular programming :** *Large programs are split up into different modules of programming and the modules are programmed separately and are tested separately and all the modules are assembled into a single program* by integrating into a single program. The process of combining the different modules is also known as linking. Different modules can be designed by different programmers. *These modules can be reused in other programs if the modules are programmed in a general manner.*

Conceptually, modules represent a separation of concerns, and improve maintainability by enforcing logical boundaries between components. Modules are typically incorporated into the program through interfaces. A module interface expresses the elements that are provided and required by the module. The elements defined in the interface are detectable by other modules. The implementation contains the working code that corresponds to the elements declared in the interface.

Languages that formally support the module concept include Ada, Algol, BlitzMax, COBOL, Component Pascal, D, Erlang, F, Fortran, Haskell, IBM/360 Assembler, IBM RPG, Java (packages are considered modules in the JLS), MATLAB, ML, Modula-2, Modula-3, Morpho, Oberon, NEWP, OCaml, Pascal, Perl, PL/I, Python, and Ruby. The IBM System i also uses Modules in CL, COBOL, and RPG when programming in the Integrated Language Environment (ILE). Modular programming can be performed even where the programming language lacks explicit syntactic features to support named modules.

Software tools can create modular code units from groups of components. Libraries of components built from separately compiled modules can be combined into a whole by using a linker. modular designed systems, if built correctly, far more reusable than a traditional monolithic design, since all (or many) of these modules may then be reused (without change) in other projects. This also facilitates the "breaking down" of projects (through "divide and conquer") into several smaller projects.

iii) **Structured Programming :** Structured programming was first proposed in the beginning of 1970's by Prof. Edsger Wybe Dijkstra. *Structured programming is a technique for organizing and coding computer programs in which a hierarchy of modules is used, or structures like 'for' structures, 'while' structures etc., each having a single entry point and a single exit point,* and in which control is passed downward through the structure without unconditional branches to other structures, eliminating the use of unconditional branching like 'goto' statement.

` Structured programming (sometimes known as modular programming) is a subset of procedural programming that enforces a logical structure on the program being written to make it more efficient and easier to understand and modify. Certain languages such as Ada, Pascal, and dBASE are designed with features that encourage or enforce a logical program structure.

Structured programming frequently employs a top-down design model, in which developers map out the overall program structure into separate subsections. A defined function or set of similar functions is coded in a separate module or submodule, which means that code can be loaded into memory more efficiently and that modules can be reused in other programs. After a module has been tested individually, it is then integrated with other modules into the overall program structure.

Program flow follows a simple hierarchical model that employs looping constructs such as "for," "repeat," and "while." Use of the "Go To" statement is discouraged.

Object-oriented programming (OOP) can be thought of as a type of structured programming, uses structured programming techniques for program flow, and adds more structure for data to the model.

1.8. TOP-DOWN PROGRAMMING AND STEP-WISE REFINEMENT :

Top-down programming is a style in programming. The mainstay of traditional procedural languages, in which design begins by specifying complex pieces and then dividing them into successively smaller pieces. The technique for writing a program using top–down methods is to write a main procedure that names all the major functions it will need. Later, the programming team looks at the requirements of each of those functions and the process is repeated. These compartmentalized sub-routines eventually will perform actions so simple they can be easily and concisely coded. When all the various sub-routines have been coded the program is ready for testing. By defining how the application comes together at a high level, lower level work can be self-contained. By defining how the lower level abstractions are expected to integrate into higher level ones, interfaces become clearly defined.

Also called "stepwise refinement," it is a software development technique that imposes a hierarchical structure on the design of the program. It starts out by defining the solution at the highest level of functionality and breaking it down further and further into small routines that can be easily documented and coded.

C Languages follows Top-down programming approach with the main function at the top and user defined functions following the main function.

1.8.1 (a) Advantages of Top down Programming:

1. All aspects of the program are know before programming starts
2. Programmers are focused on the task, because they know every aspect of the program.
3. The code should be clean and easy to understand, because it all has purpose.
4. Easier to understand the problem
5. Easier to add a new module
6. Easier to modify a module
7. Easier to test/debug a module
8. Several programmers can work on the same problem/program.
9. Modules can be kept in a library and reused in another program

(b) Disadvantages of Top down programming :

1. Testing is hard until near the end of the project, because no code is written until it is all properly designed.
2. Any mistakes made early on in the development cycle can be hard to fix.

1.9. BOTTOM-UP PROGRAMMING :

Bottom-up programming is the opposite of top-down programming. It refers to a style of programming where an application is constructed starting with existing primitives of the programming language, and constructing gradually more and more complicated features, until the all of the application has been written.

C++ Language follows bottom of approach with objects, classes etc., defined in the beginning and the main programming following after them.

1.9.1 Working of bottom-up programming :

In a language such as C or Java, bottom-up programming takes the form of constructing abstract data types from primitives of the language or from existing abstract data types.

In Common Lisp, in addition to constructing abstract data types, it is common to build *functions* bottom-up from simpler functions, and to use macros to construct new *special forms* from simpler ones.

In a bottom-up approach, the individual base elements of the system are first specified in great detail. These elements are then linked together to form larger subsystems, which then in turn are linked, sometimes in many levels, until a complete top-level system is formed. This strategy often resembles a "seed" model, whereby the beginnings are small, but eventually grow in complexity

and completeness. Object-oriented programing (OOP) is a paradigm that uses "objects" to design applications and computer programs. In mechanical engineering with software programs such as Pro/ENGINEER, Solidworks, and Autodesk Inventor users can design products as pieces not part of the whole and later add those pieces together to form assemblies like building LEGO. Engineers call this piece part design.

This bottom-up approach has one weakness. Good intuition is necessary to decide the functionality that is to be provided by the module. If a system is to be built from existing system, this approach is more suitable as it starts from some existing modules. Pro/ENGINEER (as well as other commercial computer-aided design (CAD) programs) does however hold the possibility to create top-down design by the use of so-called *skeletons*. They are generic structures that hold information on the overall layout of the product. Parts can inherit interfaces and parameters from this generic structure. Like parts, skeletons can be put into a hierarchy. Thus, it is possible to build the overall layout of a product before the parts are designed.

1.9.2 (a) Advantages of bottom-up programming :

1. Testing is simplified. The code can be properly tested at any stage in the development cycle.
2. Pieces of programs written bottom-up tend to be more general, and thus more reusable, than pieces of programs written top-down. In fact, one can argue that the purpose bottom-up programming is to create an application-specific language . Such a language is suitable for implementing an entire class of applications, not only the one that is to be written. This fact greatly simplifies maintenance, in particular adding new features to the application. It also makes it possible to delay the final decision concerning the exact functionality of the application. Being able to delay this decision makes it less likely that the client has changed his or her mind between the establishment of the specifications of the application and its implementation.
3. Design flaws can be easily spotted and fixed. Because all design work is done through the code.

(b) Disadvantages of bottom up programming :

1. Programmers may come across many problems during development because they don't have a structured well though out design plan.
2. It can be easy to deviate from the task during development.
3. The code may be messy, because the task was misunderstood early ion development and code has to be changed a lot.

1.10. OPERATING SYSTEMS :

An **operating system** (**OS**) is a collection of software that manages computer hardware resources and provides common services for computer programs. The operating system is a vital component of the system software in a computer system. Application programs require an operating system to function.

Time-sharing operating systems schedule tasks for efficient use of the system and may also include accounting for cost allocation of processor time, mass storage, printing, and other resources.

For hardware functions such as input and output and memory allocation, the operating system acts as an intermediary between programs and the computer hardware, although the application code is usually executed directly by the hardware and will frequently make a system call to an OS function or be interrupted by it. Operating systems can be found on almost any device that contains a computer—from cellular phones and video game consoles to supercomputers and web servers

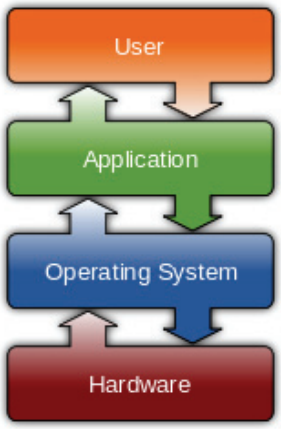

Fig.1.2. An Operating System

Examples of popular modern operating systems include Android, BSD, iOS, Linux, Mac OS X, Microsoft Windows,[3] Windows Phone, and IBM z/OS. All these, except Windows and z/OS, share roots in UNIX.

1.10.1 Types of operating systems :

(i) Real Time Operating System :

Real timeA real-time operating system is a multitasking operating system that aims at executing real-time applications, intended for applications with fixed deadlines (real-time computing). Such applications include some small embedded systems, automobile engine controllers, industrial robots, spacecraft, industrial control, and some large-scale computing systems. Real-time operating systems often use specialized scheduling algorithms

so that they can achieve a deterministic nature of behavior. The main objective of real-time operating systems is their quick and predictable response to events. They have an event-driven or time-sharing design and often aspects of both. An event-driven system switches between tasks based on their priorities or external events while time-sharing operating systems switch tasks based on clock interrupts. A real-time operating system (RTOS) is a multitasking operating system

(ii) Multi-user Operating System :

Multi-userA multi-user operating system allows multiple users to access a computer system at the same time. Time-sharing systems and Internet servers can be classified as multi-user systems as they enable multiple-user access to a computer through the sharing of time. Single-user operating systems have only one user but may allow multiple programs to run at the same time.

Multi-tasking vs. single-tasking : A multi-tasking operating system allows more than one program to be running at a time, from the point of view of human time scales. A single-tasking system has only one running program. Multi-tasking can be of two types: pre-emptive or co-operative. In pre-emptive multitasking, the operating system slices the CPU time and dedicates one slot to each of the programs. Unix-like operating systems such as Solaris and Linux support pre-emptive multitasking, as does Amiga OS. Cooperative multitasking is achieved by relying on each process to give time to the other processes in a defined manner. 16-bit versions of Microsoft Windows used cooperative multi-tasking. 32-bit versions, both Windows NT and Win9x, used pre-emptive multi-tasking. Mac OS prior to OS X used to support cooperative multitasking.

(iii) Distributed system :

A distributed operating system manages a group of independent computers and makes them appear to be a single computer. The development of networked computers that could be linked and communicate with each other gave rise to distributed computing. Distributed computations are carried out on more than one machine. When computers in a group work in cooperation, they make a distributed system.

(iv) Embedded operating systems :

Embedded Operating Systems are designed to be used in embedded computer systems. They are designed to operate on small machines like PDAs with less autonomy. They are able to operate with a limited number of resources. They are very compact and extremely efficient by design. Windows CE and Minix 3 are some examples of embedded operating systems.

1.10.2. Examples of operating systems

(i) UNIX and UNIX-like operating systems

Fig. 1.3. Unix and Unix-like Operating System

Ken Thompson wrote B, mainly based on BCPL, which he used to write Unix, based on his experience in the MULTICS project. B was replaced by C, and Unix developed into a large, complex family of inter-related operating systems which have been influential in every modern operating system The *UNIX-like* family is a diverse group of operating systems, with several major sub-categories including System V, BSD, and Linux. The name "UNIX" is a trademark of The Open Group which licenses it for use with any operating system that has been shown to conform to their definitions. "UNIX-like" is commonly used to refer to the large set of operating systems which resemble the original UNIX.Unix-like systems run on a wide variety of computer architectures. They are used heavily for servers in business, as well as workstations in academic and engineering environments. Free UNIX variants, such as Linux and BSD, are popular in these areas. Four operating systems are certified by the The Open Group (holder of the Unix trademark) as Unix. HP's HP-UX and IBM's AIX are both descendants of the original System V Unix and are designed to run only on their respective vendor's hardware. In contrast, Sun Microsystems's Solaris Operating System can run on multiple types of hardware, including x86 and Sparc servers, and PCs. Apple's Mac OS X, Unix interoperability was sought by establishing the POSIX standard. The POSIX standard can be applied to any operating system, although it was originally created for various Unix variants.

(ii) The standard user interface of Mac OS X :

Fig. 1.4. standard user interface of Mac OS X

Mac OS X is a line of open core graphical operating systems developed, marketed, and sold by Apple Inc., the latest of which is pre-loaded on all currently shipping Macintosh computers. Mac OS X is the successor to the original Mac OS, which had been Apple's primary operating system since 1984.

Six more distinct "client" and "server" editions of Mac OS X have been released, the most recent being OS X 10.8 "Mountain Lion", which was first made available on February 16, 2012 for developers, and was then released to the public on July 25, 2012. Releases of Mac OS X are named after big cats.

The server edition, Mac OS X Server, is architecturally identical to its desktop counterpart but usually runs on Apple's line of Macintosh server hardware. Mac OS X Server includes work group management and administration software tools that provide simplified access to key network services, including a mail transfer agent, a Samba server, an LDAP server, a domain name server, and others. In Mac OS X v10.7 Lion, all server aspects of Mac OS X Server have been integrated into the client version. Mountain lion is the latest version in Mac O.S.

GNU is a Unix-like operating system that is free software—it respects your freedom. You can install Linux-based versions of GNU which are entirely free software. The name "GNU" is a recursive acronym for "GNU's Not Unix!". "GNU" is pronounced *g'noo*, as one syllable, like saying "grew" but replacing the *r* with *n*.

A Unix-like operating system is a software collection of applications, libraries, and developer tools, plus a program to allocate resources and talk to the hardware, known as a kernel.

(iii) GNU, Linux, and Linux kernel :

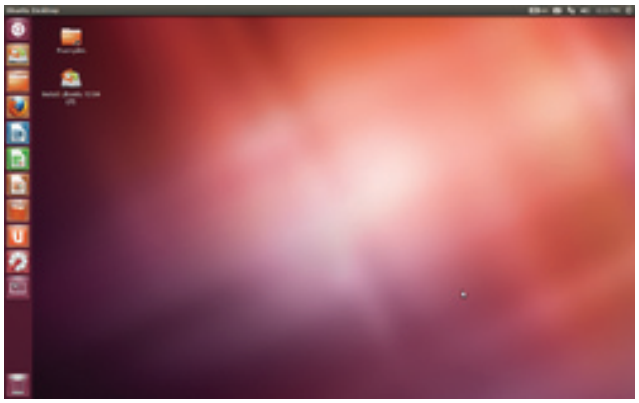

Fig. 1.5. Screen shot of Linux

Ubuntu, desktop Linux distribution is a very popular Linux Operting System.

(iv) Android :

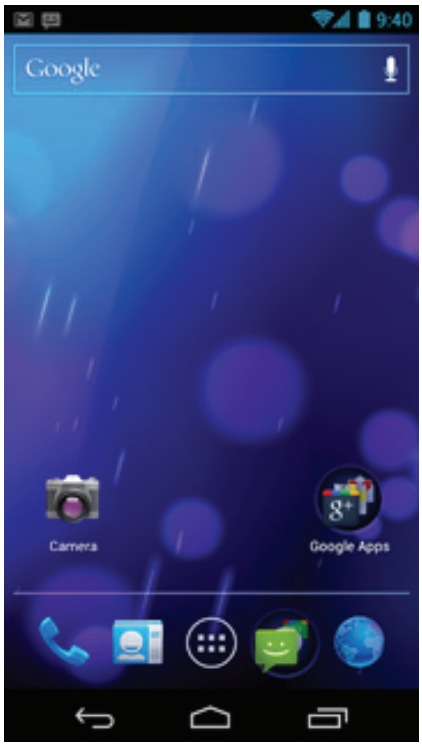

Fig. 1.6. Android Operating System Model in Mobile Phones

It is a popular mobile operating system using the Linux kernelLinux (or GNU/Linux) is a Unix-like operating system that was developed without any actual Unix code, unlike BSD and its variants. Linux can be used on a wide range of devices from supercomputers to wristwatches. The Linux kernel is released under an open source license, so anyone can read and modify its code. It has been modified to run on a large variety of electronics. Although estimates suggest that Linux is used on 1.82% of all personal computers, it has been widely adopted for use in servers and embedded systems (such as cell phones). Linux has superseded Unix in most places, and is used on the 10 most powerful supercomputers in the world. The Linux kernel is used in some popular distributions, such as Red Hat, Debian, Ubuntu, Linux Mint and Google's Android.

(v) Google Chrome OS :

Chrome is an operating system based on the Linux kernel and designed by Google. Since Chrome OS targets computer users who spend most of their time on the Internet, it is mainly a web browser with no ability to run applications. It relies on Internet applications (or Web apps) used in the web browser to accomplish tasks such as word processing and media viewing, as well as online storage for storing most files.

(vi) Microsoft Windows :

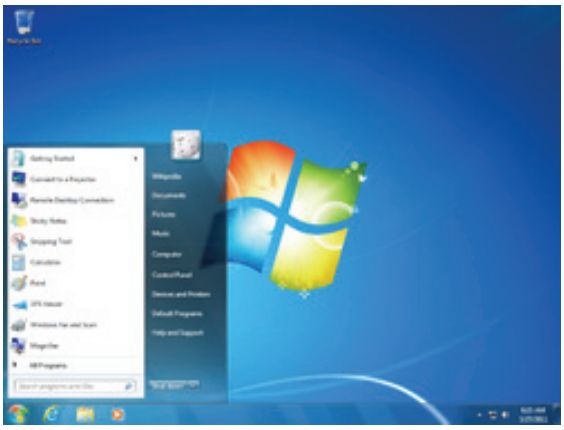

Fig.1.7. Screen shot of MS Windows O.S

Microsoft Windows 7 Desktop. Microsoft has just now released in 2012 its Windows 8 O.S. Still 32 bit Windows XP is very popular.Microsoft Windows is a family of proprietary operating systems designed by Microsoft Corporation and primarily targeted to Intel architecture based computers, with an estimated 88.9 percent total usage share on Web connected computers.[Microsoft Windows originated in 1985 as an operating environment running on top of MS-DOS, which was the standard operating system shipped on most Intel architecture personal computers at the time. In 1995, Windows 95 was released which only used MS-DOS as a bootstrap. For backwards compatibility,The newest version is Windows 8 for workstations and Windows Server 2012 for servers. Windows 7 are new aditions. ·Win9x could run real-mode MS-DOS and 16 bits Windows 3.x drivers. Windows Me, released in 2000, was the last version in the Win9x family. Later versions have all been based on the Windows NT kernel. Current versions of Windows run on IA-32 and x86-64 microprocessors, although Windows 8 will support ARM architecture. In the past, Windows NT supported non-Intel architectures.

1.10.3. Components Of an Operating System :

The components of an operating system all exist in order to make the different parts of a computer work together. All user software needs to go through the operating system in order to use any of the hardware, whether it be as simple as a mouse or keyboard or complex as an Internet connection.

(i) Kernel :

A kernel connects the application software to the hardware of a computer.Main article: Kernel (computing)With the aid of the firmware and device drivers, the kernel provides the most basic level of control over all of the computer's hardware devices.

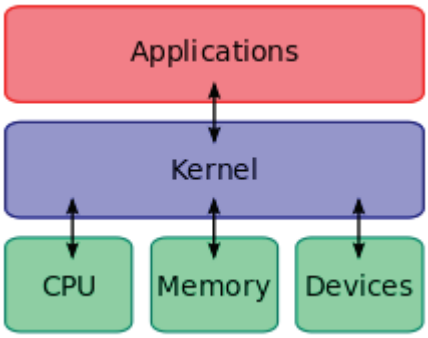

Fig.1.8. Components of Unix/Linus O.S.

Kernel manages memory access for programs in the RAM, it determines which programs get access to which hardware resources, it sets up or resets the CPU's operating states for optimal operation at all times, and it organizes the data for long-term non-volatile storage with file systems on such media as disks, tapes, flash memory, etc.

1.10.4 Modes :

(i) Protected mode and Supervisor mode :

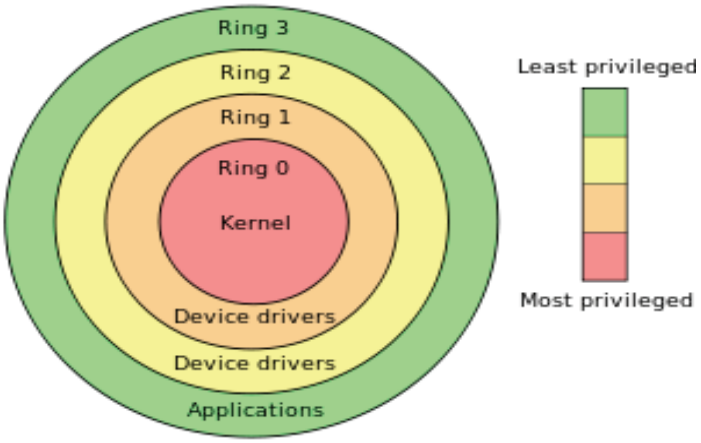

Fig.1.9 Different layers in Unix Operating System

Privilege rings for the x86 available in protected mode. Operating systems determine which processes run in each mode.Modern CPUs support multiple modes of operation. CPUs with this capability use at least two modes: protected mode and supervisor mode. The supervisor mode is used by the operating system's kernel for low level tasks that need unrestricted access to hardware, such as controlling how memory is written and erased, and communication with devices like graphics cards. Protected mode, in contrast, is used for almost everything else. Applications operate within protected mode, and can only use hardware by communicating with the kernel, which controls everything in

supervisor mode. CPUs might have other modes similar to protected mode as well, such as the virtual modes in order to emulate older processor types, such as 16-bit processors on a 32-bit one, or 32-bit processors on a 64-bit one.When a computer first starts up, it is automatically running in supervisor mode. The first few programs to run on the computer, being the BIOS or EFI, bootloader, and the operating system have unlimited access to hardware - and this is required because, by definition, initializing a protected environment can only be done outside of one. However, when the operating system passes control to another program, it can place the CPU into protected mode.In protected mode, programs may have access to a more limited set of the CPU's instructions. A user program may leave protected mode only by triggering an interrupt, causing control to be passed back to the kernel. In this way the operating system can maintain exclusive control over things like access to hardware and memory.The term "protected mode resource" generally refers to one or more CPU registers, which contain information that the running program isn't allowed to alter. Attempts to alter these resources generally causes a switch to supervisor mode, where the operating system can deal with the illegal operation the program was attempting (for example, by killing the program).

(ii) Memory management :

Memory protection enables the kernel to limit a process' access to the computer's memory. Various methods of memory protection exist, including memory segmentation and paging. All methods require some level of hardware support (such as the 80286 MMU), which doesn't exist in all computers. In both segmentation and paging, certain protected mode registers specify to the CPU what memory address it should allow a running program to access. Attempts to access other addresses will trigger an interrupt which will cause the CPU to re-enter supervisor mode, placing the kernel in charge. This is called a segmentation violation or Seg-V for short, and since it is both difficult to assign a meaningful result to such an operation, and because it is usually a sign of a misbehaving program, the kernel will generally resort to terminating the offending program, and will report the error.

(iii) Virtual memory :

Virtual memory and Page fault Many operating systems can "trick" programs into using memory scattered around the hard disk and RAM as if it is one continuous chunk of memory, called virtual memory.The use of virtual memory addressing (such as paging or segmentation) means that the kernel can choose what memory each program may use at any given time, allowing the operating system to use the same memory locations for multiple tasks. If a program tries to access memory that isn't in its current range of accessible memory, but nonetheless has been allocated to it, the kernel will be interrupted in the

same way as it would if the program were to exceed its allocated memory. (See section on memory management.)

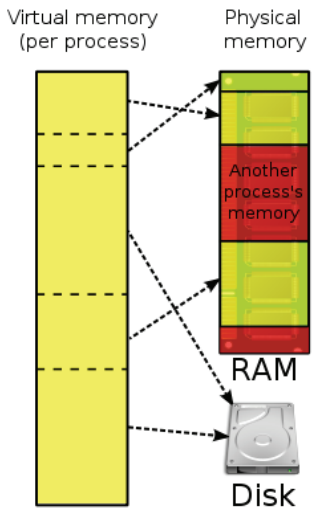

Fig.1.10.Virtual Memory Operations

Under UNIX this kind of interrupt is referred to as a page fault.When the kernel detects a page fault it will generally adjust the virtual memory range of the program which triggered it, granting it access to the memory requested. This gives the kernel discretionary power over where a particular application's memory is stored, or even whether or not it has actually been allocated yet.In modern operating systems, memory which is accessed less frequently can be temporarily stored on disk or other media to make that space available for use by other programs. This is called swapping, as an area of memory can be used by multiple programs, and what that memory area contains can be swapped or exchanged on demand."Virtual memory" provides the programmer or the user with the perception that there is a much larger amount of RAM in the computer than is really there. The philosophy governing preemptive multitasking is that of ensuring that all programs are given regular time on the CPU. This implies that all programs must be limited in how much time they are allowed to spend on the CPU without being interrupted. To accomplish this, modern operating system kernels make use of a timed interrupt. A protected mode timer is set by the kernel which triggers a return to supervisor mode after the specified time has elapsed. (See above sections on Interrupts and Dual Mode Operation.)A virtual file system or VFS. allows the operating system to provide programs with access to an unlimited number of devices with an infinite variety of file systems installed on them, through the use of specific device drivers and file system drivers.A connected storage device, such as a hard drive, is accessed through a device driver. The device driver understands the specific language of the drive and is able to translate that language into a standard language used by the operating system to access

all disk drives. On UNIX, this is the language of block devices.When the kernel has an appropriate device driver in place, it can then access the contents of the disk drive in raw format, which may contain one or more file systems. A file system driver is used to translate the commands used to access each specific file system into a standard set of commands that the operating system can use to talk to all file systems. Programs can then deal with these file systems on the basis of filenames, and directories/folders, contained within a hierarchical structure. They can create, delete, open, and close files, as well as gather various information about them, including access permissions, size, free space, and creation and modification dates.Unlike other operating systems, Linux and UNIX allow any file system to be used regardless of the media it is stored in, whether it is a hard drive, a disc (CD,DVD...), a USB flash drive, or even contained within a file located on another file system.

1.10.5 Networking of Computers :

Computer networkCurrently most operating systems support a variety of networking protocols, hardware, and applications for using them. This means that computers running dissimilar operating systems can participate in a common network for sharing resources such as computing, files, printers, and scanners using either wired or wireless connections. Networks can essentially allow a computer's operating system to access the resources of a remote computer to support the same functions as it could if those resources were connected directly to the local computer. This includes everything from simple communication, to using networked file systems or even sharing another computer's graphics or sound hardware. Some network services allow the resources of a computer to be accessed transparently, such as SSH which allows networked users direct access to a computer's command line interface.

(i) Client/server Configuration :

Client/server networking allows a program on a computer, called a client, to connect via a network to another computer, called a server. Servers offer (or host) various services to other network computers and users. These services are usually provided through ports or numbered access points beyond the server's network address. Each port number is usually associated with a maximum of one running program, which is responsible for handling requests to that port. A daemon, being a user program, can in turn access the local hardware resources of that computer by passing requests to the operating system kernel.Many operating systems support one or more vendor-specific or open networking protocols as well, for example, SNA on IBM systems, DECnet on systems from Digital Equipment Corporation, and Microsoft-specific protocols (SMB) on Windows. Specific protocols for specific tasks may also be supported such as NFS for file access. Protocols like ESound, or esd can be easily extended over the network to provide sound from local applications, on a remote system's sound hardware.

1.10.6. Computer Security :

A computer being secure depends on a number of technologies working properly. A modern operating system provides access to a number of resources, which are available to software running on the system, and to external devices like networks via the kernel.The operating system must be capable of distinguishing between requests which should be allowed to be processed, and others which should not be processed. While some systems may simply distinguish between "privileged" and "non-privileged", systems commonly have a form of requester *identity*, such as a user name. To establish identity there may be a process of *authentication*. Often a username must be quoted, and each username may have a password. Other methods of authentication, such as magnetic cards or biometric data, might be used instead. In some cases, especially connections from the network, resources may be accessed with no authentication at all (such as reading files over a network share). Also covered by the concept of requester **identity** is *authorization*; the particular services and resources accessible by the requester once logged into a system are tied to either the requester's user account or to the variously configured groups of users to which the requester belongs. In addition to the allow/disallow model of security, a system with a high level of security will also offer auditing options. These would allow tracking of requests for access to resources (such as, "who has been reading this file?"). Internal security, or security from an already running program is only possible if all possibly harmful requests must be carried out through interrupts to the operating system kernel. If programs can directly access hardware and resources, they cannot be secured. External security involves a request from outside the computer, such as a login at a connected console or some kind of network connection. External requests are often passed through device drivers to the operating system's kernel, where they can be passed onto applications, or carried out directly. Security of operating systems has long been a concern because of highly sensitive data held on computers, both of a commercial and military nature. The United States Government Department of Defense (DoD) created the *Trusted Computer System Evaluation Criteria* (TCSEC) which is a standard that sets basic requirements for assessing the effectiveness of security. This became of vital importance to operating system makers, because the TCSEC was used to evaluate, classify and select trusted operating systems being considered for the processing, storage and retrieval of sensitive or classified information. Network services include offerings such as file sharing, print services, email, web sites, and file transfer protocols (FTP), most of which can have compromised security. Internal security is especially relevant for multi-user systems; it allows each user of the system to have private files that the other users cannot tamper with or read. Internal security is also vital if auditing is to be of any use, since a program can potentially bypass the operating system, inclusive of bypassing auditing.

1.10.7. Operating system user interface :

(i) A screenshot of the Bourne Again Shell command line :

Each command is typed out after the 'prompt', and then its output appears below, working its way down the screen. The current command prompt is at the bottom.

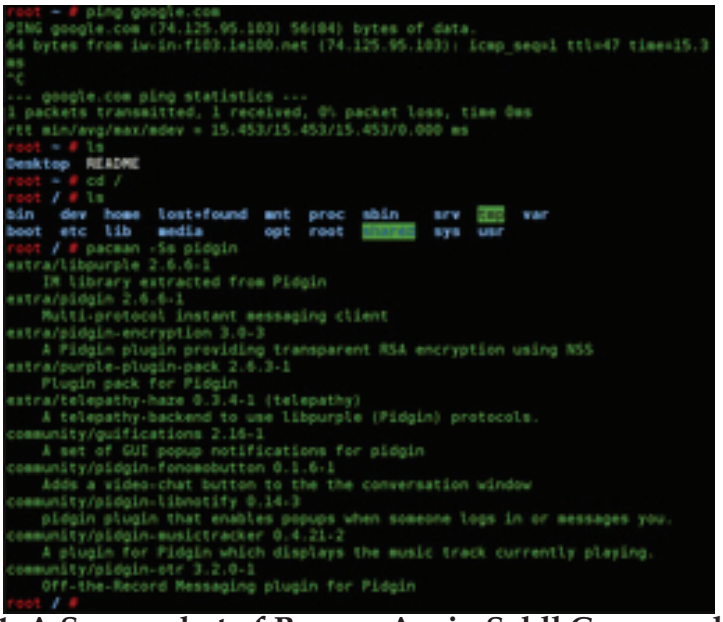

Fig.1.11. A Screen shot of Bourne Again Sehll Command Lines.

Every computer that is to be operated by an individual requires a user interface. The user interface is usually referred to as a shell and is essential if human interaction is to be supported. The user interface views the directory structure and requests services from the operating system that will acquire data from input hardware devices, such as a keyboard, mouse or credit card reader, and requests operating system services to display prompts, status messages and such on output hardware devices, such as a video monitor or printer. The two most common forms of a user interface have historically been the command-line interface, where computer commands are typed out line-by-line, and the graphical user interface, where a visual environment (most commonly a WIMP) is present.

(ii) Graphical user interfaces :

Most of the modern computer systems support graphical user interfaces (GUI), and often include them. In some computer systems, such as the original implementation of Mac OS, the GUI is integrated into the kernel.A screenshot of the KDE Plasma Desktop graphical user interface. Programs take the form of images on the screen, and the files, folders (directories), and applications take the form of icons and symbols. A mouse is used to navigate the computer.Graphical user interfaces evolve over time.

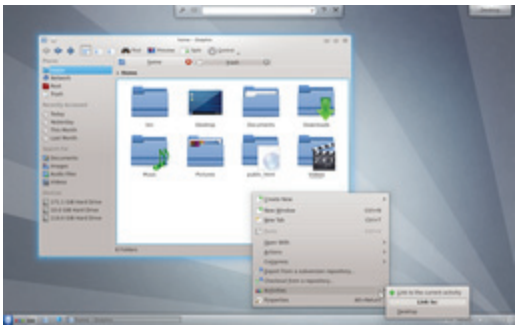

Fig. 1.12. A Screenshot of the KDE Plasma Desktop graphical user interface

For example, Windows has modified its user interface almost every time a new major version of Windows is released, and the Mac OS GUI changed dramatically with the introduction of Mac OS X in 1999.

1.10.8. Diversity of operating systems and portability :

Application software is generally written for use on a specific operating system, and sometimes even for specific hardware. When porting the application to run on another OS, the functionality required by that application may be implemented differently by that OS (the names of functions, meaning of arguments, etc.) requiring the application to be adapted, changed, or otherwise maintained.This cost in supporting operating systems diversity can be avoided by instead writing applications against software platforms like Java or Qt. These abstractions have already borne the cost of adaptation to specific operating systems and their system libraries.

(i) **Java :**

Java is a programming language originally developed by James Gosling at Sun Microsystems (which has since merged into Oracle Corporation) and released in 1995 as a core component of Sun Microsystems' Java platform. The language derives much of its syntax from C and C++, but it has fewer low-level facilities than either of them. Java applications are typically compiled to bytecode (class file) that can run on any Java virtual machine (JVM) regardless of computer architecture. Java is a general-purpose, concurrent, class-based, object-oriented language that is specifically designed to have as few implementation dependencies as possible. It is intended to let application developers "write once, run anywhere" (WORA), meaning that code that runs on one platform does not need to be recompiled to run on another. Java is as of 2012 one of the most popular programming languages in use, particularly for client-server web applications, with a reported 10 million users.

(ii) Java platform :

One characteristic of Java is portability, which means that computer programs written in the Java language must run similarly on any hardware/

operating-system platform. This is achieved by compiling the Java language code to an intermediate representation called Java bytecode, instead of directly to platform-specific machine code. Java bytecode instructions are analogous to machine code, but they are intended to be interpreted by a virtual machine (VM) written specifically for the host hardware. End-users commonly use a Java Runtime Environment (JRE) installed on their own machine for standalone Java applications, or in a Web browser for Java applets.

Standardized libraries provide a generic way to access host-specific features such as graphics, threading, and networking.

A major benefit of using bytecode is porting. However, the overhead of interpretation means that interpreted programs almost always run more slowly than programs compiled to native executables would. Just-in-Time (JIT) compilers were introduced from an early stage that compile bytecodes to machine code during runtime.

Another approach is for operating system vendors to adopt standards. For example, POSIX and OS abstraction layers provide commonalities that reduce porting costs.

1.10.9. Black Berry and Black Berry O.S :

Most BlackBerry devices are smartphones and are primarily known for their ability to send and receive push email and instant messages, mobile telephone, text messaging, Internet faxing, Web browsing and other wireless information services. It is an example of a convergent device.while maintaining a high level of security through on-device message encryption. They are also designed to function as personal digital assistants, portable media players, internet browsers, gaming devices, cameras and more. BlackBerry devices support a large variety of instant messaging features, with the most popular being the proprietary BlackBerry Messenger service. The BlackBerry PlayBook is a tablet computer offered by RIM.

The operating system used by BlackBerry devices is a proprietary multitasking environment developed by RIM. The operating system is designed for use of input devices such as the track wheel, track ball, and track pad. The OS provides support for Java MIDP 1.0 and WAP 1.2. Previous versions allowed wireless synchronization with Microsoft Exchange Server email and calendar, as well as with Lotus Domino email. OS 5.0 provides a subset of MIDP 2.0, and allows complete wireless activation and synchronization with Exchange email, calendar, tasks, notes and contacts, and adds support for Novell GroupWise and Lotus Notes. The BlackBerry Curve 9360, BlackBerry Torch 9810, Bold 9900/9930, Curve 9310/9320 and Torch 9850/9860 feature the most recent BlackBerry OS 7 (launched in 2011).

Third-party developers can write software using these APIs, and proprietary BlackBerry APIs as well. Any application that makes use of certain restricted functionality must be digitally signed so that it can be associated to a developer account at RIM..

A new OS, BlackBerry 10, is slated for release on new BlackBerry models on January 30, 2013, RIM CEO Thorsten Heins announced. At BlackBerry World 2012, RIM CEO Thorsten Heins showed off a few new features of the upcoming OS, including a camera which is able to go back in time to ensure a perfect shot, an intelligent, predictive, and adapting keyboard, and a user interface designed around the idea of "flow".

1.10.10. C Programming is a subset of C++ programming :

C Programming is a subset of C++ programming and most of the character sets, keywords, operations, data structure etc., are common to both the languages and hence it has been proposed to club C and C++ and also to treat the data structures in the two languages comparing them and make a comprehensive study of both the languages and data structures in this book. It is also felt that a comparative study of both C and C++ will enable comprehension of both the languages.

A modest effort has been made in this book to elucidate the salient features of both the languages and the use of data structures in both the languages with the help of adequate background theory and working programs.

* * *

COMPREHENSION - 1

Note : By answering all the questions given in this comprehension one can answer any objective type of questions easily whatever be the form of the question. Answers for these questions can be obtained by going through the topics in this chapter carefully.

1. What are the different blocks of a computer?
2. What are the different components of each block?
3. What are Primary and Secondary Memories?
4. Mention different Input and output devices.
5. What constitutes CPU?
6. What are meant by Registers?
7. How many types of Registers are there?
8. What is the function of ALU?
9. What is the function of Cache Memory?
10. What does Timing and controls unit do?
11. What are the different levels of computer languages?
12. What does Assembly language do?
13. What level is C language?
14. What is the specialty of C language?
15. What is meant by an operating System?
16. What are the different phases of Programming? Explain.
17. What is a Flow Chart? What purpose does it serve?
18. What is meant by Pseudo code? In what way is it different from Source code?
19. What is the process of executing a program?
20. What are the points to be remembered in Programming? What is meant by Modular and Structured Programs?
21. What are the different types of Testing? Explain.
22. What is meant by Bench mark Problems? What purpose do they serve?
23. What is Documentation? What purpose does it serve?
24. Why should we write a Generic program?
25. What is meant by file extensions? How is the file extensions represented?
26. What are Top-down and Bottom-up approaches in programming?
27. What type of programming do C and C++ follow?

28. What are the different approaches followed in programming?
29. What is an operating system?
30. What are the different popular operating systems?
31. What is meant by open source operating systems? What are its advantages? Give examples.
32. What is proprietory operating system? What are its advantages and disadvantages? give examples.
33. Explain multi-user operating system? How does it work?
34. What is Real time operating system? How does it work?
35. What are the different types of Unix-based operating systems?
36. Where do we use Android operating system?
37. What is meant by Kernel in Unix operating system?
38. What are the different layers of Unix operating system and what are their functions?
39. What are the components of operating system?
40. What are the different modes of operating system?
41. What is meant by Memory Management?
42. What is Virtual Memory? What is its purpose?
43. What is meant by Networking of Computers?
44. What is Client-server configuration? explain.
45. What is meant by computer security?
46. Explain GUI.
47. What is meant by portability of operating systems?
48. Explain Java and Java platform. How does it work?
49. What is meant by Black Berry?
50. What is Black berry operating system?

* * *

EXERCISE - 1

Note : By answering all the questions given in this exercise one can answer subjective type of questions easily including problems. The worked examples in the chapter may help in solving these questions.

1. (a) Draw the Block diagram of a Computer and explain its functioning?
   (b) Mention the different devices used in different blocks.
2. (a) Define Algorithm.
   (b) What is the use of flow chart ?
   (c) Explain the steps followed in program development and its execution?
3. (a) What is the difference between a Micro Processor and Micro Computer?
   (b) Explain about different types of memories used in a computer.
4. (a) Explain Flow chart, Pseudo code, Source code, and Machine code.
   (b) How do you conclude that the results obtained from the Computer are correct?
5. (a) Explain the different types of testing a program.
   (b) Explain the importance of documentation while Programming.
6. (a) Explain the different approaches to programming.
   (b) Explain about File Extensions.
7. (a) What is meant by Compilation of a Program?
   (b) What is the difference between a compiler and an interpreter?
8. (a) What is meant by different levels of languages? At what level does C language belong to?
   (b) Why are C and C++ languages considered to be closely related?
9. (a) What is the importance of C Programming?
   (b) Explain about Linking in execution of a Program.
10. Discuss about Top-down programming, its advantages and disadvantages.
11. Discuss about Bottom-up programming, its advantages and disadvantages.
12. Explain the necessity and functions of an Operating system?
13. What are the different components of Operating system?
14. Explain Unix Operating system and its varients?
15. Explain distributed Operating systems.
16. Explain portability of Operating system.
17. Explain Memory management.
18. Explain Networking of computer systems.

* * *