Unit – 1

Computer Fundamentals: What is a Computer, Evolution of Computers, Generations of Computers, Classification of Computers, Anatomy of a Computer, Memory revisited, Introduction to Operating systems, Operational overview of a CPU.

* * * * *

1.1 What is a Computer?

A computer is a machine or device that performs processes, calculations and operations based on instructions provided by a software or hardware program. It is designed to execute applications and provides a variety of solutions by combining integrated hardware and software components. A computer is made up of multiple parts and components that facilitate user functionality.

A computer has two primary categories:

- **Hardware**: Physical structure that houses a computer's processor, memory, storage, communication ports and peripheral devices.
- **Software**: Includes operating system (OS) and software applications.

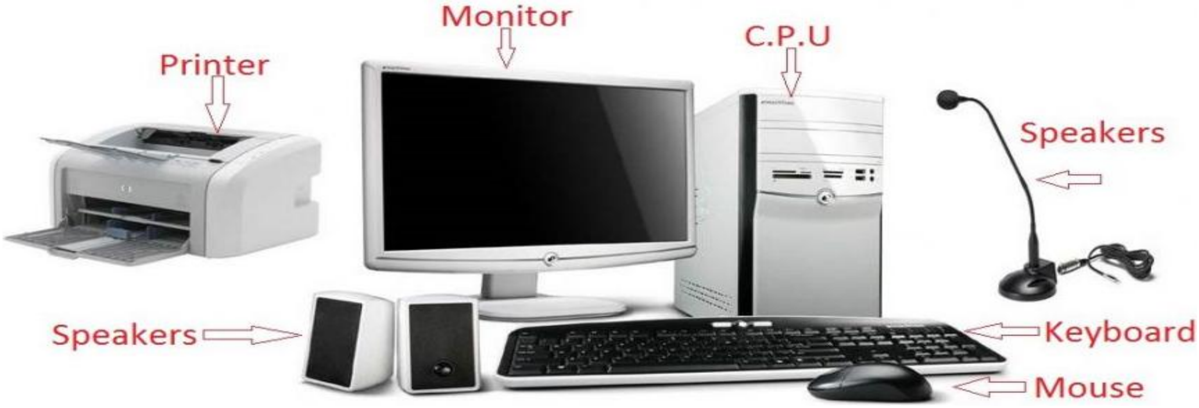


Fig 1.1: Computer

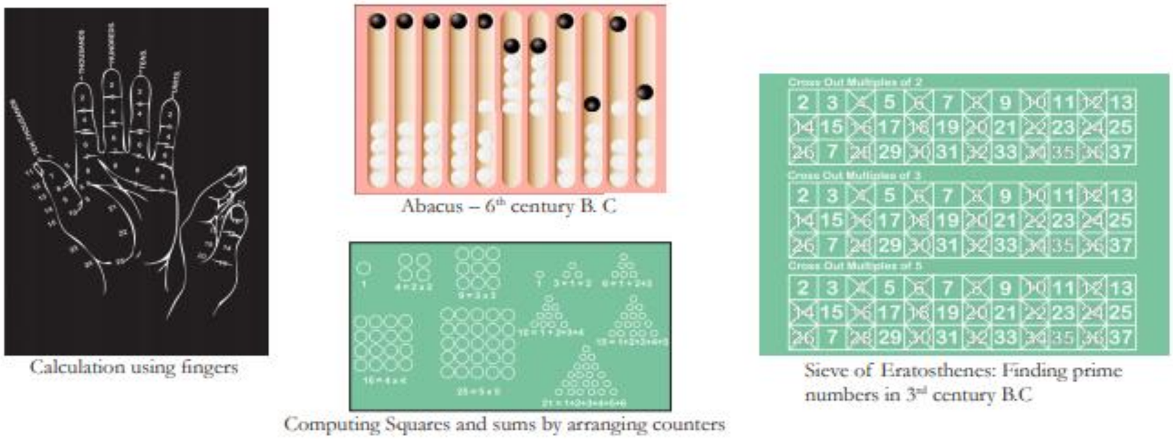
A computer works with software programs that are sent to its underlying hardware architecture for reading, interpretation and execution. Computers are classified according to computing power, capacity, size, mobility and other factors, as personal computers (PC), desktop computers, laptop computers, minicomputers, handheld computers and devices, mainframes or supercomputers.

1.2 Evolution of Computers?

The answer to the question “Who invented the computer?” is not a simple one. Because it is not a single machine but a collection of different complicated parts, so development of each part can be considered as a separate invention. Many Scientists have contributed to the history of computers. Let us go through various computing devices which were developed prior to the existing computer.

1.2.1 Abacus

Many centuries ago when man started to count the numbers, he thought of a device which can trace the numbers and thus came the existence of ABACUS. It was the first counting device which was developed in China more than 3000 years ago. The name Abacus was obtained from Greek word Abax which means slab. This device basically consists of a rectangular wooden frame and beads. The frame contains horizontal rods and the beads which have holes are passed through the rods. Counting was done by moving the beads from one end of the frame to the other.


Calculation using fingers

Abacus – 6th century B. C

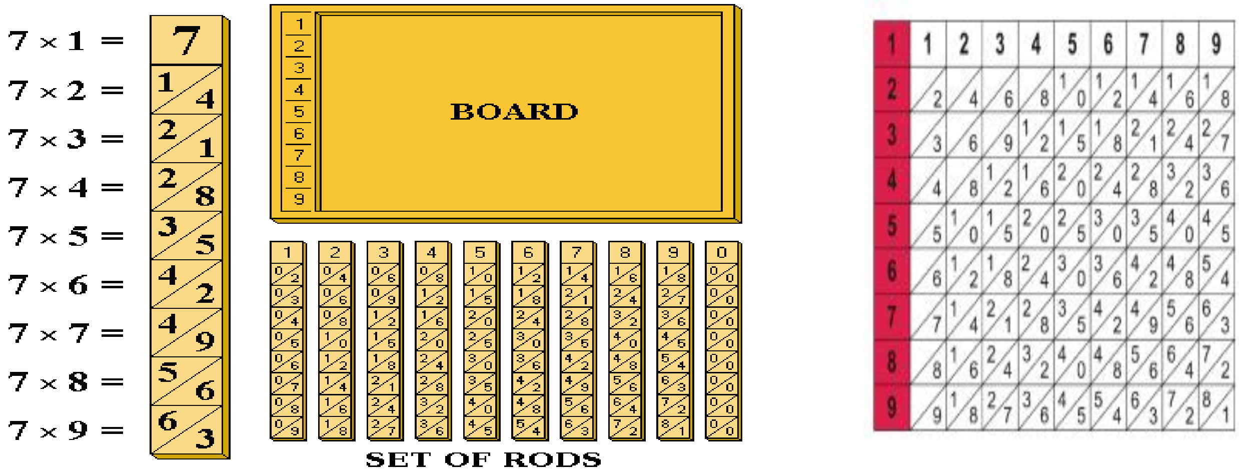
Computing Squares and sums by arranging counters

Sieve of Eratosthenes: Finding prime numbers in 3rd century B.C

Fig 1.2: Abacus

1.2.2 Napier's Bones

It is a device which contains a set of rods made of bones. It was developed by John Napier, a Scottish Mathematician and hence the device was named as Napier's bones. The device was mainly developed for performing multiplication and division. Later in 1614 he also introduced logarithms.



1	1	2	3	4	5	6	7	8	9
2	/2	/4	/6	/8	1/0	1/2	1/4	1/6	1/8
3	/3	/6	/9	1/2	1/5	1/8	2/1	2/4	2/7
4	/4	/8	1/2	1/6	2/0	2/4	2/8	3/2	3/6
5	/5	1/0	1/5	2/0	2/5	3/0	3/5	4/0	4/5
6	/6	1/2	1/8	2/4	3/0	3/6	4/2	4/8	5/4
7	/7	1/4	2/1	2/8	3/5	4/2	4/9	5/6	6/3
8	/8	1/6	2/4	3/2	4/0	4/8	5/6	6/4	7/2
9	/9	1/8	2/7	3/6	4/5	5/4	6/3	7/2	8/1

Fig 1.3: Napier's bones

1.2.3 Pascaline

Pascaline is a calculating machine developed by Blaise Pascal, a French Mathematician. It was the first device with an ability to perform additions and subtractions on whole numbers. The device is made up of interlocked cog wheels which contain numbers 0 to 9 on its circumference. When one wheel completes its rotation the other wheel moves by one segment. Pascal patented this device in 1647 and produced it on mass scale and earned a handful of money.

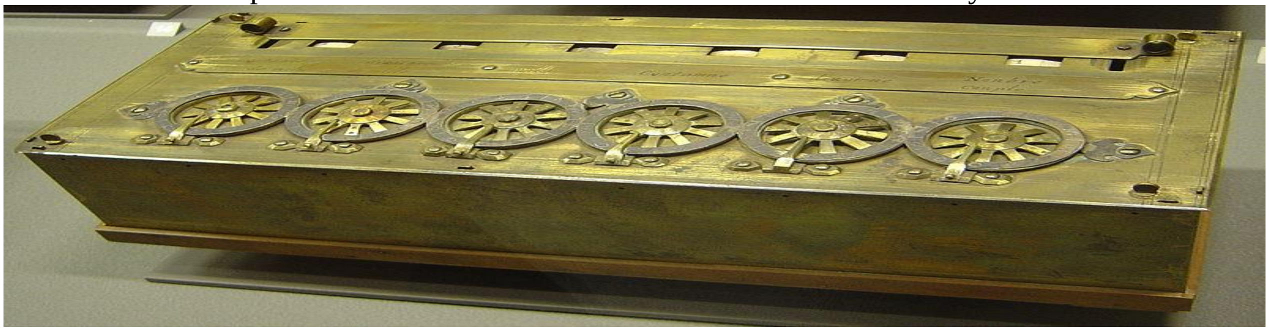


Fig 1.4: Pascaline

1.2.4 Punched Card System

Punched Card System is used for storing and retrieving data. This was invented by Herman Hollerith, an American Statistician in US Census Bureau. The system stores the data coded in the form of punched holes.

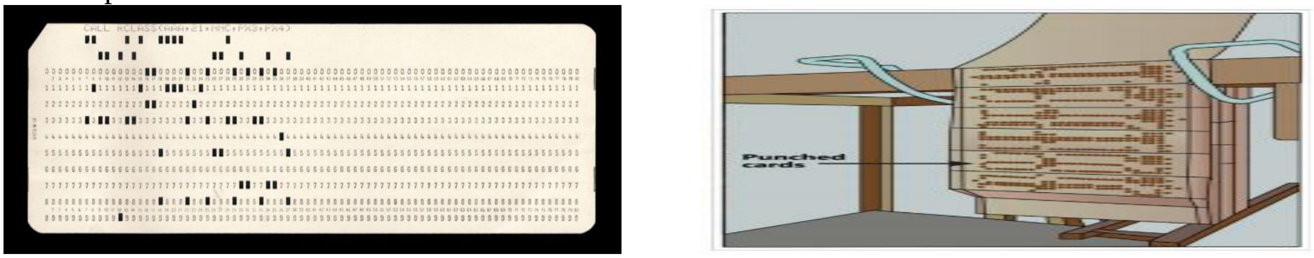


Fig 1.5: Punched Card System

The Jacquard loom invented by Joseph Marie Jacquard used punched cards to control a sequence of operations. A pattern of the loom's weave could be changed by changing the punched card.

1.2.5 Tabulator

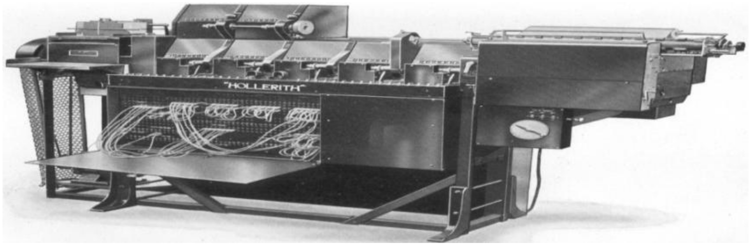


Fig 1.6: Tabulator

Herman Hollerith also invented Tabulator which was the first step towards programming. The first tabulator which he invented in 1890 was used to operate only on 1890 census cards.

As he was a statistician in census bureau, he developed devices to simplify the tasks related to his department. Later in 1906, Type 1 tabulator was developed with a plug board control panel which allowed it to do different jobs without being rebuilt. His inventions were the basis for the modern information processing industry.

1.2.6 Digital Era

- Coming to the digital era, Binary system made its entry into the computer world. According to this system, 0's and 1's were used.
- This system was suggested by Claude Shannon, an American Mathematician. The first electronic computer was built by Dr. John Vincent Atanasoff, a Physics Professor and Clifford Berry. The computer was names as ABC (Atanasoff-Berry Computer). This computer used vacuum tubes for data storage. It was designed mainly for solving systems of simultaneous equations. In 1946, General Purpose Computer was developed which contained 18000 valves and used to consume 100 kilowatts of power and weighted several tones.

1.2.7 Transistors

- In 1947, Transistors were introduced into the computers. With the introduction of transistors, computations were simpler and faster.
- In 1957, IBM developed FORTRAN.
- In 1959, Integrated Circuit(IC) came into existence which was later used in the computers.
- In 1960, Mainframe computer was designed which used IC for the first time.
- In1970, Memory chip with 1KB storage capacity was developed by Intel.

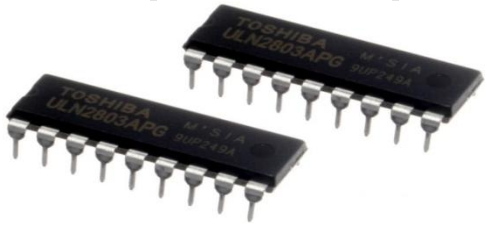


Fig 1.7: Transistors

- In 1975, First micro computer was developed by H. Edward Roberts (now the father of micro computer).
- In 1980's and 1990's, many modifications and up gradations were done and the usage of chips and various other stuffs changed the computers completely.

1.3 Generations of Computers

Computer generations classification is mainly based on the considerations are the architecture, languages, modes of operation, etc. The function performed by the computer and the speed of their operations have been changing since the old days to the most modern computer. Based on the period of development and the features incorporated, the computers are classified into different generations. The classification and time periods are given below:

1. First Generation Computer (Mid 1940 - 1950)
2. Second Generation Computer (Mid 1950 - 1960)
3. Third Generation Computer (Mid 1960 - 1970)
4. Fourth Generation Computer (Mid 1970 - onward)
5. Fifth Generation Computer (Mid 1980's onwards)

1.3.1 First Generation Computer (Mid 1940 - 1950)

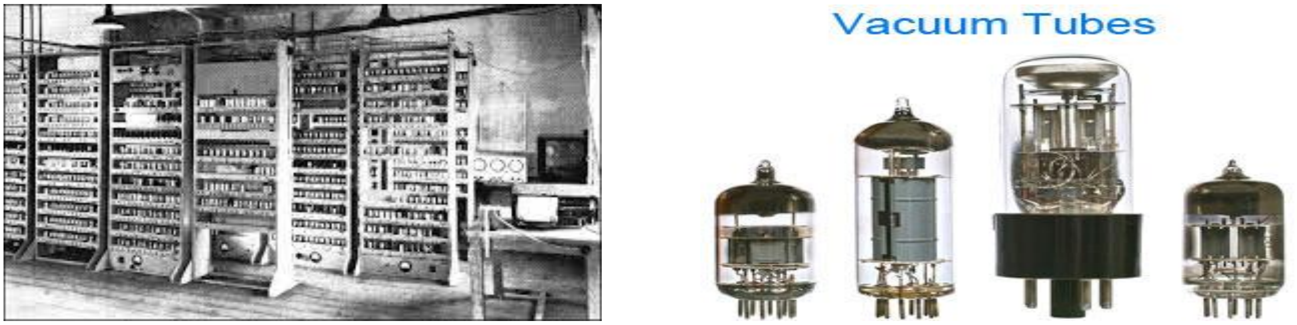


Fig 1.8: First Generation Computer

First generation computers were characterized by the fact that operating instructions were made to order for the specific task for which the computer was to be used. It was operated on the "Principle of Thermionic Emission". In the first generation computer, vacuum tubes as CPU, magnetic drum for data storage, and machines languages were used for giving instruction. The computer of this generation was very large in size called room-sized computers. The programming of first generation computers was done in machine languages (0s and 1s). Afterward, assembly languages were developed and used in first generation computer.

The main features of the first generation are −

- Vacuum tube technology
- Unreliable
- Supported machine language only
- Very costly
- Generated a lot of heat
- Slow input and output devices
- Huge size
- Need of AC
- Non-portable
- Consumed a lot of electricity

The example of first generation computers is ENIAC, UNIVAC, EDVAC, and EDSAC.

1.3.2 Second Generation Computer (Mid 1950's - 1960)

Fig 1.9: Second Generation Computer

Second generation computer replaced machine language with assembly language, allowing abbreviated programming codes to replace long, difficult binary codes. The transistor was developed in this generation. A transistor transfers electric signals across a resistor. A transistor was highly reliable compared to tubes. The transistor was far more superior in performance on account of their miniature size, smaller power consumption, and heat production rate. The second generation computer used these semiconductor devices.

Some of its features are:

- Technology used: Transistor
- Operating speed was in terms of a microsecond.
- Assembly language and machines independent language such as COBOL (Common Business Oriented Language) and FORTRAN
- (Formula Translation) were introduced the size of the computer.
- Magnetic core memory was used as primary memory.
- Magnetic drum and magnetic tape were used as secondary memory.
- Power required to operate them was low.
- It could perform scientific calculation such as solving differential equations.
- Storage capacity and use of computers are increased.

1.3.3 Third Generation Computer (Mid 1960's - 1970)

Transistors were replaced with an integrated circuit known popularly as chips. Scientist managed to fit many components on a single chip. As a result, the computer became ever smaller as more components were squeezed on the chip. IC was first designed and fabricated by Jack S Kilby at Texas Instrument and by Robert S Noyce at Fairchild independently. IC is a circuit consisting of a large number of electronic components placed on a single silicon chip by a photo-lithographic process. Magnetic disks began to replace magnetic tape for auxiliary and video display terminals were introduced for the output of data. Keyboards were used for the input of data. A new operating system was introduced for automatic processing and multi-programming. These computers were highly reliable, relatively expensive and faster. High-level programming languages continued to be a developer. The example of third generation computers is IBM-360 series, ICL-900 series, and Honeywell 200 series.

Features of the third generation computers are:

- The technology used: IC (Integrated Circuit).
- Transistors were replaced by IC in their electronic circuitry.
- High-level languages like FORTAN, BASIC and other are used to develop programs.
- Semiconductor memory like RAM and ROM were used as primary memory.

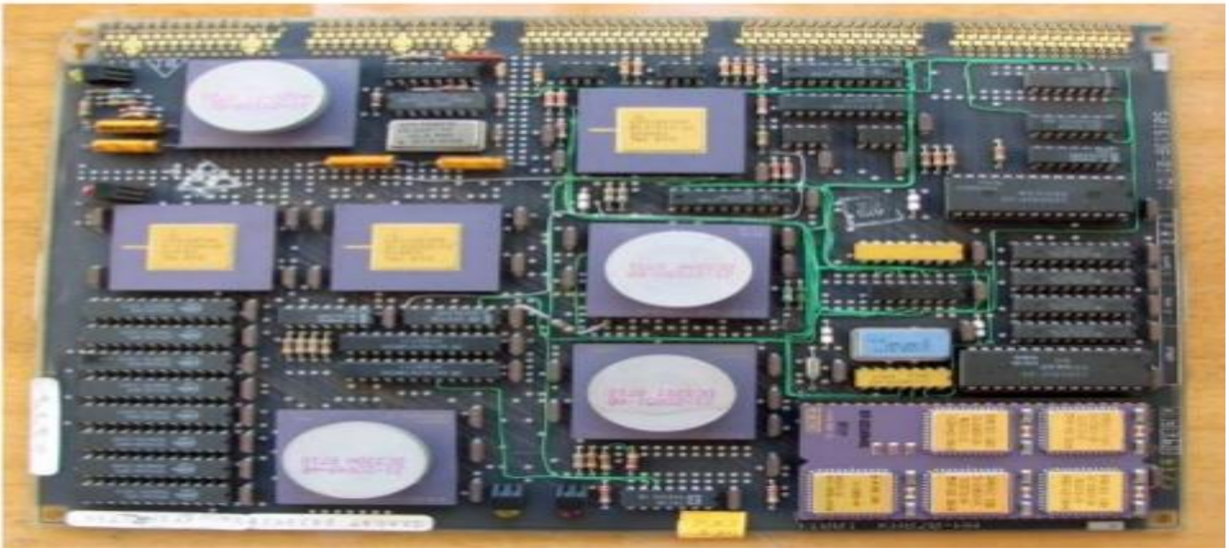


Fig 1.10: Third Generation Computer

- Monitor and keyboard were introduced for data input and output respectively.
- Multiprogramming facility was developed.
- The computer was used in census calculation, military, banks and industries.
- Size, cost, power requirement and heat generation decreased.
- Processing speed and storage capacity used of computer increased.

1.3.4 Fourth Generation Computer (Mid 1970's onward)

The invention of microprocessor chip marked the beginning of the fourth generation computers. Semiconductor memories replaced magnetic core memories. The invention of microprocessors led to the development of microcomputer or the personal computer. The first microprocessor called Intel 4004 was developed by American Intel Corporation 1971. This computer has faster generation language and application software for microcomputers became popular and allowed home and business users to adapt their computers for word processing, spreadsheet manipulating, file handing and graphics. In this generation, the concept of computer networks and CD-ROMs came into existence.

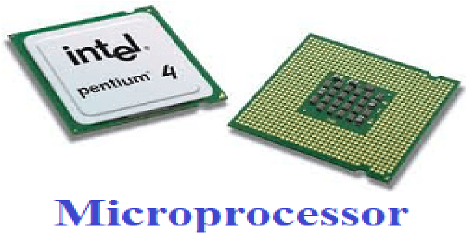


Fig 1.11: Fourth Generation Computer

Features of the fourth generation computer are:

- Technology in use: VLSI is introduced and used Microprocessor-based technology.
- Problem-oriented fourth generation language (4GL) is used to develop the program.
- Semiconductor like RAM, ROM and cache memory is used as a primary memory.
- Magnetic disks like hard disk, optical disk (CD, DVD), Blue-ray disk, flashes memory (memory chip, pen drive) are used as secondary memory.
- E-mail, Internet and mobile communication are developed.
- Advanced, user-friendly, web page software are developed.
- Size, cost, power requirement, heat generation decreased compared to the previous generation.
- Operating speed, storage capacity ,use of computer increased compared to the previous generation

The example of the fourth generation computer is IBM-PC, HP laptops, Mac notebook etc.

1.3.5 Fifth Generation Computer (Mid 1980's Present and future)

The aim is to bring machines with genuine IQ, the ability to reason logically and with real knowledge of the word. Thus, this computer will be totally different, totally novel and totally new than last four generations of computer. Fifth generation computer was based on Artificial Intelligence (AI) and that is still developing process, but not yet a reality i.e this computer is incomplete. The scientists are working on it still. These computers will be able to converse with people and will be able to mimic human sense, manual skills, and intelligence.

Features of the fifth generation computers are:

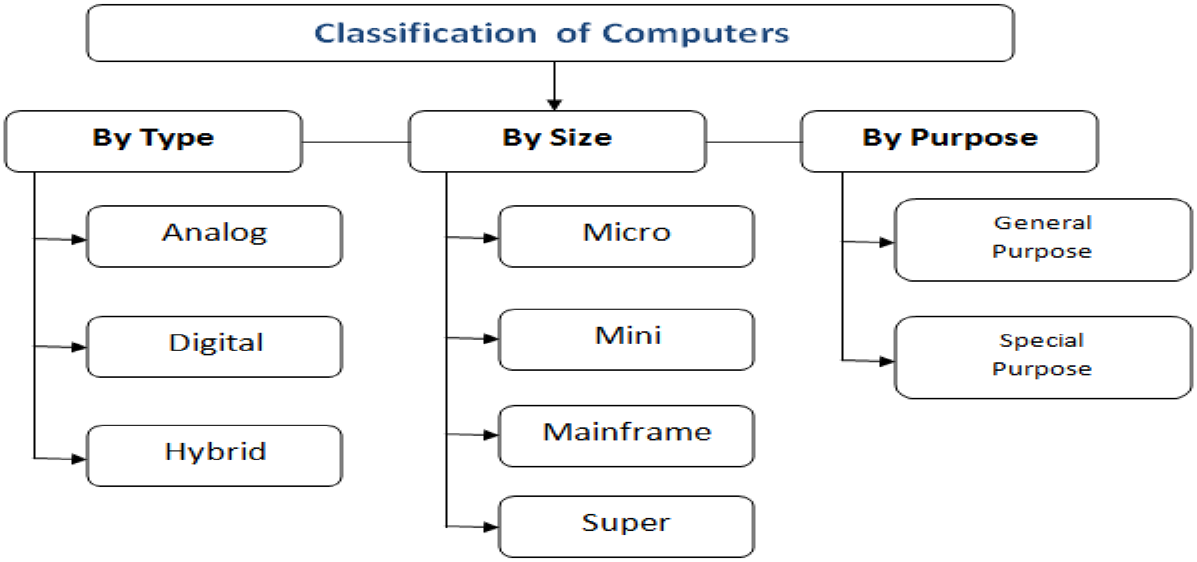
- Technology to be used: These machines will incorporate Bio-chip and VVLSI (Very Very Large Scale Integration) or Ultra-Large Scale Integration (ULSI)
- The computer will have Artificial Intelligence (AI).
- Natural language will be used to develop programs.
- The computer will have parallel processing in full fledge.
- The operating speed will be in terms of LIPS(Logical Inference per Second)
- This aim is to solve highly complex problems, which require great intelligence and expertise when solved by people.
- Quantum computation, molecular and nanotechnology will be used completely.

Fig 1.12: Fifth Generation Computer

1.4 Classification of Computers

In general computers are classified into major categories based on.

a) According to the purpose of the computer.
b) According to the operation of computer.
c) According to the size of computer.


Classification of Computers

By Type: Analog, Digital, Hybrid

By Size: Micro, Mini, Mainframe, Super

By Purpose: General Purpose, Special Purpose

Fig 1.13: Classifications of Computer

(a) Classification as per purpose of the computer

1. **General Purpose Computers**: These computers are theoretically used for any type of applications. These computers can be used in solving a business Problem and also used to solve mathematical equation with same accuracy and consistency. Most of the computers now are general purpose digital computers. All the P.C's, which have become household affair.
2. **Special Purpose Computers**: These digital computers are designed, made and used for any specific job. These are usually used for those purposes which are critical and need great accuracy and response like Satellite launching, weather forecasting etc.

(b) Classification as per operational (Data Handling) principle of the computer

1. **Analog Computers:**
   - ✓ Analog computers are those computers which take input in the form of variables and produces output in the form of graphs.
   - ✓ The results given by analog computers are approximate results, these computers deals with quantities and variables such as voltage, pressure, temperature, speed etc. Example: voltmeter
2. **Digital Computers:**
   - ✓ Digital computers are those which take input in form of digits and produces output in the form of digits only. 0's and 1's are called binary digits
   - ✓ Digital computers can give accurate and faster results.
   - ✓ Digital computers are either general purpose computers or special purpose ones. General purpose computers, as their name suggests, are designed for specific types of data processing while general purpose computers are meant for general use.
3. **Hybrid Computers:** These computers are a combination of both digital and analog computers. In this type of computers, the digital segments perform process control by conversion of analog signals to digital ones. It is used to measure BP, temperature in hospital.

(c) Classification as per sizes and functionality of the computer

1. **Supercomputers:** The highly calculation-intensive tasks can be effectively performed by means of supercomputers. Quantum physics, mechanics, weather forecasting, molecular theory are best studied by means of supercomputers. Their ability of parallel processing and their well-designed memory hierarchy give the supercomputers, large transaction processing powers. Ex. PARAM developed in India.
2. **Servers:** They are computers designed to provide services to client machines in a computer network. They have larger storage capacities and powerful processors. Running on them are programs that serve client requests and allocate resources like memory and time to client machines. Usually they are very large in size, as they have large processors and many hard drives. They are designed to be fail-safe and resistant to crash.
3. **Mainframe Computers:** Large organizations use mainframes for highly critical applications such as bulk data processing and ERP. Most of the mainframe computers have capacities to host multiple operating systems and operate as a number of virtual machines. They can substitute for several small servers.
4. **Wearable Computers:** A record-setting step in the evolution of computers was the creation of wearable computers. These computers can be worn on the body and are often used in the study of behavior modeling and human health. Military and health professionals have incorporated wearable computers into their daily routine, as a part of such studies. When the users' hands and sensory organs are engaged in other activities, wearable computers are of great help in tracking human actions.

5. **Minicomputers:** In terms of size and processing capacity, minicomputers lie in between mainframes and microcomputers. Minicomputers are also called mid-range systems or workstations. The term began to be popularly used in the 1960s to refer to relatively smaller third generation computers. They took up the space that would be needed for a refrigerator or two and used transistor and core memory technologies. The 12-bit PDP-8 minicomputer of the Digital Equipment Corporation was the first successful minicomputer.
6. **Microcomputers:** A computer with a microprocessor and its central processing unit is known as a microcomputer. They do not occupy space as much as mainframes do. When supplemented with a keyboard and a mouse, microcomputers can be called personal computers. A monitor, a keyboard and other similar input-output devices, computer memory in the form of RAM and a power supply unit come packaged in a microcomputer. These computers can fit on desks or tables and prove to be the best choice for single-user tasks.
7. **Desktops:** A desktop is intended to be used on a single location. The spare parts of a desktop computer are readily available at relatively lower costs. Power consumption is not as critical as that in laptops. Desktops are widely popular for daily use in the workplace and households.
8. **Laptops:** Similar in operation to desktops, laptop computers are miniaturized and optimized for mobile use. Laptops run on a single battery or an external adapter that charges the computer batteries. They are enabled with an inbuilt keyboard, touch pad acting as a mouse and a liquid crystal display. Their portability and capacity to operate on battery power have proven to be of great help to mobile users.
9. **Notebooks:** They fall in the category of laptops, but are inexpensive and relatively smaller in size. They had a smaller feature set and lesser capacities in comparison to regular laptops, at the time they came into the market. But with passing time, notebooks too began featuring almost everything that notebooks had. By the end of 2008, notebooks had begun to overtake notebooks in terms of market share and sales.
10. **Personal Digital Assistants (PDAs):** It is a handheld computer and popularly known as a palmtop. It has a touch screen and a memory card for storage of data. PDAs can also be used as portable audio players, web browsers and smart phones. Most of them can access the Internet by means of Bluetooth or Wi-Fi communication.
11. **Tablet Computers**: Tablets are mobile computers that are very handy to use. They use the touch screen technology. Tablets come with an onscreen keyboard or use a stylus or a digital pen. Apple’s iPad redefined the class of tablet computers.
12. **Workstations:** A terminal or desktop computer in a network. In this context, workstation is just a generic term for a user's machine (client machine) in contrast to a "server" or "mainframe."

1.5 Anatomy of a Computer

The Computer mainly consist the functions input, process, output and storage. These functions were described in the manner of diagram as follows. The Block diagram of computer consists mainly i.e., Input unit, CPU, Output unit, Secondary Storage unit.

1. Input: This is the process of entering data and programs in to the computer system. Therefore, the input unit takes data from us to the computer in an organized manner for processing through an input device such as keyboard, mouse, MICR, OCR, Etc.,

2. Main Memory: It is also known as internal memory. It is very fast in operation. It is used to store data and instructions. Data has to be fed into the system before the actual processing starts. It contains a part of the operating system Software, one or more execution programs being executed.

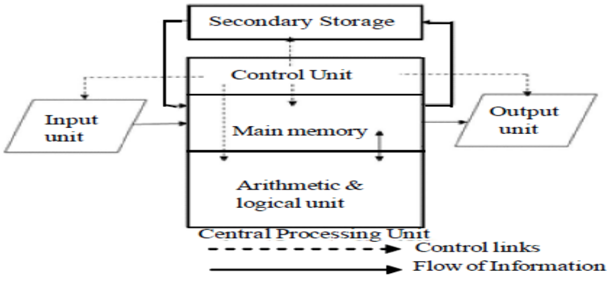


Fig 1.14: Block diagram of Computer

3. Output: This is the process of producing results from the data for getting useful information. Similarly the output produced by the computer after processing must also be kept somewhere inside the computer before being given to you in human readable form through the screen or printer. Again the output is also stored inside the computer for further processing.

4. Control Unit (CU): The next component of computer is the Control Unit, which acts like the supervisor seeing that things are done in proper fashion. Control Unit is responsible for coordinating various operations using time signal. The control unit determines the sequence in which computer programs and instructions are executed. Things like processing of programs stored in the main memory, interpretation of the instructions and issuing of signals for other units of the computer to execute them. It also acts as a switch board operator when several users access the computer simultaneously. Thereby it coordinates the activities of computer's peripheral equipment as they perform the input and output.

5. Arithmetic Logical Unit (ALU): After you enter data through the input device it is stored in the primary storage. The actual processing of the data and instruction are performed by Arithmetic Logical Unit. The major operations performed by the ALU are addition, subtraction, multiplication, division, logic and comparison. Data is transferred to ALU from storage unit when required. After processing the output is returned back to storage unit for further processing or getting stored.

6. Secondary storage: It is also known as auxiliary memory. It is closely linked with the main memory. Since main memory can't be flooded with unwanted data at particular moment, same is stored in auxiliary memory from which desired data is fed to main memory as and when required by it. Thus secondary storage is used to hold mass of information i.e., system software, application programs, cinemas, games and data files. Obviously the capacity of secondary storage is very high compared to main memory. Auxiliary memory usually in the form of Magnetic disk, Magnetic tape, CD's, Memory cards, Pen drives Etc.,

1.6 Memory revisited

The different types of memories available for a computer are shown in Fig. 1.15

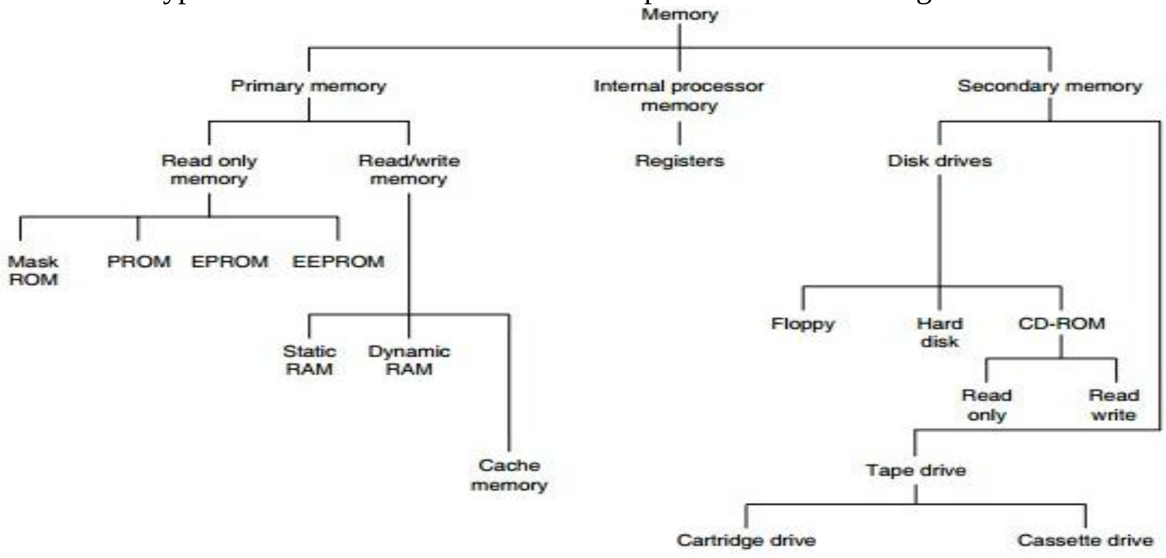


Fig 1.15: Types of memory

1.6.1 Primary Memory

All modern computers use semiconductor memory as primary memory. One of the important semiconductor memories used in desktop computers is ***Random Access Memory (RAM)***. Here "random access" means that any storage location can be accessed (both read and write) directly. This memory is faster, cheaper, and provides more storage space in lesser physical area. These very large-scale integrated semiconductor memory chips are mounted on pluggable printed circuit board (PCBs). Enhancement or replacement of memory with such PCB memory modules is easy. These characteristics have made semiconductor memory more popular and attractive.

Fig 1.16: Random Access Memory

The only drawback of semiconductor memory is that it is volatile, i.e., it loses its contents whenever power is switched off. RAM holds the data and instructions waiting to be processed by the processor. In addition to data and program's instructions, RAM also holds operating system instructions that control the basic functions of a computer system. These instructions are loaded into RAM every time the computer is turned on, and they remain there until the computer is turn off.

There are two types of RAM used in computer systems– *dynamic* and *static*.

Dynamic RAM (DRAM) is a type of RAM that employs refresh circuits to retain its content in its logic circuits. Each memory cell in DRAM consists of a single transistor. The junction capacitor of the transistor is responsible for holding the electrical charge that designates a single bit as logical 1. The absence of a charge designates a bit as logical 0. Capacitors lose their charge over time and therefore need to be recharged or refreshed at pre-determined intervals by a refreshing circuitry.

A more expensive and faster type of RAM, ***Static RAM (SRAM)***, does not require such type of refreshing circuitry. It uses between four to six transistors in a special 'flipflop' circuit that holds a 1 or 0 while the computer system is in operation. SRAM in computer systems is usually used as processor caches and as I/O buffers. Printers and liquid crystal displays (LCDs) often use SRAM to buffer images. SRAM is also widely used in networking devices, such as routers, switches, and cable modems, to buffer transmission information. The basic differences between SRAM and DRAM are listed in Table 1.1.

Static RAM	Dynamic RAM
It does not require refreshing.	It requires extra electronic circuitry that "refreshes" memory periodically; otherwise its content will be lost.
It is more expensive than dynamic RAM.	It is less expensive than static RAM.
It is lower in bit density.	It holds more bits of storage in single integrated circuit.
It is faster than dynamic RAM.	It is slower than SRAM, due to refreshing.

Table 1.1: Static RAM versus dynamic RAM

There are several popular types of dynamic RAM used in computers. They are SDRAM (Synchronous Dynamic RAM), RDRAM (Rambus Dynamic RAM) and DDR RAM (Double Data Rate RAM). The SDRAM used to be the most common type of RAM for personal computers. It was reasonably fast and inexpensive. It is no more used in the present day for personal computers as much improved RAMs are available now. The RDRAM was developed by Rambus Corporation and is its proprietary technology. It is also the most expensive RAM and is used mostly in video interface cards and high-end computers that require fast computation speed and data transfer. RDRAMs are preferred for high-performance personal computers. The DDR RAM is a refinement of SDRAM. DDR stands for *Double Data Rate*. It gives faster performance by transmitting data on both the rising and the falling edges of each clock pulse. DDR2, DDR3 are other higher-speed versions of DDR RAM. Another type of RAM, termed Video RAM (VRAM), is used to store image data for the visual display monitor. All types of video RAM are special arrangements of dynamic RAM (DRAM). Its purpose is to act as a data storage buffer between the processor and the visual display unit.

There is a persistent mismatch between processor and main memory speeds. The processor executes an instruction faster than the time it takes to read from or write to memory. In order to improve the average memory access speed or rather to optimize the fetching of instructions or data so that these can be accessed faster when the CPU needs it, cache memory is logically positioned between the internal processor memory (registers) and main memory. The cache memory holds a subset of instructions and data values that were recently accessed by the CPU. Whenever the processer tries to access a location of memory, it first checks with the cache to determine if it is already present in it. If so, the byte or word is delivered to the processor. In such a case, the processor does not need to access the main memory. If the data is not there in the cache, then the processer has to access the main memory. The block of main memory containing the data or instruction is read into the cache and then the byte or word is delivered to the processor. There are two levels of cache.

Level 1 (Primary) cache: This type of cache memory is embedded into the processor itself. This cache is very fast and its size varies generally from 8 KB to 64 KB.

Level 2 (Secondary) cache: Level 2 cache is slightly slower than L1 cache. It is usually 64 KB to 2 MB in size. Level 2 cache is also sometimes called external cache because it was external to the processor chip when it first appeared.

Read Only Memory (ROM)

It is another type of memory that retains data and instructions stored in it even when the power is turned off. ROM is used in personal computers for storing start-up instructions provided by the manufacturer for carrying out basic operations such as bootstrapping in a PC, and is programmed for specific purposes during their fabrication. ROMs can be written only at the time of manufacture. Another similar memory, Programmable ROM (PROM), is also non-volatile and can be programmed only once by a special device.

But there are instances where the read operation is performed several times and the write operation is performed more than once though less than the number of read operations and the stored data must be retained even when power is switched off.

Fig 1.17: Read Only Memory

This led to the development of EPROMs (Erasable Programmable Read Only Memories). In the EPROM or Erasable Programmable Read Only Memory, data can be written electrically. The write operation, however, is not simple. It requires the storage cells to be erased by exposing the chip to ultraviolet light, thus bringing each cell to the same initial state. This process of erasing is time consuming. Once all the cells have been brought to the same initial state, the write operation on the EPROM can be performed electrically. There is another type of Erasable PROM known as Electrically Erasable Programmable Read Only Memory (EEPROM). Like the EPROM, data can be written onto the EEPROM by electrical signals and retained even when power is switched off. The data stored can be erased by electrical signals. However, in EEPROMs the writing time is considerably higher than reading time. The biggest advantage of EEPROM is that it is non-volatile memory and can be updated easily, while the disadvantages are the high cost and the write operation takes considerable time.

1.6.2 Secondary Memory

There are four main types of secondary storage devices available in a computer system:

- Disk drives
- CD drives (CD-R, CD-RW, and DVD)
- Tape drives
- USB flash drives.

Hard disk, floppy disk, compact disk (CD), Digital Versatile Disk (DVD) and magnetic tapes are the most common secondary storage mediums. Hard disks provide much faster performance and have larger capacity, but are normally not removable; that is, a single hard disk is permanently attached to a disk drive. Floppy disks, on the other hand, are removable, but their performance is far slower and their capacity far smaller than those of hard disks. It is called ROM because information is stored permanently when the CD is created. Devices for operating storage mediums are known as *drives*. Most of the drives used for secondary memory are based on electro-mechanical technology. Mechanical components move much more slowly than do electrical signals. That's why access to secondary memory is much slower than access to main memory.

The **floppy disk** is a thin, round piece of plastic material, coated with a magnetic medium on which information is magnetically recorded, just as music is recorded on the surface of plastic cassette tapes. The flexible floppy disk is enclosed inside a sturdier, plastic jacket to protect it from damage. The disks used in personal computers are usually 3½ inches in diameter and can store 1.44 MB of data. Earlier PCs sometimes used 5¼ inch disks. The disks store information and can be used to exchange information between computers. The floppy disk drive stores data on and retrieves it from the magnetic material of the disk, which is in the form of a disk. It has two motors one that rotates the disk media and the other that moves two read-write heads, each on either surface of the disk, forward Floppy Disk Drive or backward.

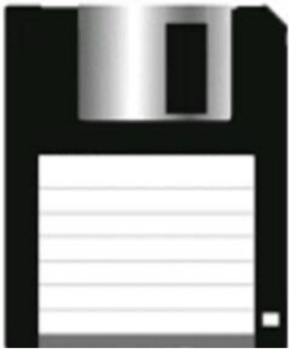


Fig 1.18: Floppy Disk

A **hard disk** is a permanent memory device mounted inside the system unit. Physically, a hard disk consists of one or more metal (sometimes aluminum) platters, coated with a metal oxide that can be magnetized. The platters are all mounted on a spindle, which allows them to spin at a constant rate. Read/write heads are attached to metal arms and positioned over each of the platter surfaces. The arms can move the read/write heads radially inwards and outwards over the surfaces of the platters (see Fig. 1.20). Data and programs are stored on the hard disk by causing the write heads to make magnetic marks on the surfaces of the platters. Read heads retrieve the data by sensing the magnetic marks on the platters. The surface of each platter is divided into concentric rings called tracks. The tracks form concentric circles on the platter's surface. Each track is divided into a certain number of sectors. A sector is capable of generally 512 bytes or sometimes 1,024 bytes of data. The head is mounted on an arm, which moves or seeks from track to track. The vertical group of tracks at the same position on each surface of each platter is called a cylinder. Cylinders are important, because all heads move at the same time. Once the heads arrive at a particular track position, all the sectors on the tracks that form a cylinder can be read without further arm motion. The storage capacity of a hard disk is very large and expressed in terms of gigabytes (GB). The data that is stored on the hard disk remains there until it is erased or deleted by the user.

The hard disk drive provides better performance and become mandatory for computer systems for the following reasons:

- Higher capacity of data storage
- Faster access time of data
- Higher data transfer rates
- Better reliability of operation
- Less data errors or data loss

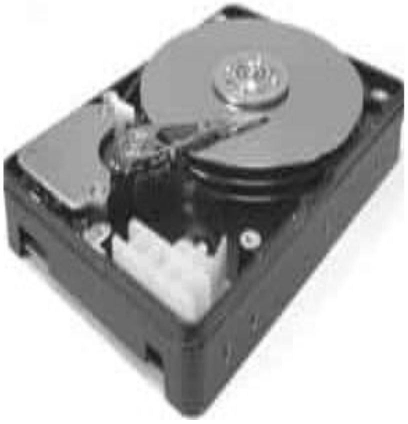


Fig 1.19: Hard Disk

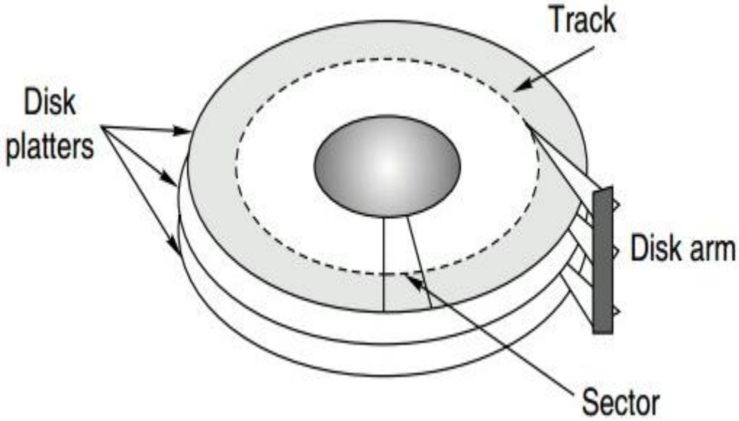


Fig 1.20: Hard Disk Organization

A **CD** is a portable secondary storage medium. Various types of CDs are available: CD-R and CD-RW. CD-RW drives are used to create and read both CD-R and CD-RW discs. Once created (i.e. when it has been "burned"), data stored on CD-R (CDRecordable) disc can't be changed. On the other hand, a CDRewritable (CD-RW) disc can be erased and reused. This disk is made of synthetic resin that is coated with a reflective material, usually aluminum. When information is written by a CD-writer drive, some microscopic pits are created on the surface of the CD. The information bit on a CD-ROM surface is coded in the form of ups and downs (known as pits and dumps), created by infrared heat. There is one laser diode on the reading head. The bits are read by shining a low - intensity laser beam onto the spinning disc. The laser beam reflects strongly from a smooth area on the disc but weakly from a pitted area. A sensor receiving the reflection determines whether each bit is a 1 or a 0 accordingly.

Fig 1.21: CD / DVD

CDs were initially a popular storage media for music; they were later used as general computer storage media. Most personal computers are equipped with a CD-Recordable (CD-R) drive. A CD-Rewritable (CD-RW) disc can be reused because the pits and flat surfaces of a normal CD are simulated on a CDRW by coating the surface of the disc with a material that, when heated to one temperature becomes amorphous (and therefore non-reflective) and when heated to a different temperature becomes crystalline (and therefore reflective).

1.7 Introduction to Operating systems

- An operating system acts as an interface between the user of a computer and computer hardware.
- The purpose of an operating system is to provide an environment in which a user can execute programs in a convenient and efficient manner.
- An operating system is a program that controls the execution of application programs and acts as an interface between the user of a computer and the computer hardware.
- An operating system is concerned with the allocation of resources and services, such as memory, processors, devices, and information. The operating system correspondingly includes programs to manage these resources, such as a traffic controller, a scheduler, memory management module, I/O programs, and a file system.

Operating system as User Interface –

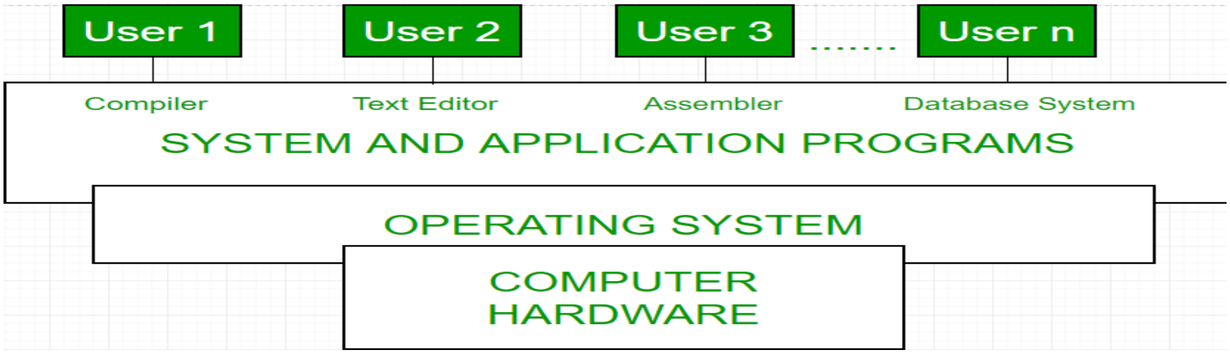


Fig 1.22: Conceptual view of a computer system

1. User
2. System and application programs
3. Operating system
4. Hardware

Every general purpose computer consists of the hardware, operating system, system programs, and application programs. The hardware consists of memory, CPU, ALU, and I/O devices, peripheral device, and storage device. System program consists of compilers, loaders, editors, OS, etc. The application program consists of business programs, database programs. Every computer must have an operating system to run other programs. The operating system coordinates the use of the hardware among the various system programs and application programs for various users. It simply provides an environment within which other programs can do useful work. The operating system is a set of special programs that run on a computer system that allows it to work properly.

It performs basic tasks such as recognizing input from the keyboard, keeping track of files and directories on the disk, sending output to the display screen and controlling peripheral devices.

The Operating system must support the following tasks. The tasks are:

1. Provides the facilities to create, modification of programs and data files using an editor.
2. Access to the compiler for translating the user program from high level language to machine language.
3. Provide a loader program to move the compiled program code to the computer's memory for execution.
4. Provide routines that handle the details of I/O programming.

The common functions of an operating system includes –

1. **Process (or) management:** A process is basically a program in execution. The operating system decides which process gets to run, for how long and perhaps at what priority.
2. **Memory management:** Operating system is responsible for keeping track of which parts of the memory are currently being used and by whom. And it allocates the memory when a process or program requests it. Often several programs may be in memory at the same time. The operating system selects processes that are to be placed in memory, where they are to be placed, and how much memory is to be given to each.
3. **Device management:** The operating system allocates the various devices to the processes and initiates the I/O operation. It also controls and schedules accesses to the input/output devices among the processes.
4. **File management:** A file is just a sequence of bytes. Files are storage areas for programs, source codes, data, documents etc. The operating system keeps track of every file in the system. The file system is an operating system module that allows users and programs to create, delete, modify, open, close, and apply other operations to various types of files. It also allows users to give names to files, to organize the files hierarchically into directories, to protect files, and to access those files using the various file operations.
5. **Security:** Prevents unauthorized access to programs and data by means of passwords and other similar techniques.
6. **Job Accounting:** Keeps track of time and resources used by various jobs and/or users.
7. **Error-detecting:** Production of dumps, traces, error messages, and other debugging and error-detecting methods.
8. **Coordination Between Other Software and Users:** Coordination and assignment of compilers, interpreters, assemblers, and other software to the various users of the computer systems.

Types of Operating System

- Batch Operating System- Sequence of jobs in a program on a computer without manual interventions.
- Time sharing operating System- allows many users to share the computer resources.(Max utilization of the resources).
- Distributed operating System- Manages a group of different computers and make appear to be a single computer.
- Network operating system- computers running in different operating system can participate in common network (It is used for security purpose).
- Real time operating system – meant applications to fix the deadlines.

Advantages of Operating system –

1. **Convenience:** An OS makes a computer more convenient to use.
2. **Efficiency:** An OS allows the computer system resources to be used in an efficient manner.
3. **Ability to Evolve:** An OS should be constructed in such a way as to permit the effective development, testing and introduction of new system functions without at the same time interfering with service.

Examples of Operating System are –

- Windows (GUI based, PC)
- GNU/Linux (Personal, Workstations, ISP, File and print server, Three-tier client/Server)
- macOS (Macintosh), used for Apple’s personal computers and work stations (MacBook, iMac).
- Android (Google’s Operating System for smartphones/tablets/smartwatches)
- iOS (Apple’s OS for iPhone, iPad and iPod Touch)

1.8 Operational overview of a CPU

- Any processing executed by central processing unit is directed by the instruction. The processing required for a single instruction is called an *instruction cycle*.
- The four steps which the CPU carries out for each machine language instruction are *fetch, decode, execute, and store* (Fig. 1.23).

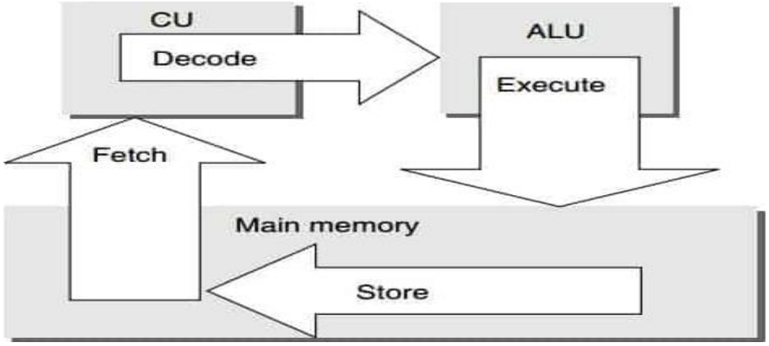


Fig 1.23: A Simplified view of an instruction cycle

The steps involved in the instruction cycle while executing a program are described below. The Program Counter (PC) is the register that keeps track of what instruction has to be executed next. At the first step, the instruction is *fetched* from main memory and loaded into Instruction Register (IR), whose address is specified by PC register.

Immediately the PC is incremented so that it points to the next instruction in the program. Once in IR, the instruction is *decoded* to determine the actions needed for its execution. The control unit then issues the sequence of control signals that enables *execution* of the instruction. Once in IR, the instruction is *decoded* to determine the actions needed for its execution. Data needed to be processed by the instructions are either fetched from a register from RAM through an *address register*. The result of the instruction is *stored* (written) to either a register or a memory location. The next instruction of a program will follow the same steps. This will continue until there is no more instruction in the program or the computer is turned off, some sort of unrecoverable error occurs.

Register: A register is a single, permanent storage location within the CPU used for a particular, defined purpose. CPU contains several important registers such as:

- The *program counter (PC)* register holds the address of the current instruction being executed.
- The *instruction register (IR)* holds the actual instruction being executed currently by the computer. To access data in memory, CPU makes use of two internal registers:
- The *memory address register* (MAR) holds the address of a memory location.
- The *memory data register* (MDR), sometimes known as the memory buffer register, will hold a data value that is being stored to or retrieved from the memory location currently addressed by the memory address register.

Exercise:

1. What is a computer? Explain in detail about evolution of computers?
2. Explain briefly about anatomy of computer?
3. What are the generations of computers? Explain in detail about all?
4. Differentiate between the RAM and ROM?
5. Explain the working of CPU?
6. Briefly about the Operating Systems?
7. Explain in detail about the classify computers?

Introduction to Programming, Algorithms and Flowcharts: Programs and Programming, Programming languages, Compiler, Interpreter, Loader, Linker, Program execution, Fourth generation languages, Fifth generation languages, Classification of Programming languages, Structured programming concept, Algorithms, Pseudo-code, Flowcharts, Strategy for designing algorithms, Tracing an algorithm to depict logic, Specification for converting algorithms into programs.

* * * * *

1.9 Programs and Programming

- A program is a set of logically related instructions that is arranged in a sequence to solve a particular problem.
- The process of writing a program is called programming.
- It is a necessary and critical step in data processing.

An incorrect program delivers results that cannot be used. There are two ways by which one can acquire a program—either purchase an existing program, referred to as *packaged software* or prepare a new program from scratch, in which case it is called *customized software*.

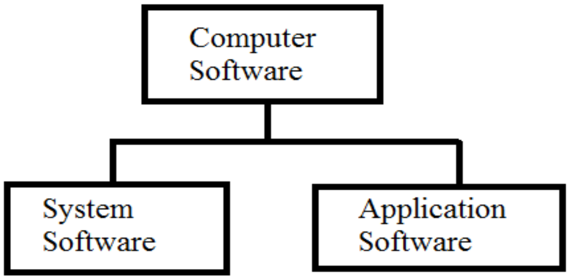


Fig 1.24: Computer Software Classification

Computer software can be broadly classified into two categories: system software and application software.

1.9.1 System Software

System software is a collection of programs that interfaces with the hardware. Some common categories of system software are described as follows.

Language translator: It is system software that transforms a computer program written by a user into a form that can be understood by the machine.

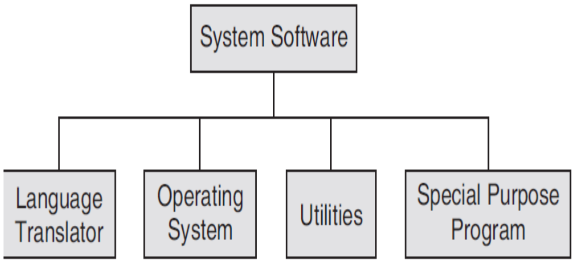


Fig 1.25: Categories of System Software

Operating system (OS): This is the most important system software that is required to operate a computer system. An operating system manages the computer's resources effectively, takes care of scheduling multiple jobs for execution, and manages the flow of data and instructions between the input/output units and the main memory. Since then a number of operating systems have been developed and some have undergone several revisions and modifications to achieve better utilization of computer resources. Advances in computer hardware have helped in the development of more efficient operating systems.

1.9.2 Application Software

- Application software is written to enable the computer to solve a specific data processing task. There are two categories of application software: pre-written software packages and user application programs.
- A number of powerful application software packages that do not require significant programming knowledge have been developed. These are easy to learn and use compared to programming languages.

- A user application program may be written using one of these packages or a programming language. The most important categories of software packages available are:
  - Database management software
  - Spreadsheet software
  - Word processing, Desktop Publishing (DTP), and presentation software
  - Multimedia software
  - Data communication software
  - Statistical and operational research software

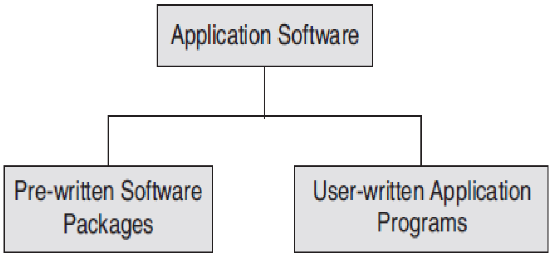

Application Software
Pre-written Software Packages
User-written Application Programs

Fig 1.26: Categories of application software

1.10 Programming languages

- To write a computer program, a standard programming language is used. A programming language is composed of a set of instructions in a language understandable to the programmer and recognizable by a computer.
- Programming languages can be classified as high-level, middle-level, and low-level. High-level languages such as BASIC, COBOL (Common Business Oriented Programming Language), and FORTRAN (Formula Translation Language) are used to write application programs. A middle-level language such as C is used for writing application and system programs. A low-level language such as the assembly language is mostly used to write system programs.
- Low-level programming languages were the first category of programming languages to evolve. Gradually, high-level and middle-level programming languages were developed and put to use.

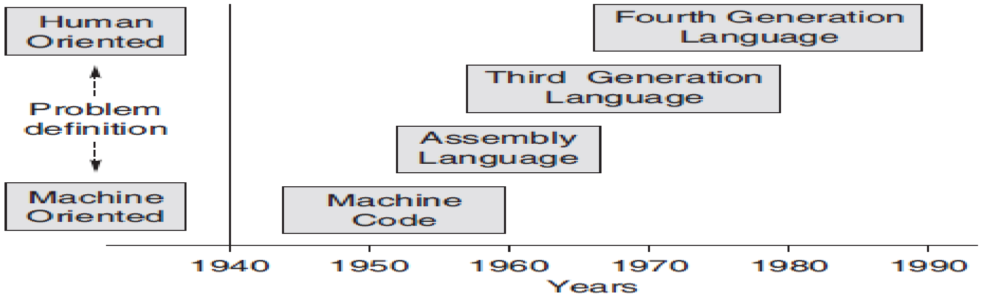


Fig 1.27: Growth of computer languages

1.10.1 System Programming Languages

System programs or software's are designed to make the computer easier to use. An example of system software is an operating system consisting of many other programs that control input/output devices, memory, and processor, schedule the execution of multiple tasks, etc. For example, instructions that move data from one location of storage to a register of the processor are required. Assembly language, which has a one-to-one correspondence with machine code, was the normal choice for writing system programs like operating systems. But, today C is widely used to develop system software.

1.10.2 Application Programming Languages

There are two main categories of application programs: *business programs* and *scientific application programs*. Application programs are designed for specific computer applications, such as payroll processing and inventory control. To write programs for payroll processing or other such applications, the programmer does not need to control the basic circuitry of a computer. Instead, the programmer needs instructions that make it easy to input data, produce output, perform calculations, and store and retrieve data. Most programming languages are designed to be good for one category of applications but not necessarily for the other, although there are some general-purpose languages that support both types. Business applications are characterized by processing of large inputs and high-volume data storage and retrieval but call for simple calculations. Languages which are suitable for business program development must support high-volume input, output, and storage but do not need to support complex calculations. On the other hand, programming languages designed for writing scientific C programs contain very powerful instructions for calculations but have poor instructions for input, output, etc. Among the traditionally used programming languages, COBOL is more suitable for business applications whereas FORTRAN is more suitable for scientific applications.

1.10.3 Low-level Languages

A low-level computer programming language is one that is closer to the native language of the computer, which is 1’s and 0’s.

Machine language

This is a sequence of instructions written in the form of binary numbers consisting of 1’s and 0’s to which the computer responds directly. The machine language is also referred to as the machine code, although the term is used more broadly to refer to any program text. A machine language instruction generally has three parts as shown in Fig. 1.28. The first part is the command or operation code that conveys to the computer what function has to be performed by the instruction. All computers have operation codes for functions such as adding, subtracting and moving. The second part of the instruction either specifies that the operand contains data on which the operation has to be performed or it specifies that the operand contains a location, the contents of which have to be subjected to the operation. Machine language is considered to be the *first generation language* (1GL).

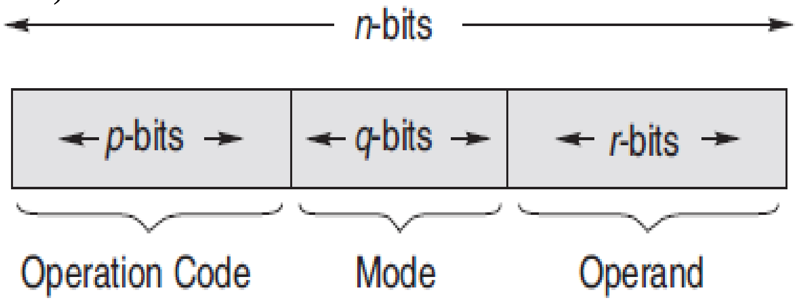


Fig 1.28: General format of machine language instruction

Advantage of machine language

- ✓ It is easily understood by the computer
- ✓ The time taken to execute the program is very less
- ✓ The computation speed is very high

Disadvantages of machine language

- ***Difficult to use:*** It is difficult to understand and develop a program using machine language. For anybody checking such a program, it would be difficult to forecast the output when it is executed. Nevertheless, computer hardware recognizes only this type of instruction code.
- ***Machine dependent:*** The programmer has to remember machine characteristics while preparing a program. As the internal design of the computer is different across types, which in turn is determined by the actual design or construction of the ALU, CU, and size of the word length of the memory unit, the machine language also varies from one type of computer to another.
- ***Error prone:*** It is hard to understand and remember the various combinations of 1’s and 0’s representing data and instructions. This makes it difficult for a programmer to concentrate fully on the logic of the problem, thus frequently causing errors.
- ***Difficult to debug and modify*** Checking machine instructions to locate errors are about as tedious as writing the instructions. Further, modifying such a program is highly problematic.

Following is an example of a machine language program for adding two numbers.

Ex. 1:

Machine Code	**Comments**
0011 1100	Load *A* register with value 7
0000 0111	
0000 0110	Load *B* register with 10
0000 1010	
1000 0000	A = A + B
0011 1010	Store the result into the memory location whose address is 100
0110 0110	
0111 0110	Halt processing

Assembly language

- This is also considered to be a *second generation language* (2GL).
- When symbols such as letters, digits, or special characters are employed for the operation, operand, and other parts of the instruction code, the representation is called an assembly language instruction.
- Such representations are known as mnemonic codes; they are used instead of binary codes. A program written with mnemonic codes forms an assembly language program.
- Machine and assembly languages are referred to as low-level languages since the coding for a problem is at the individual instruction level. Each computer has its own assembly language that is dependent upon the internal architecture of the processor.
- An *assembler* is a translator that takes input in the form of the assembly language program and produces machine language code as its output.

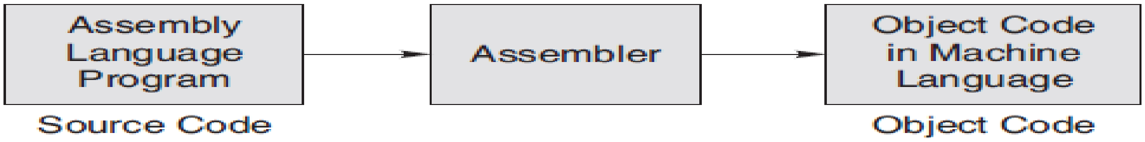
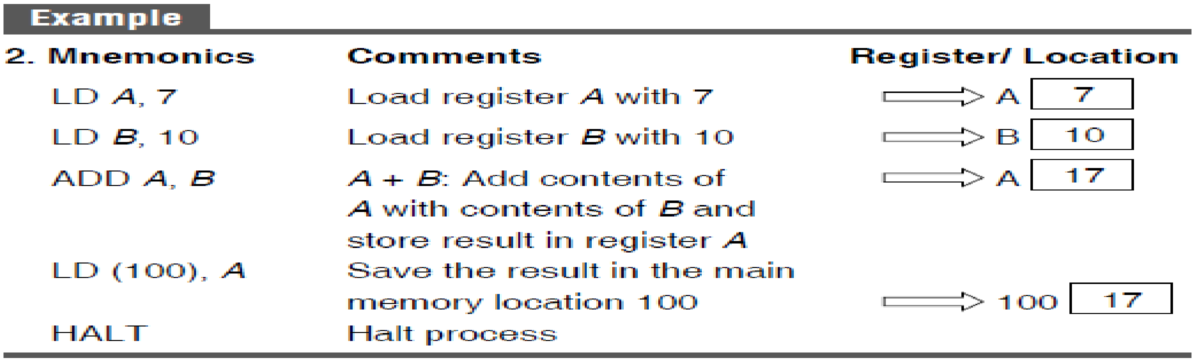


Fig 1.29: Assembler

The following is an example of an assembly language program for adding two numbers *X* and *Y* and storing the result in some memory location.



Example

2. Mnemonics	**Comments**	**Register/ Location**
LD *A*, 7	Load register *A* with 7	⇨ A 7
LD *B*, 10	Load register *B* with 10	⇨ B 10
ADD *A*, *B*	*A* + *B*: Add contents of *A* with contents of *B* and store result in register *A*	⇨ A 17
LD (100), *A*	Save the result in the main memory location 100	⇨ 100 17
HALT	Halt process	

Advantage of assembly language

- ✓ Writing a program in assembly language is more convenient than writing one in machine language.
- ✓ Instead of binary sequence, as in machine language, a program in assembly language is written in the form of symbolic instructions. This gives the assembly language program improved readability

Disadvantages of assembly language

- Assembly language is specific to particular machine architecture, i.e., machine dependent. Assembly languages are designed for a specific make and model of a microprocessor. This means that assembly language programs written for one processor will not work on a different processor if it is architecturally different. That is why an assembly language program is not portable.

- Programming is difficult and time consuming.
- The programmer should know all about the logical structure of the computer.

1.10.4 High-level Languages

- High-level programming languages such as COBOL, FORTRAN, and BASIC have instructions that are similar to human languages and have a set grammar that makes it easy for a programmer to write programs and identify and correct errors in them.
- The time and cost of creating machine and assembly language programs were quite high. This motivated the development of high-level languages.
- To illustrate this point, a program written in BASIC, to obtain the sum of two numbers, is shown below.

Ex 3.	**Stmt. No.**	**Program stmnt**	**Comments**
	10	LET X = 7	Put 7 into X
	20	LET Y = 10	Put 10 into Y
	30	LET SUM = X + Y	Add values in X and Y and put in SUM.
	40	PRINT SUM	Output the content in SUM.
	50	END	Stop

The time and cost of creating machine and assembly language programs were quite high. This motivated the development of high-level languages.

Advantages of high-level programming languages

- ***Readability*** Programs written in these languages are more readable than those written in assembly and machine languages.
- ***Portability*** High-level programming languages can be run on different machines with little or no change. It is, therefore, possible to exchange software, leading to creation of program libraries.
- ***Easy debugging*** Errors can be easily detected and removed.
- ***Ease in the development of software*** Since the commands of these programming languages are closer to the English language, software can be developed with ease.
- High-level languages are also called *third generation languages* (3GLs).

1.11 Compiler, Interpreter, Loader, Linker

For executing a program written in a high-level language, it must be first translated into a form the machine can understand. This is done by software called the *compiler*. The compiler takes the high-level language program as input and produces the machine language code as output for the machine to execute the program. This is illustrated in Fig. 1.30.

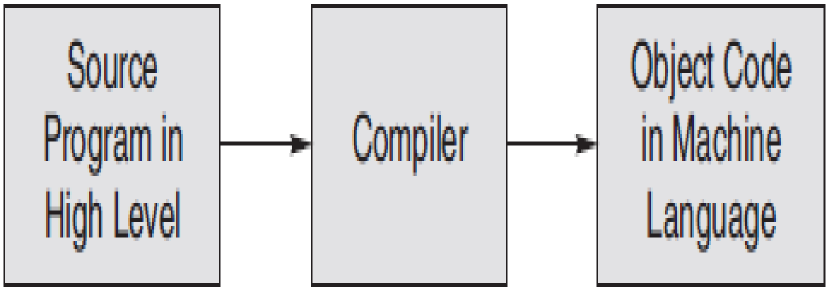


Fig 1.30: Compiler Action

During the process of translation, the compiler reads the source program statement-wise and checks for syntax errors. In case of any error, the computer generates a printout of the same. This action is known as *diagnostics*.

Compiler	Interpreter
Scans the entire program before translating it into machine code.	Translated and executes the program line by line.
Converts the entire program to machine code and executes program only when all the syntax errors are removed.	The interpreter executes one line at a time, after checking and correcting its syntax errors and then converting it to machine code.
Slow in debugging or removal of mistakes from a program.	Good for fast debugging.
Program execution time is less.	Program execution time is more.

Table 1.2: Differences between a compiler and an Interpreter

1.11.1 Compiling and Executing High-level Language Programs

1.Compiling:

- The compiling process consists of two steps: the analysis of the source program and the synthesis of the object program in the machine language of the specified machine.
- The analysis phase uses the exact description of the source programming language. A source language is described using ***lexical*** **rules,** ***syntax*** **rules, and** ***semantic*** **rules.**
- **Lexical rules** specify the valid syntactic elements or words of the language. **Syntax rules** specify the way in which valid syntactic elements are combined to form the statements of the language. Syntax rules are often described using a notation known as BNF (Backus Naur Form) grammar. **Semantic rules** assign meanings to valid statements of the language.

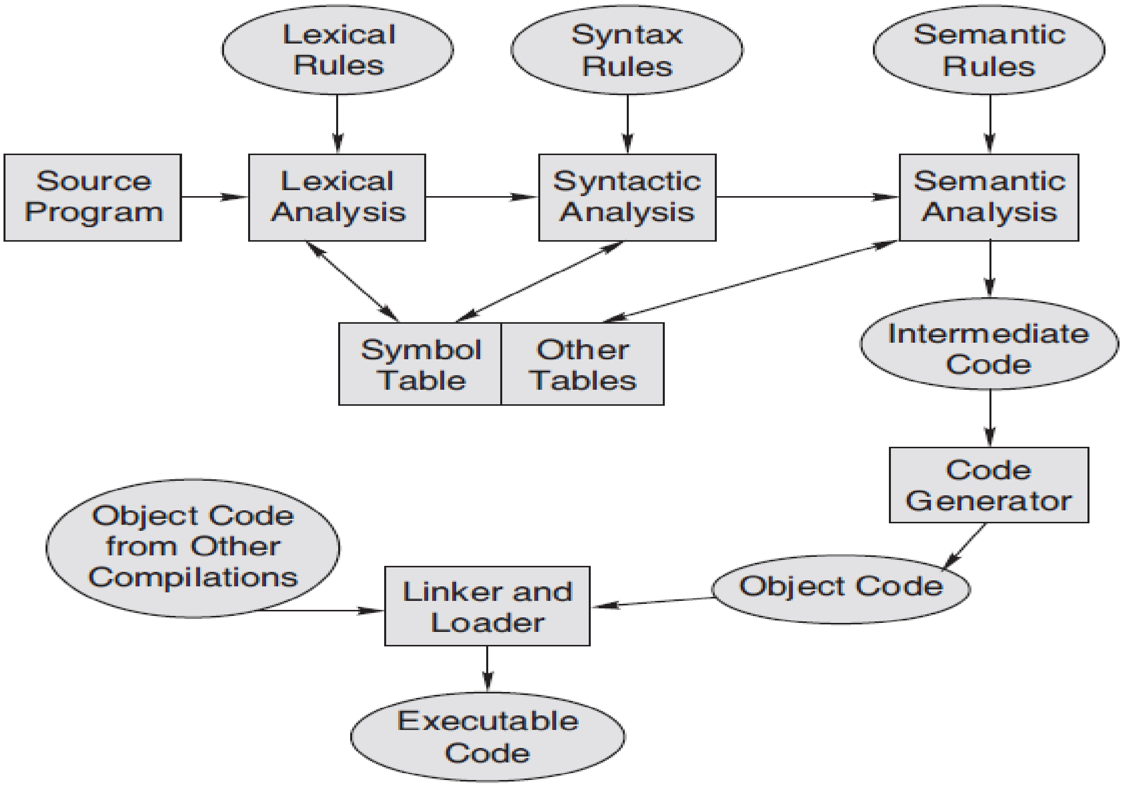


Fig 1.31: The Process of Compilation

The steps in the process of translating a source program in a high-level language to executable code are depicted in Fig. 1.31. The first block is the *lexical analyzer*. It takes successive lines of a program and breaks them into individual lexical items namely, identifier, operator delimiter, etc. and attaches a type tag to each of these. Beside this, it constructs a *symbol table* for each identifier and finds the internal representation of each constant. The symbol table is used later to allocate memory to each variable.

The second stage of translation is called *syntax analysis* or *parsing*. In this phase, expressions, declarations, and other statements are identified by using the results of lexical analysis. Syntax analysis is done by using techniques based on formal grammar of the programming language. In the semantic analysis phase, the syntactic units recognized by the syntax analyzer are processed. An intermediate representation of the final machine language code is produced.

The last phase of translation is code generation, when optimization to reduce the length of machine language program is carried out. The output of the code generator is a machine level language program for the specified computer. If a subprogram library is used or if some subroutines are separately translated and compiled, a final linking and loading step is needed to produce the complete machine language program in an executable form.

If subroutines were compiled separately, then the address allocation of the resulting machine language instructions would not be final. When all routines are connected and placed together in the main memory, suitable memory addresses are allocated. The linker's job is to find the correct main memory locations of the final executable program. The loader then places the executable program in memory at its correct address.

2. Execution:

Therefore, the execution of a program written in high-level language involves the following steps:

1. *Translation* of the program resulting in the object program.
2. *Linking* of the translated program with other object programs needed for execution, thereby resulting in a binary program.
3. *Relocation* of the program to execute from the specific memory area allocated to it.
4. *Loading* of the program in the memory for the purpose of execution.

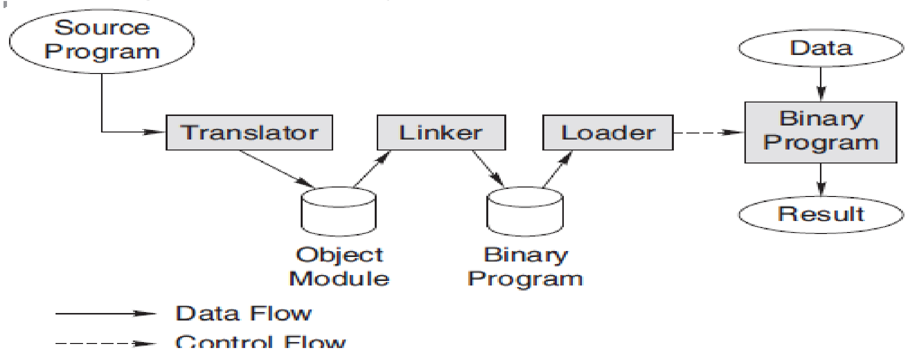


Fig 1.32: A schematic of program execution

Linker

Linking resolves symbolic references between object programs. It makes object programs known to each other. The features of a programming language influence the linking requirements of a program. In FORTRAN/COBOL, all program units are translated separately. Hence, all subprogram calls and common variable references require linking. PASCAL procedures are typically nested inside the main program. Hence, procedure references do not require linking; they can be handled through relocation. References to built-in functions however require linking. In C, files are translated separately. Thus, only function calls that cross file boundaries and references to global data require linking. Linking makes the addresses of programs known to each other so that transfer of control from one subprogram to another or a main program takes place during execution.

Relocation

Relocation means adjustment of all address-dependent locations, such as address constant, to correspond to the allocated space, which means simple modification of the object program so that it can be loaded at an address different from the location originally specified. Relocation is more than simply moving a program from one area to another in the main memory. It refers to the adjustment of address fields. The task of relocation is to add some constant value to each relative address in the memory segment.

Loader

Loading means physically placing the machine instructions and data into main memory, also known as primary storage area. A loader is a system program that accepts object programs and prepares them for execution and initiates the execution (see Fig. 1.32). The functions performed by the loader are:

- Assignment of load-time storage area to the program
- Loading of program into assigned area
- Relocation of program to execute properly from its load time storage area
- Linking of programs with one another

Thus, a loader is a program that places a program's instructions and data into primary storage locations. An *absolute loader* places these items into the precise locations indicated in the machine language program. A *relocating loader* may load a program at various places in primary storage depending on the availability of primary storage area at the time of loading. A program may be relocated dynamica0lly with the help of a *relocating register*.

The base address of the program in primary storage is placed in the relocating register. The contents of the relocation register are added to each address developed by a running program. The user is able to execute the program as if it begins at location zero. At execution time, as the program runs, all address references involve the relocation register. This allows the program to reside in memory locations other than those for which it was translated to occupy.

Linking Loader and Linkage Editor

User programs often contain only a small portion of the instructions and data needed to solve a given problem. Large subroutine libraries are provided so that a programmer wanting to perform certain common operations may use system-supplied routines to do so. Input/output, in particular, is normally handled by routines outside the user program. Hence, the machine language program produced by the translator must normally be combined with other machine language programs residing within the library to form a useful execution unit. This process of program combination is called linking and the software that performs this operation is variously known as a *linking loader* or a *linkage editor*. Linking is done after object code generation, prior to program execution time. At load time, a linking loader combines whatever programs are required and loads them directly into primary storage. A linkage editor also performs the same task, but it creates a load image that it preserves on secondary storage for future reference. Whenever a program is to be executed, the load image produced by the linkage editor may be loaded immediately without the overhead of recombining program segments.

1.12 Program execution

The primary memory of a computer, also called the Random Access Memory, is divided into units known as words. Depending on the computer, a word of memory may be two, four, or even eight bytes in size. Each word is associated with a unique address, which is a positive integer that helps the CPU to access the word. Addresses increase consecutively from the top of the memory to its bottom. When a program is compiled and linked, each instruction and each item of data is assigned an address. At execution time, the CPU finds instructions and data from these addresses. The PC, or program counter, is a CPU register that holds the address of the next instruction to be executed in a program. In the beginning, the PC holds the address of the zeroth instruction of the program. The CPU fetches and then executes the instruction found at this address. The PC is meanwhile incremented to the address of the next instruction in the program. Having executed one instruction, the CPU goes back to look up the PC where it finds the address of the next instruction in the program. This instruction may not necessarily be in the next memory location. It could be at quite a different address. For example, the last statement could have been a go to statement, which unconditionally transfers control to a different point in the program; or there may have been a branch to a function subprogram. The CPU fetches the contents of the words addressed by the PC in the same amount of time, whatever their physical locations. The CPU has random access capability to any and all words of the memory, no matter what their addresses. Program execution proceeds in this way until the CPU has processed the last instruction.

1.13 Fourth generation languages

The Fourth Generation Language is a non-procedural language that allows the user to simply specify what the output should be without describing how data should be processed to produce the result. Fourth generation programming languages are not as clearly defined as are the other earlier generation languages. Most people feel that a fourth generation language, commonly referred to as 4GL, is a high-level language that requires significantly fewer instructions to accomplish a particular task than does a third generation language. Thus, a programmer should be able to write a program faster in 4GL than in a third generation language.

Most third generation languages are procedural languages. That is, the programmer must specify the steps of the procedure the computer has to follow in a program. By contrast, most fourth generation languages are nonprocedural languages. The programmer does not have to give the details of the procedure in the program, but specify, instead, what is wanted. For example, assume that a programmer needs to display some data on the screen, such as the address of a particular employee, say MANAS, from the EMP file. In a procedural language, the programmer would have to write a series of instructions using the following steps:

Step 1: Get a record from the EMP file.

Step 2: If this is the record for MANAS, display the address.

Step 3: If this is not the record for MANAS, go to step 1, until end-of-file.

In a non-procedural language (4GL), however, the programmer would write a single instruction that says:

Get the address of MANAS from EMP file.

Major fourth generation languages are used to get information from files and databases, as in the previous example, and to display or print the information. These fourth generation languages contain a query language, which is used to answer queries or questions with data from a database. The following example shows a query in a common query language, SQL.

SELECT ADDRESS FROM EMP WHERE NAME = 'MANAS'

End user-oriented 4GLs are designed for applications that process low data volumes. These 4GLs run on mainframe computers and may be employed either by information users or by the programmers. This type of 4GL may have its own internal database management software that in turn interacts with the organization's DBMS package. People who are not professional programmers use these products to query databases, develop their own custom-made applications, and generate their own reports with minimum amount of training. For example, ORACLE offers a number of tools suitable for the end user.

Some fourth generation languages are used to produce complex printed reports. These languages contain certain types of programs called generators. With a report generator, the programmer specifies the headings, detailed data, and totals needed in a report. Thus, the report generator produces the required report using data from a file. Other fourth generation languages are used to design screens for data input and output and for menus. These languages contain certain types of programs called screen painters. The programmer designs the screen to look as desired and, therefore, it can be said that the programmer paints the screen using the screen painter program. Fourth generation languages are mostly machine independent. Usually they can be used on more than one type of computer. They are mostly used for office automation or business applications, and not for scientific programs. Some fourth generation languages are designed to be easily learnt and employed by end users.

Advantages of 4GLs

- Programming productivity is increased. One line of a 4GL code is equivalent to several lines of a 3GL code.
- System development is faster.
- Program maintenance is easier.
- End users can often develop their own applications.
- Programs developed in 4GLs are more portable than those developed in other generation languages.
- Documentation is of improved order because most 4GLs are self-documenting.

The differences between third generation languages and fourth generated languages are shown in Table 1.3.

3GL	4GL
Meant for use by professional programmers.	May be used by non professional programmers as well as by professional programmers.
Requires specifications of how to perform a task.	Requires specifications of what task to perform.
All alternatives must be specified.	System determines how to perform the task.
Execution time is less.	Default alternatives are built in. user need not specify these alternatives
Requires large number of procedural instructions code may be difficult to read, understand and maintain by the user.	Requires fewer instructions.
Typically, file oriented.	Difficult to debug.

Table 1.3 3GL vs 4

1.14 Fifth generation languages

Natural languages represent the next step in the development of programming languages belonging to fifth generation languages. Natural language is similar to query language, with one difference: it eliminates the need for the user or programmer to learn a specific vocabulary, grammar, or syntax. The text of a natural language statement resembles human speech closely. 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. Natural language takes the user one step further away from having to deal directly and in detail with computer hardware and software. These languages are also designed to make the computer smarter—that is, to simulate the human learning process. Natural languages already available for microcomputers include CLOUT, Q & A, and SAVY RETRIEVER (for use with databases) and HAL(Human Access Language) for use with LOTUS.

1.15 Classification of Programming languages

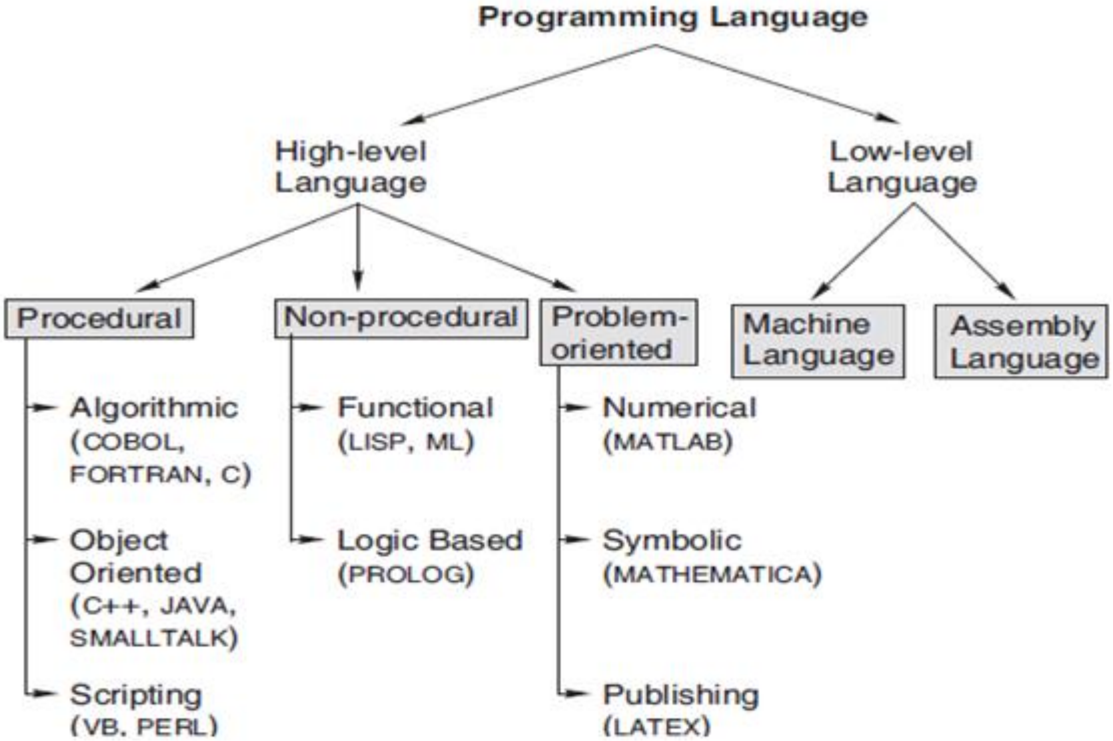


Fig 1.33: Programming language classification

1.15.1 Procedural Languages

Algorithmic languages: These are high-level languages designed for forming convenient expression of procedures, used in the solution of a wide class of problems. In this language, the programmer must specify the steps the computer has to follow while executing a program. Some of languages that fall in the category are C, COBOL, and FORTRAN.

Object-oriented language: The basic philosophy of object oriented programming is to deal with objects rather than functions or subroutines as in strictly algorithmic languages. Objects are self-contained modules that contain data as well as the functions needed to manipulate the data within the same module. In a conventional programming language, data and subroutines or functions are separate. In object oriented programming, subroutines as well as data are locally defined in objects. The difference affects the way a programmer goes about writing a program as well as how information is represented and activated in the computer. The most important object-oriented programming features are

- Abstraction
- encapsulation and data hiding
- polymorphism
- inheritance
- reusable code

C++, JAVA, SMALLTALK, etc. are examples of object oriented languages.

Scripting languages: These languages assume that a collection of useful programs, each performing a task, already exists. It has facilities to combine these components to perform a complex task. A scripting language may thus be thought of as a glue language, which sticks a variety of components together. One of the earliest scripting languages is the UNIX shell. Now there are several scripting languages such as VB script and Perl.

1.15.2 Problem-oriented Languages

These are high-level languages designed for developing a convenient expression of a given class of problems.

1.15.3 Non-procedural Languages

Functional (applicative) languages: These functional languages solve a problem by applying a set of functions to the initial variables in specific ways to get the answer. The functional programming style relies on the idea of function application rather than on the notion of variables and assignments. A program written in a functional language consists of function calls together with arguments to functions. LISP, ML, etc. are examples of functional languages.

Logic-based programming language: A logic program is expressed as a set of atomic sentences, known as fact, and horn clauses, such as if-then rules. A query is then posed. The execution of the program now begins and the system tries to find out if the answer to the query is true or false for the given facts and rules. Such languages include PROLOG.

1.16 Structured programming concept

Structured programming has been called a revolution in programming and is considered as one of the most important advancements in software in the past two decades.

Structured programming is:

- Concerned with improving the programming process through better organization of programs and better programming notation to facilitate correct and clear description of data and control structure.
- Concerned with improved programming languages and organized programming techniques which should be understandable and therefore, more easily modifiable and suitable for documentation.
- More economical to run because good organization and notation make it easier for an optimizing compiler to understand the program logic.
- More correct and therefore more easily debugged, because general correctness theorems dealing with structures can be applied to proving the correctness of programs.

Structured programming can be defined as a

- Top–down analysis for program solving
- Modularization for program structure and organization
- Structured code for individual modules.

1.16.1 Top–Down Analysis

A program is a collection of instructions in a particular language that is prepared to solve a specific problem. For larger programs, developing a solution can be very complicated. From where should it start? Where should it terminate? Top-down analysis is a method of problem solving and problem analysis. The essential idea is to subdivide a large problem into several smaller tasks or parts for ease of analysis. Top-down analysis, therefore, simplifies or reduces the complexity of the process of problem solving. It is not limited by the type of program. Top-down analysis is a general method for attending to any problem. It provides a strategy that has to be followed for solving all problems.

There are two essential ideas in top-down analysis:

- Subdivision of a problem
- hierarchy of tasks

Subdivision of a problem means breaking a big problem into two or more smaller problems. Therefore, to solve the big problem, first these smaller problems have to be solved. Top-down analysis does not simply divide a problem into two or more smaller problems. It goes further than that. Each of these smaller problems is further subdivided. This process continues downwards, creating a hierarchy of tasks, from one level to the next, until no further break up is possible.

The four basic steps to top-down analysis are as follows:

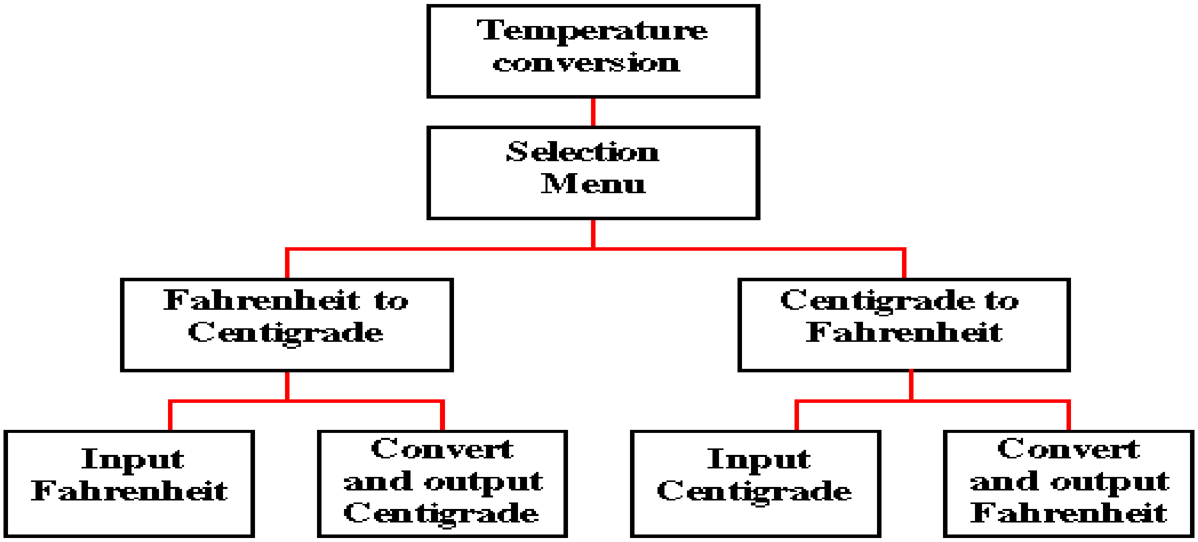
Step 1: Define the complete scope of the problem to determine the basic requirement for its solution. Three factors must be considered in the definition of a programming problem.

- Input: What data is required to be processed by the program?
- Process: What must be done with the input data? What type of processing is required?
- Output: What information should the program produce? In what form should it be presented?

Step 2: Based on the definition of the problem, divide the problem into two or more separate parts.

Step 3: Carefully define the scope of each of these separate tasks and subdivide them further, if necessary, into two or more smaller tasks.

Step 4: Repeat step 3. Every step at the lowest level describes a simple task, which cannot be broken further.



1.16.2 Modular Programming

Modular programming is a program that is divided into logically independent smaller sections, which can be written separately. These sections, being separate and independent units, are called modules.

- A module consists of a series of program instructions or statements in some programming language.
- A module is clearly terminated by some special markers required by the syntax of the language. For example, a BASIC language subroutine is terminated by the return statement.
- A module as a whole has a unique name.
- A module has only one entry point to which control is transferred from the outside and only one exit point from which control is returned to the calling module.

The following are some advantages of modular programming.

- Complex programs may be divided into simpler and more manageable elements.
- Simultaneous coding of different modules by several programmers is possible.
- A library of modules may be created, and these modules may be used in other programs as and when needed.
- The location of program errors may be traced to a particular module; thus, debugging and maintenance may be simplified.

1.16.3 Structured Code

- After the top-down analysis and design of the modular structure, the third and final phase of structured programming involves the use of structured code.
- Structured programming is a method of coding, i.e., writing a program that produces a well-organized module.
- A high-level language supports several control statements, also called structured control statements or structured code, to produce a well-organized structured module.
- Each programming language has different syntax for these statements. In C, for, while, and do...while statements represent repetitive execution. In BASIC, fornext and while-wend are examples of repetitive execution.
- Let us consider the goto statement of BASIC, which is a simple but not a structured control statement. The goto statement can break the normal flow of the program and transfer control to any arbitrary point in a program.

Line No. **Code**

```
10          INPUT X
20          IF X < 0 THEN GOTO 90
30          G = X/2
40          R = X/G
50          G = (R + G)/2
60          IF ABS(R - G) < 0.001 THEN GOTO 40
70          PRINT G
80          GOTO 100
90          PRINT INVALID INPUT”
100         END
```

The structured version of this program using while wend statement is given below.

```
INPUT X
IF X > 0 THEN
        G = X/2
        R = X/G
        WHILE ABS (R – G) < 0.001
                R = X/G
                G = (R + G)/2
        WEND
        PRINT G
ELSE
        PRINT “INVALID INPUT”
END
```

Now if there is no normal break of control flow, gotos are inevitable in unstructured languages but they can be and should be always avoided while using structured programs except in unavoidable situations.

1.16.4 The Process of Programming

The job of a programmer is not just writing program instructions. The programmer does several other additional jobs to create a working program. There are some logical and sequential job steps which the programmer has to follow to make the program operational.

These are as follows:

1. Understand the problem to be solved
2. Think and design the solution logic
3. Write the program in the chosen programming language

4. Translate the program to machine code
5. Test the program with sample data
6. Put the program into operation

The first job of the programmer is to understand the problem. To do that the requirements of the problem should be clearly defined. And for this, the programmer may have to interact with the user to know the needs of the user. Thus this phase of the job determines the 'what to' of the task.

The next job is to develop the logic of solving the problem. Different solution logics are designed and the orders in which these are to be used in the program are defined. Hence, this phase of the job specifies the 'how to' of the task. Once the logics are developed, the third phase of the job is to write the program using a chosen programming language. The rules of the programming language have to be observed while writing the program instructions. The computer recognizes and works with 1's and 0's.

Hence program instructions have to be converted to 1's and 0's for the computer to execute it. Thus, after the program is written, it is translated to the machine code, which is in 1's and 0's with the help of a translating program. Now, the program is tested with dummy data. Errors in the programming logic are detected during this phase and are removed by making necessary changes in either the logic or the program instructions. The last phase is to make the program operational. This means, the program is put to actual use. Errors occurring in this phase are rectified to finally make the program work to the user's satisfaction.

1.71 Algorithms

1.17.1 What is an Algorithm?

Computer scientist Niklaus Wirth stated that

Program = Algorithms + Data

An algorithm is a part of the plan for the computer program. In fact, an algorithm is 'an effective procedure for solving a problem in a finite number of steps'. It is effective, which means that an answer is found and it has a finite number of steps. A well-designed algorithm will always provide an answer; it may not be the desired answer but there will be an answer. It may be that the answer is that there is no answer. A well- designed algorithm is also guaranteed to terminate.

1.17.2 Different Ways of Stating Algorithms

Algorithms may be represented in various ways. There are four ways of stating algorithms.
These are as follows:

- Step-form
- Pseudo-code
- Flowchart
- Nassi-Schneiderman

In the step form representation, the procedure of solving a problem is stated with written statements. Each statement solves a part of the problem and these together complete the solution. The step-form uses just normal language to define each procedure. Every statement, that defines an action, is logically related to the preceding statement. This algorithm has been discussed in the following section with the help of an example.

The pseudo-code is a written form representation of the algorithm. However it differs from the step form as it uses a restricted vocabulary to define its action of solving the problem. One problem with human language is that it can seem to be imprecise. But the pseudo-code, which is in human language, tends toward more precision by using a limited vocabulary.

Flowchart and Nassi-Schneiderman are graphically oriented representation forms. They use symbols and language to represent sequence, decision, and repetition actions. Only the flowchart method of representing the problem solution has been explained with several examples.

1.17.3 Key Features of an Algorithm and the Step-form

Here is an example of an algorithm, for making a pot of tea.

1. If the kettle does not contain water, then fill the kettle.
2. Plug the kettle into the power point and switch it on.
3. If the teapot is not empty, then empty the teapot.
4. Place tea leaves in the teapot.
5. If the water in the kettle is not boiling, then go to step 5.
6. Switch off the kettle.
7. Pour water from the kettle into the teapot.

It can be seen that the algorithm has a number of steps and that some steps (steps 1, 3, and 5) involve decision making and one step (step 5 in this case) involves repetition, in this case the process of waiting for the kettle to boil.

From this example, it is evident that algorithms show these three features:

- Sequence (also known as process)
- Decision (also known as selection)
- Repetition (also known as iteration or looping)

Therefore, an algorithm can be stated using three basic constructs: sequence, decision, and repetition.

Sequence

Sequence means that each step or process in the algorithm is executed in the specified order. In the above example, each process must be in the proper place otherwise the algorithm will fail.

The decision constructs— if ... then, if ... then ... else ...

In algorithms the outcome of a decision is either true or false; there is no state in between.

The outcome of the decision is based on some condition that can only result in a true or false value. For example,

```
if today is Friday then collect pay
```

is a decision and the decision takes the general form:

```
if proposition then process
```

A proposition, in this sense, is a statement, which can only be true or false. It is either true that ‘today is Friday’ or it is false that ‘today is not Friday’. It cannot be both true and false. If the proposition is true, then the process or procedure that follows the then is executed. The decision
can also be stated as:

```
if proposition
     then process1
else process2
```

This is the if … then … else … form of the decision.

This means that if the proposition is true then execute process1, else, or otherwise, execute process2. The first form of the decision if proposition then process has a null else, that is, there is no else.

The repetition constructs— repeat and while

Repetition can be implemented using constructs like the repeat loop, while loop, and if.. then .. goto .. loop. The Repeat loop is used to iterate or repeat a process or sequence of processes until some condition becomes true. It has the general form:

```
Repeat
     Process1
     Process2
     ..............
     ..............
     ProcessN
     Until proposition
```

Here is an example.

```
Repeat
     Fill water in kettle
Until kettle is full
```

The process is ‘Fill water in kettle,’ the proposition is ‘kettle is full’.

The Repeat loop does some processing before testing the state of the proposition.

What happens though if in the above example the kettle is already full? If the kettle is already full at the start of the Repeat loop, then filling more water will lead to an overflow.

This is a drawback of the Repeat construct.

In such a case the while loop is more appropriate. The above example with the while loop is shown as follows:

```
while kettle is not full
     fill water in kettle
```

Since the decision about the kettle being full or not is made before filling water, the possibility of an overflow is eliminated.

The while loop finds out whether some condition is true before repeating a process or a sequence of processes.

If the condition is false, the process or the sequence of processes is not executed. The general form of while loop is:

```
while proposition
begin
     Process 1
     Process 2
     ........
     ......
     Process N
end
```

The if .. then goto .. is also used to repeat a process or a sequence of processes until the given proposition is false. In the kettle example, this construct would be implemented as follows:

1. Fill some water in kettle
2. if kettle not full then goto 1

So long as the proposition ‘kettle not full’ is true the process, ‘fi ll some water in kettle’ is repeated.

The general form of if .. then goto .. is:

```
Process1
Process2
……….
……….
ProcessN
if proposition then goto Process1
```

Termination

The definition of algorithm cannot be restricted to procedures that eventually finish. Algorithms might also include procedures that could run forever without stopping. Such a procedure has been called a computational method by Knuth or calculation procedure or algorithm by Kleene. However, Kleene notes that such a method must eventually exhibit ‘some object.’ Minsky (1967) makes the observation that, if an algorithm has not terminated, then how can the following question be answered: “Will it terminate with the correct answer?” Thus the answer is: undecidable. It can never be known, nor can the designer do an analysis beforehand to find it out. The analysis of algorithms for their likelihood of termination is called termination analysis.

Correctness

The prepared algorithm needs to be verified for its correctness. Correctness means how easily its logic can be argued to meet the algorithm's primary goal. This requires the algorithm to be made in such a way that all the elements in it are traceable to the requirements. Correctness requires that all the components like the data structures, modules, external interfaces, and module interconnections are completely specified. In other words, correctness is the degree to which an algorithm performs its specified function. The most common measure of correctness is defects per Kilo Lines of Code (KLOC) that implements the algorithm, where defect is defined as the verified lack of conformance to requirements.

1.17.4 What are Variables?

Therefore, it is now necessary to understand the concept of data. It is known that data is a symbolic representation of value and that programs set the context that gives data a proper meaning. In programs, data is transformed into information. The question is, how is data represented in programs? Almost every algorithm contains data and usually the data is 'contained' in what is called a variable. The variable is a container for a value that may vary during the execution of the program. For example, in the tea-making algorithm, the level of water in the kettle is a variable; the temperature of the water is a variable, and the quantity of tea leaves is also a variable.

Each variable in a program is given a name, for example,

- Water_Level
- Water_Temperature
- Tea_Leaves_Quantity

and at any given time the value, which is represented by Water_Level, for instance, may be different to its value at some other time. The statement

if the kettle does not contain water then fill the kettle could also be written as

    if Water_Level is 0 then fill the kettle
    or
    if Water_Level = 0 then fill the kettle

At some point Water_Level will be the maximum value, whatever that is, and the kettle will be full.

Variables and data types

The data used in algorithms can be of different types. The simplest types of data that an algorithm might use are

- Numeric data, e.g., 12, 11.45, 901, etc.
- Alphabetic or character data such as 'A', 'Z', or 'This is alphabetic'
- Logical data, that is, propositions with true/false values

Naming of variables

One should always try to choose meaningful names for variables in algorithms to improve the readability of the algorithm or program. This is particularly important in large and complex programs.

In the tea-making algorithm, plain English was used. It has been shown how variable names may be used for some of the algorithm variables. In Table 1.3, the right-hand column contains variable names which are shorter than the original and do not hide the meaning of the original phrase. Underscores have been given to indicate that the words belong together and represent a variable.

Algorithm in Plain English	Algorithm using Variable Names
1. If the kettle does not contain water, then fill the kettle.	1. If kettle_empty then fill the kettle.
2. Plug the kettle into the power point and	2. Plug the kettle into the power point and

switch it on.	switch it on.
3. If the teapot is not empty, then empty the teapot.	3. If teapot_not_empty then empty the teapot.
4. Place tea leaves in the teapot.	4. Place tea leaves in the teapot.
5. If the water in the kettle is not boiling then go to step 5.	5. If water_not_boiling then goto step 5.
6. Switch off the kettle.	6. Switch off the kettle.
7. Pour water from the kettle into the teapot.	7. Pour water from the kettle into the teapot.

Table 1.3 Algorithm using variable names.

There are no hard and fast rules about how variables should be named but there are many conventions. It is a good idea to adopt a conventional way of naming variables. The algorithms and programs can benefit from using naming conventions for processes too.

1.17.5 Subroutines

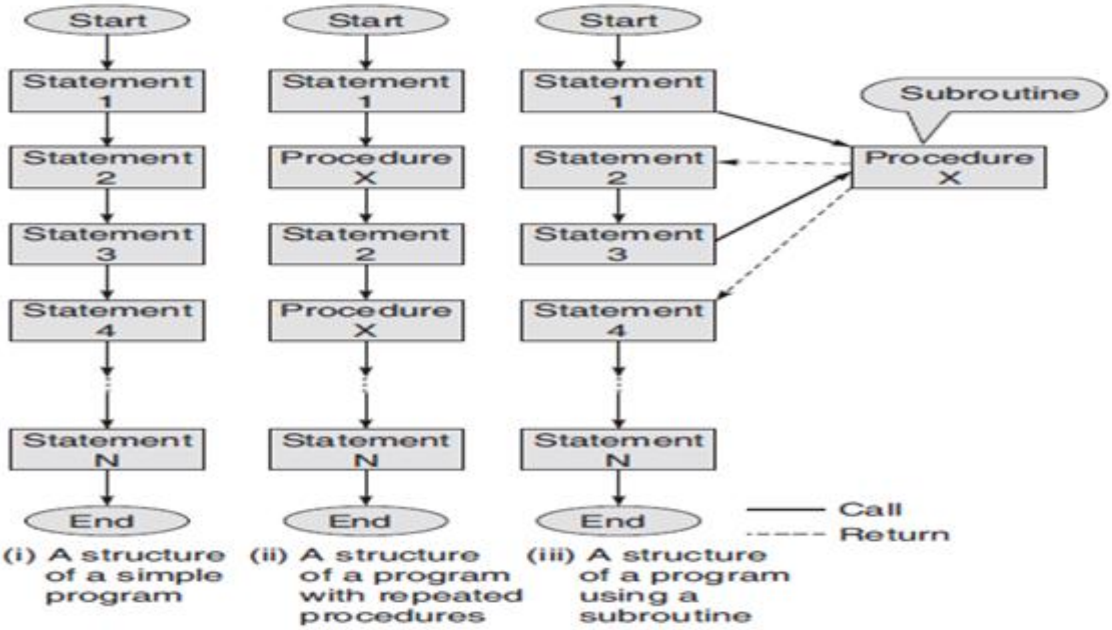

Start
Statement 1
Statement 2
Statement 3
Statement 4
Statement N
End
Procedure X
Subroutine
Call
Return
(i) A structure of a simple program
(ii) A structure of a program with repeated procedures
(iii) A structure of a program using a subroutine

Fig 1.34: Program Structure

A simple program is a combination of statements that are implemented in a sequential order. A statement block is a group of statements. Such a program is shown in Fig. 1.34(i). There might be a specific c block of statement, which is also known as a procedure that is run several times at different points in the implementation sequence of the larger program. This is shown in Fig.1.34 (ii). Here, this specific block of statement is named “procedure X”. In this example program, the “procedure X” is written twice in this example. This enhances the size of the program. Since this particular procedure is required to be run at two specific points in the implementation sequence of the larger program, it may be treated as a separate entity and not included in the main program. In fact, this procedure may be called whenever required as shown in Fig.1.34 (iii). Such a procedure is known as a subroutine.

Therefore, a subroutine, also known as procedure, method or function, is a portion of instruction that is invoked from within a larger program to perform a specific task. At the same time the subroutine is relatively independent of the remaining statements of the larger program. The subroutine behaves in much the same way as a program that is used as one step in a larger program. A subroutine is often written so that it can be started ("called") several times and/or from several places during a single execution of the program, including from other subroutines, and then branch back (*return*) to the next instruction after the "call", once the subroutine's task is done. Thus, such subroutines are invoked with a CALL statement with or without passing of parameters from the calling program. The subroutine works on the parameters if given to it, otherwise it works out the results and gives out the result by itself and returns to the calling program or pass the results to the calling program before returning to it.

The technique of writing subroutine has some distinct advantages. The subroutine reduces duplication of block of statements within a program, enables reuse of the block of statements that forms the subroutine across multiple programs, decomposes a complex task into simpler steps, divides a large programming task among various programmers or various stages of a project and hides implementation details from users.

However, there are some disadvantages in using subroutines. The starting or invocation of a subroutine requires some computational overhead in the call mechanism itself. The subroutine requires some well defined housekeeping techniques at its entry and exit from it.

Some examples on developing algorithms using step-form

For illustrating the step-form the following conventions are assumed:

1. Each algorithm will be logically enclosed by two statements START and STOP.
2. To accept data from user, the INPUT or READ statements are to be used.
3. To display any user message or the content in a variable, PRINT statement will be used. Note that the message will be enclosed within quotes.
4. There are several steps in an algorithm. Each step results in an action. The steps are to be acted upon sequentially in the order they are arranged or directed.
5. The arithmetic operators that will be used in the expressions are
   - **i.** '←' ….Assignment (the left-hand side of '←' should always be a single variable)
     Example: The expression x ← 6 means that a value 6 is assigned to the variable x. In terms of memory storage, it means a value of 6 is stored at a location in memory which is allocated to the variable x.
   - **ii.** '+'….. Addition
     Example: The expression z ← x + y means the value contained in variable x and the value contained in variable y is added and the resulting value obtained is assigned to the variable z.
   - **iii.** '–'….. Subtraction
     Example: The expression z ← x – y means the value contained in variable y is subtracted from the value contained in variable x and the resulting value obtained is assigned to the variable z
   - **iv.** '*'….. Multiplication
     Example: Consider the following expressions written in sequence:
     ```
     x ← 5
     y ← 6
     z ← x * y
     ```
     The result of the multiplication between x and y is 30. This value is therefore assigned to z.
   - **v.** '/'….. Division
     Example: The following expressions written in sequence illustrates the meaning of the division operator:

```
x ← 10
y ← 6
z ← x/y
```

The quotient of the division between x and y is 1 and the remainder is 4. When such an operator is used the quotient is taken as the result whereas the remainder is rejected. So here the result obtained from the expression x/y is 1 and this is assigned to z.

6. In propositions, the commonly used relational operators will include

i. '>' ….. Greater than

Example: The expression x > y means if the value contained in x is larger than that in y then the outcome of the expression is true, which will be taken as 1. Otherwise, if the outcome is false then it would be taken as 0.

ii. '<=' …..Less than or equal to

Example: The expression x <= y implies that if the value held in x is either less than or equal to the value held in y then the outcome of the expression is true and so it will be taken as 1. But if the outcome of the relational expression is false then it is taken as 0.

iii. '<' …… Less than

Example: Here the expression x < y implies that if the value held in x is less than that held in y then the relational expression is true, which is taken as 1, otherwise the expression is false and hence will be taken as 0.

iv. '=' …… Equality

Example: The expression x = y means that if the value in x and that in y are same then this relational expression is true and hence the outcome is 1 otherwise the outcome is false or 0.

v. '>=' …… Greater than or equal to

Example: The expression x >= y implies that if the value in x is larger or equal to that in y then the outcome of the expression is true or 1, otherwise it is false or 0.

vi. '!=' …… Non- equality

Example: The expression x != y means that if the value contained in x is not equal to the value contained in y then the outcome of the expression is true or 1, otherwise it is false or 0.

Note: The 'equal to (=)' operator is used both for assignment as well as equality specification. When used in proposition, it specifies equality otherwise assignment. To differentiate 'assignment' from 'equality' left arrow (←) may be used. For example, a ←b is an assignment but a = b is a proposition for checking the equality.

7. The most commonly used logical operators will be AND, OR and NOT. These operators are used to specify multiple test conditions forming composite proposition. These are

i. 'AND'…… Conjunction

The outcome of an expression is true or 1 when both the propositions AND-ed is true otherwise it is false or 0.

Example: Consider the expressions

```
x ← 2
y ← 1
x = 2 AND y = 0
```

In the above expression the proposition 'x = 2' is true because the value in x is 2. Similarly, the proposition 'y = 0' is untrue as y holds 1 and therefore this proposition is false or 0. Thus, the above expression may be represented as 'true' AND 'false' the outcome for which is false or 0.

ii. ‘OR’ …… Disjunction

The outcome of an expression is true or 1 when anyone of the propositions OR-ed is true otherwise it is false or 0.

Example: Consider the expressions

x ← 2
y ← 1
x = 2 OR y = 0

Here, the proposition ‘x = 2’ is true since x holds 2 while the proposition ‘y = 0’ is untrue or false. Hence the third expression may be represented as ‘true’ OR ‘false’ the outcome for which is true or 1.

iii. ‘NOT’ …… Negation

If outcome of a proposition is ‘true’, it becomes ‘false’ when negated or NOT-ed.

Example: Consider the expression

x ← 2
NOT x = 2

The proposition ‘x = 2’ is ‘true’ as x contains the value 2. But the second expression negates this by the logical operator NOT which gives an outcome ‘false’.

Ex 4. Write the algorithm for finding the sum of any two numbers.

Solution Let the two numbers be *A* and *B* and let their sum be equal to *C*. Then, the desired algorithm is given as follows:

```
1. START
2. PRINT "ENTER TWO NUMBERS"
3. INPUT A, B
4. C ← A + B          //Add values assigned to A and B and
assign this value to C
5. PRINT C
6. STOP
```

Explanation The first step is the starting point of the algorithm.

The next step requests the programmer to enter the two numbers that have to be added. Step 3 takes in the two numbers given by the programmer and keeps them in variables A and B. The fourth step adds the two numbers and assigns the resulting value to the variable C. The fifth step prints the result stored in C on the output device. The sixth step terminates the procedure.

Ex 5. Write the algorithm for determining the remainder of a division operation where the dividend and divisor are both integers.

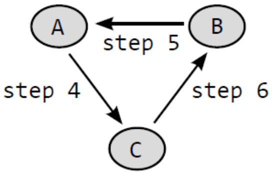
Solution Let *N* and *D* be the dividend and divisor, respectively. Assume *Q* to be the quotient, which is an integer, and *R* to be the remainder. The algorithm for the given problem is as follows.

```
1. START
2. PRINT "ENTER DIVIDEND"
3. INPUT N
4. PRINT "ENTER DIVISOR"
5. INPUT D      //Only integer value is obtained and remainder
ignored
6. Q ← N/D (Integer division)
7. R ← N - Q * D
8. PRINT R
9. STOP
```

Explanation The first step indicates the starting point of the algorithm. The next step asks the programmer to enter the dividend value. The third step keeps the dividend value in the variable N. Step 4 asks for the divisor value to be entered. This is kept in the variable D. In step 6, the value in N is divided by that in D. Since both the numbers are integers, the result is an integer. This value is assigned to Q. Any remainder in this step is ignored. In step 7, the remainder is computed by subtracting the product of the integer quotient and the integer divisor from integer dividend N. The computed value of the remainder is an integer here and obviously less than the divisor. The remainder value is assigned to the variable R. This value is printed on an output device in step 8. Step 9 terminates the algorithm.

Ex 6. Construct the algorithm for interchanging the numeric values of two variables.
Solution Let the two variables be *A* and *B*. Consider *C* to be a third variable that is used to store the value of one of the variables during the process of interchanging the values.
The algorithm for the given problem is as follows.

```
1. START
2. PRINT "ENTER THE VALUE OF A & B"
3. INPUT A, B
4. C ← A
5. A ← B
```



```
6. B ← C
7. PRINT A, B
8. END
```

Ex 7. Write an algorithm that compares two numbers and prints either the message identifying the greater number or the message stating that both numbers are equal.
Solution This example demonstrates how the process of selection or decision making is implemented in an algorithm using the step form. Here, two variables, A and B, are assumed to represent the two numbers that are being compared. The algorithm for this problem is given as follows.

```
1. START
2. PRINT "ENTER TWO NUMBERS"
3. INPUT A, B
4. IF A > B THEN
       PRINT "A IS GREATER THAN B"
5. IF B > A THEN
       PRINT "B IS GREATER THAN A"
6. IF A = B THEN
       PRINT "BOTH ARE EQUAL"
7. STOP
```

Ex 8. Write an algorithm to check whether a number given by the user is odd or even.
Solution Let the number to be checked be represented by *N*. The number *N* is divided by 2 to give an integer quotient, denoted by *Q*. If the remainder, designated as *R*, is zero, *N* is even; otherwise *N* is odd. This logic has been applied in the following algorithm.

```
1. START
2. PRINT “ENTER THE NUMBER”
3. INPUT N
4. Q ← N/2 (Integer division)
5. R ← N – Q * 2
6. IF R = 0 THEN
     PRINT “N IS EVEN”
7. IF R != 0 THEN
      PRINT “N IS ODD”
8. STOP
```

Ex 9. Print the largest number among three numbers.
Solution Let the three numbers be represented by *A*, *B*, and *C*. There can be three ways of solving the problem. The three algorithms, with some differences, are given below.

```
1. START
2. PRINT “ENTER THREE NUMBERS”
3. INPUT A, B, C
4. IF A >= B AND B >= C
      THEN PRINT A
5. IF B >= C AND C >= A
      THEN PRINT B
ELSE
      PRINT C
6. STOP
```

Or
Here, the algorithm uses a **nested if** construct.

```
1. START
2. PRINT “ENTER THREE NUMBERS”
3. INPUT A, B, C
4. IF A > B THEN
      IF A > C THEN
           PRINT A
      ELSE
           PRINT C
   ELSE IF B > C THEN
       PRINT B
   ELSE
      PRINT C
5. STOP
```

Ex 10. Take three sides of a triangle as input and check whether the triangle can be drawn or not. If possible, classify the triangle as equilateral, isosceles, or scalene.
Solution Let the length of three sides of the triangle be represented by *A*, *B*, and *C*. Two alternative algorithms for solving the problem are given, with explanations after each step, as follows:

```
1. START
   Step 1 starts the procedure.
2. PRINT “ENTER LENGTH OF THREE SIDES OF A TRIANGLE”
   Step 2 outputs a message asking for the entry of the
   lengths for each side of the triangle.
3. INPUT A, B, C
   Step 3 reads the values for the lengths that has been
   entered and assigns them to A, B and C.
4. IF A + B > C AND B + C > A AND A + C > B THEN
      PRINT "TRIANGLE CAN BE DRAWN"
   ELSE
      PRINT "TRIANGLE CANNOT BE DRAWN": GO TO 6
```

It is well known that in a triangle, the summation of lengths of any two sides is always greater than the length of the third side. This is checked in step 4. So for a triangle all the propositions "A + B > C ", " B + C > A " and " A + C > B " must be true. In such a case, with the lengths of the three sides, that has been entered, a triangle can be formed. Thus, the message "TRIANGLE CAN BE DRAWN" is printed and the next step 5 is executed. But if any one of the above three propositions is not true then the message "TRIANGLE CANNOT BE DRAWN" is printed and so no classification is required. Thus in such a case the algorithm is terminated in step 6.

```
5. IF A = B AND B = C THEN
      PRINT "EQUILATERAL"
   ELSE
      IF A != B AND B != C AND C !=A THEN
           PRINT "SCALENE"
      ELSE
           PRINT "ISOSCELES"
```

After it has been found in step 4 that a triangle can be drawn, this step is executed. To find whether the triangle is an "EQUILATERAL" triangle the propositions "A = B" and "B = C" are checked. If both of these are true, then the message "EQUILATERAL" is printed which means that the triangle is an equilateral triangle. On the other hand if any or both the propositions "A = B" and "B = C" are found to be untrue then the propositions "A != B" and "B != C" and "C !=A" are checked. If none of the sides are equal to each other then all these propositions are found to be true and so the message "SCALENE" will be printed. But if these propositions "A != B" and "B != C" and "C !=A" are false then the triangle is obviously an isosceles triangle and hence the message "ISOSCELES" is printed.

```
6. STOP
```

The procedure terminates here.

Or

This algorithm differs from the previous one and applies an alternate way to test whether a triangle can be drawn with the given sides and also identify its type.

```
1. START
2. PRINT "ENTER THE LENGTH OF 3 SIDES OF A TRIANGLE"
3. INPUT A, B, C
4. IF A + B > C AND B + C > A AND C + A > B THEN
      PRINT "TRIANGLE CAN BE DRAWN"
   ELSE
      PRINT "TRIANGLE CANNOT BE DRAWN"
           : GO TO 8
5. IF A = B AND B = C THEN
      PRINT "EQUILATERAL TRIANGLE"
           : GO TO 8
6. IF A = B OR B = C OR C = A THEN
      PRINT "ISOSCELES TRIANGLE"
           : GO TO 8
```

```
7. PRINT "SCALENE TRIANGLE"
8. STOP
```

Having followed the explanations given with each of the earlier examples, the reader has already understood how the stepwise representation of any algorithm of any problem starts, constructs the logic statements and terminates. In a similar way the following example exhibits the stepwise representation of algorithms for various problems.

Ex 11. In an academic institution, grades have to be printed for students who appeared in the final exam. The criteria for allocating the grades against the percentage of total marks obtained are as follows.

Marks	***Grade***	***Marks***	***Grade***
91–100	O	61–70	B
81–90	E	51–60	C
71–80	A	<= 50	F

The percentage of total marks obtained by each student in the final exam is to be given as input to get a printout of the grade the student is awarded.

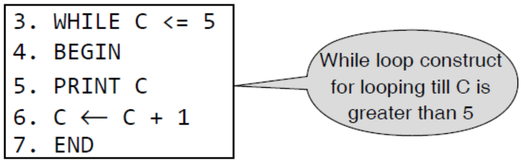
Solution The percentage of marks obtained by a student is represented by *N*. The algorithm for the given problem is as follows.

```
1. START
2. PRINT "ENTER THE OBTAINED PERCENTAGE MARKS"
3. INPUT N
4. IF N > 0 AND N <= 50 THEN
       PRINT "F"
5. IF N > 50 AND N <= 60 THEN
       PRINT "C"
6. IF N > 60 AND N <= 70 THEN
       PRINT "B"
7. IF N > 70 AND N <= 80 THEN
       PRINT "A"
8. IF N > 80 AND N <= 90 THEN
       PRINT "E"
9. IF N > 90 AND N <= 100 THEN
       PRINT "O"
10. STOP
```

Ex 12. Construct an algorithm for incrementing the value of a variable that starts with an initial value of 1 and stops when the value becomes 5.

Solution This problem illustrates the use of iteration or loop construct. Let the variable be represented by *C*. The algorithm for the said problem is given as follows.

```
1. START
2. C ← 1
```



```
3. WHILE C <= 5
4. BEGIN
5. PRINT C
6. C ← C + 1
7. END
```

While loop construct for looping till C is greater than 5

```
8. STOP
```

Ex 13. Write an algorithm for the addition of *N* given numbers.
Solution Let the sum of *N* given numbers be represented by *S*. Each time a number is given as input, it is assigned to the variable *A*. The algorithm using the loop construct ‘if … then goto …’ is used as follows:

```
1. START
2. PRINT “HOW MANY NUMBERS?”
3. INPUT N
4. S ← 0
5. C ← 1
6. PRINT “ENTER NUMBER”
7. INPUT A
8. S ← S + A
9. C ← C + 1
10. IF C <= N THEN GOTO 6
11. PRINT S
12. STOP
```

Ex 14. Develop the algorithm for finding the sum of the series 1 + 2 + 3 + 4 + … up to *N* terms.
Solution Let the sum of the series be represented by *S* and the number of terms by *N*. The algorithm for computing the sum is given as follows.

```
1. START
2. PRINT “HOW MANY TERMS?”
3. INPUT N
4. S ← 0
5. C ← 1
6. S ← S + C
7. C ← C + 1
8. IF C <= N THEN GOTO 6
9. PRINT S
10. STOP
```

Ex 15. Write an algorithm for determining the sum of the series 2 + 4 + 8 + … up to *N*.
Solution Let the sum of the series be represented by *S* and the number of terms in the series by *N*. The algorithm for this problem is given as follows.

```
1. START
2. PRINT “ENTER THE VALUE OF N”
3. INPUT N
4. S ← 0
5. C ← 2
6. S ← S + C
7. C ← C * 2
8. IF C <= N THEN GOTO STEP 6
9. PRINT S
10. STOP
```

Ex 16. Write an algorithm to find out whether a given number is a prime number or not.
Solution The algorithm for checking whether a given number is a prime number or not is as follows.

```
1. START
2. PRINT “ENTER THE NUMBER”
3. INPUT N
4. IF N = 2 THEN
```

```
         PRINT "CO-PRIME" GOTO STEP 12
    5. D ← 2
    6. Q ← N/D (Integer division)
    7. R ← N – Q*D
    8. IF R = 0 THEN GOTO STEP 11
    9. D ← D + 1
    10. IF D <= N/2 THEN GOTO STEP 6
    11. IF R = 0 THEN
         PRINT "NOT PRIME"
       ELSE
         PRINT "PRIME"
    12. STOP
```

Ex 17. Write an algorithm for calculating the factorial of a given number *N*.

Solution The algorithm for finding the factorial of number *N* is as follows.

```
    1. START
    2. PRINT "ENTER THE NUMBER"
    3. INPUT N
    4. F ← 1
    5. C ← 1
    6. WHILE C <= N
    7. BEGIN
    8. F ← F * C
    9. C ← C + 1
    10. END
    11. PRINT F
    12. STOP
```

Ex 18. Write an algorithm to print the Fibonacci series up to *N* terms.

Solution The Fibonacci series consisting of the following terms 1, 1, 2, 3, 5, 8, 13, … is generated using the following algorithm.

```
    1. START
    2. PRINT "ENTER THE NUMBER OF TERMS"
    3. INPUT N
    4. C ← 1
    5. T ← 1
    6. T1 ← 0
    7. T2 ← 1
    8. PRINT T
    9. T ← T1 + T2
    10. C ← C + 1
    11. T1 ← T2
    12. T2 ← T
    13. IF C <= N THEN GOTO 8
    14. STOP
```

Ex 19. Write an algorithm to find the sum of the series 1 + *x* + *x*2 + *x*3 + *x*4 + … up to *N* terms.

Solution

```
    1. START
    2. PRINT "HOW MANY TERMS"
    3. INPUT N.
```

```
4. PRINT "ENTER VALUE OF X"
5. INPUT X
6. T ← 1
7. C ← 1
8. S ← 0
9. S ← S + T
10. C ← C + 1
11. T ← T * X
12. IF C <= N THEN GOTO 9
13. PRINT S
14. STOP
```

Ex 20. Write the algorithm for computing the sum of digits in a number.

Solution

```
1. START
2. PRINT "ENTER THE NUMBER"
3. INPUT N
4. S ← 0
5. Q ← N/10 (Integer division)
6. R ← N – Q * 10
7. S ← S + R
8. N ← Q
9. IF N > 0 THEN GOTO 5
10. PRINT S
11. STOP
```

Ex 21. Write an algorithm to find the largest number among a list of numbers.

Solution The largest number can be found using the following algorithm.

```
1. START
2. PRINT "ENTER, TOTAL COUNT OF NUMBERS IN LIST"
3. INPUT N
4. C ← 0
5. PRINT "ENTER THE NUMBER"
6. INPUT A
7. C ← C + 1
8. MAX ← A
9. PRINT "ENTER THE NUMBER"
10. INPUT B
11. C ← C + 1
12. IF B > MAX THEN
          MAX ← B
13. IF C <= N THEN GOTO STEP 9
14. PRINT MAX
15. STOP
```

Ex 22. Write an algorithm to check whether a given number is an Armstrong number or not. An Armstrong number is one in which the sum of the cube of each of the digits equals that number.

Solution If a number 153 is considered, the required sum is (13 + 53 + 33), i.e., 153. This shows that the number is an Armstrong number. The algorithm to check whether 153 is an Armstrong number or not, is given as follows.

```
1. START
```

```
2. PRINT "ENTER THE NUMBER"
3. INPUT N
4. M ← N
5. S ← 0
6. Q ← N/10 (Integer division)
7. R ← N – Q * 10
8. S ← S + R * R * R
9. N ← Q
10. IF N > 0 THEN GOTO STEP 6
11. IF S = M THEN
      PRINT "THE NUMBER IS ARMSTRONG"
     ELSE PRINT "THE NUMBER IS NOT ARMSTRONG"
12. STOP
```

Ex 23. Write an algorithm for computing the sum of the series 1 + *x* +*x*2/2! + *x*3/3! + *x*4/4! + … up to *N* terms.

Solution

```
1. START
2. PRINT "ENTER NUMBER OF TERMS"
3. INPUT N
4. PRINT "ENTER A NUMBER"
5. INPUT X
6. T ← 1
7. S ← 0
8. C ← 1
9. S ← S + T
10. T ← T * X/C
11. C ← C + 1
12. IF C <= N THEN GO TO STEP 9
13. PRINT S
14. STOP
```

1.18 Pseudo-code

Like step-form, Pseudo-code is a written statement of an algorithm using a restricted and well-defined vocabulary. It is similar to a 3GL, and for many programmers and program designers it is the preferred way to state algorithms and program specifications. Although there is no standard for pseudo-code, it is generally quite easy to read and use. For instance, a sample pseudo-code is written as follows:

```
dowhile kettle_empty
      Add_Water_To_Kettle
end dowhile
```

As can be seen, it is a precise statement of a while loop.

1.19 Flowcharts

- A flowchart provides appropriate steps to be followed in order to arrive at the solution to a problem.
- A flowchart comprises a set of various standard shaped boxes that are interconnected by flow lines. Flow lines have arrows to indicate the direction of the flow of control between the boxes. The activity to be performed is written within the boxes in English.

- These flowcharts play a vital role in the programming of a problem and are quite helpful in understanding the logic of complicated and lengthy problems.
- Once the flowchart is drawn, it becomes easy to write the program in any high-level language. Often flowcharts are helpful in explaining the program to others. Hence, a flowchart is a must for better documentation of a complex program.

Standards for flowcharts The following standards should be adhered to while drawing flow charts.

- Flowcharts must be drawn on white, unlined 81/2 × 11 paper, on one side only.
- Flowcharts start on the top of the page and flow down and to the right.
- Only standard flowcharting symbols should be used.
- A template to draw the final version of flowchart should be used.
- The contents of each symbol should be printed legibly.
- English should be used in flowcharts, not programming language.
- The flowchart for each subroutine, if any, must appear on a separate page. Each subroutine begins with a terminal symbol with the subroutine name and a terminal symbol labeled return at the end.
- Draw arrows between symbols with a straight edge and use arrowheads to indicate the direction of the logic flow.

Guidelines for drawing a flowchart Flowcharts are usually drawn using standard symbols; however, some special symbols can also be developed when required. Some standard symbols frequently required for flowcharting many computer programs are shown in Fig.1.35.

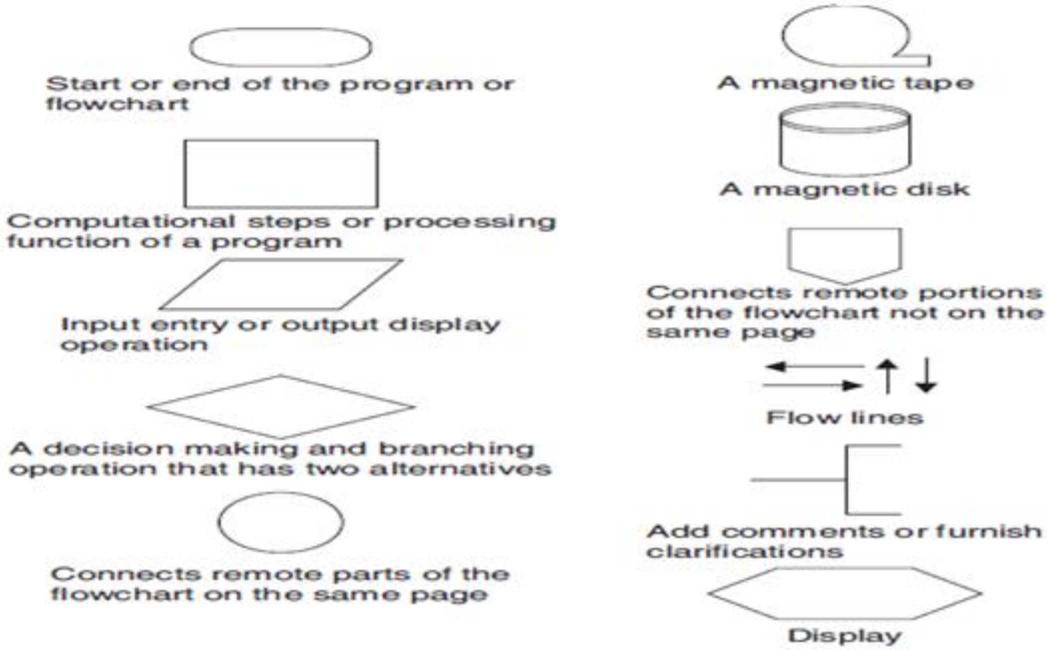
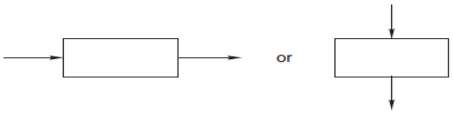

Start or end of the program or flowchart
A magnetic tape
A magnetic disk
Computational steps or processing function of a program
Connects remote portions of the flowchart not on the same page
Input entry or output display operation
Flow lines
A decision making and branching operation that has two alternatives
Add comments or furnish clarifications
Connects remote parts of the flowchart on the same page
Display

Fig 1.35: Flow chart Symbols

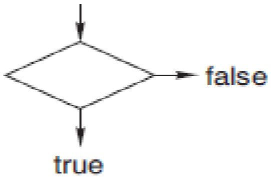
The following are some guidelines in flowcharting.

- In drawing a proper flowchart, all necessary requirements should be listed out in a logical order.
- There should be a logical **start** and **stop** to the flowchart.
- The flowchart should be clear, neat, and easy to follow. There should be no ambiguity in understanding the flowchart.

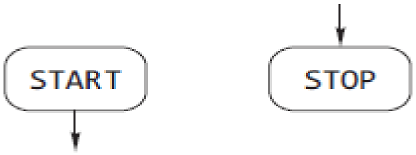
- The usual direction of the flow of a procedure or system is from left to right or top to bottom.
- Only one flow line should emerge from a process symbol.



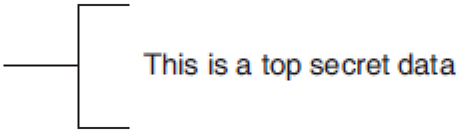
- Only one flow line should enter a decision symbol, but two or three flow lines, one for each possible answer, can leave the decision symbol.



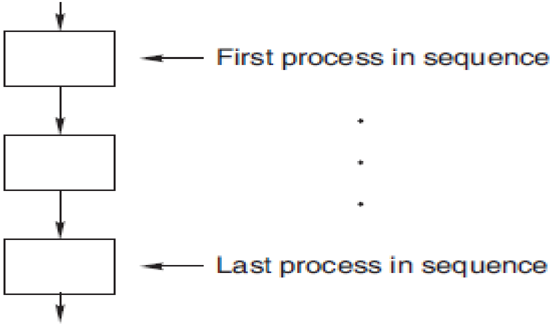
- Only one flow line is used in conjunction with a terminal symbol.



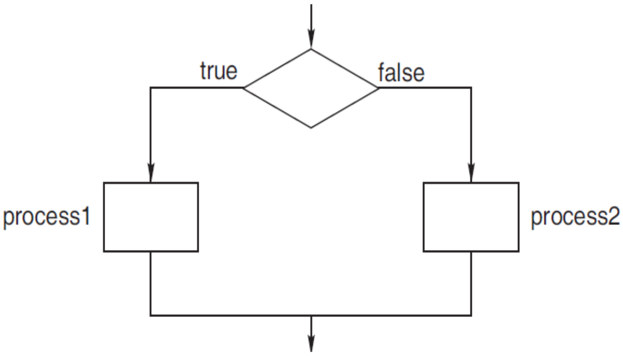
- The writing within standard symbols should be brief. If necessary, the annotation symbol can be used to describe data or computational steps more clearly.



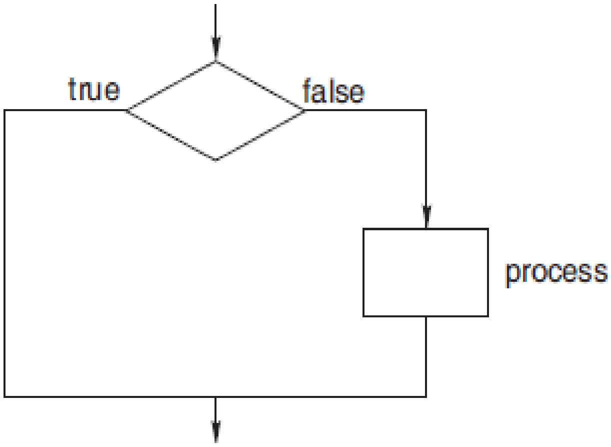
- If the flowchart becomes complex, connector symbols should be used to reduce the number of flow lines. The intersection of flow lines should be avoided to make the flowchart a more effective and better way of communication.
- The validity of the flowchart should be tested by passing simple test data through it.
- A *sequence* of steps or processes that are executed in a particular order is shown using process symbols connected with flow lines. One flow line enters the first process while one flow line emerges from the last process in the sequence.



- *Selection* of a process or step is depicted by the decision making and process symbols. Only one input indicated by one incoming flow line and one output flowing out of this structure exists. The decision symbol and the process symbols are connected by flow lines.



- *Iteration* or *looping* is depicted by a combination of process and decision symbols placed in proper order. Here flow lines are used to connect the symbols and depict input and output to this structure.



Advantages of using flowcharts

- *Communication*: Flowcharts are a better way of communicating the logic of a system to all concerned.
- *Effective analysis*: With the help of flowcharts, problems can be analyzed more effectively.
- *Proper documentation*: Program flowcharts serve as a good program documentation needed for various purposes.
- *Efficient coding*: Flowcharts act as a guide or blueprint during the systems analysis and program development phase.
- *Proper debugging*: Flowcharts help in the debugging process.
- *Efficient program maintenance*: The maintenance of an operating program becomes easy with the help of a flowchart.

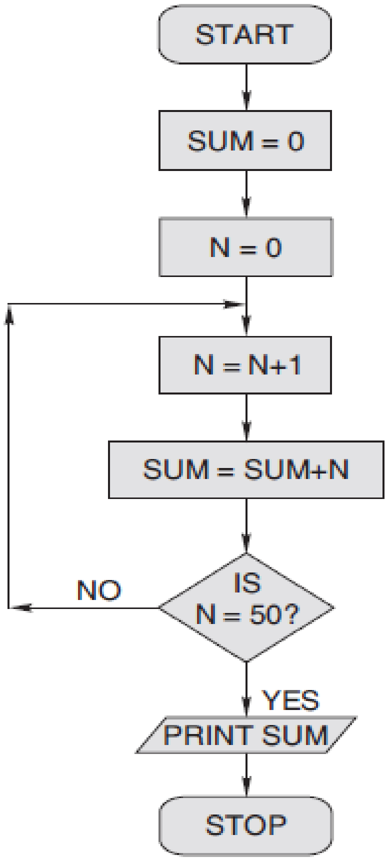
Limitations of using flowcharts

- *Complex logic*: Sometimes, the program logic is quite complicated. In such a case, a flowchart becomes complex and clumsy.
- *Alterations and modifications*: If alterations are required, the flowchart may need to be redrawn completely.
- *Reproduction*: Since the flowchart symbols cannot be typed in, the reproduction of a flowchart becomes a problem.
- *Loss of objective*: The essentials of what has to be done can easily be lost in the technical details of how it is to be done.

Flowcharting examples A few examples on flowcharting are presented for a proper understanding of the technique. This will help the student in the program development process at a later stage.

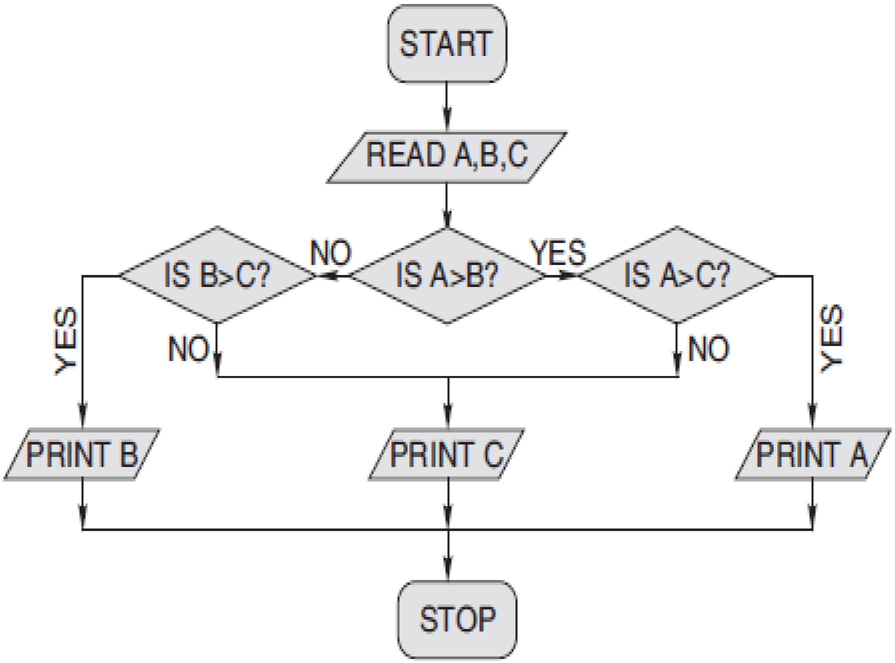
Ex 24. Draw a flowchart to find the sum of the first 50 natural numbers.

Solution



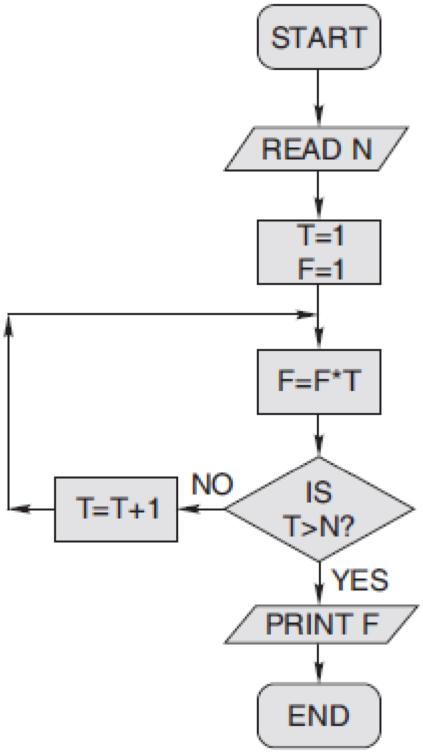
Ex 25. Draw a flowchart to find the largest of three numbers *A*, *B*, and *C*.

Solution



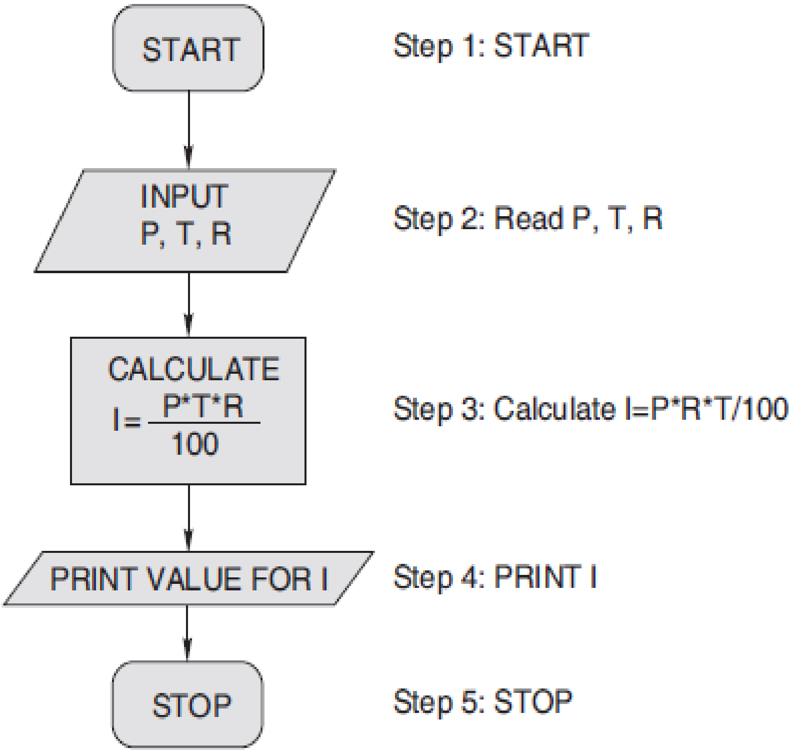
Ex 26. Draw a flowchart for computing factorial N (N!) where $N! = 1 \times 2 \times 3 \times \ldots \times N$.

Solution



Ex 27. Draw a flowchart for calculating the simple interest using the formula $SI = (P * T * R)/100$, where P denotes the principal amount, T time, and R rate of interest. Also, show the algorithm in step-form.

Solution



Step 1: START

Step 2: Read P, T, R

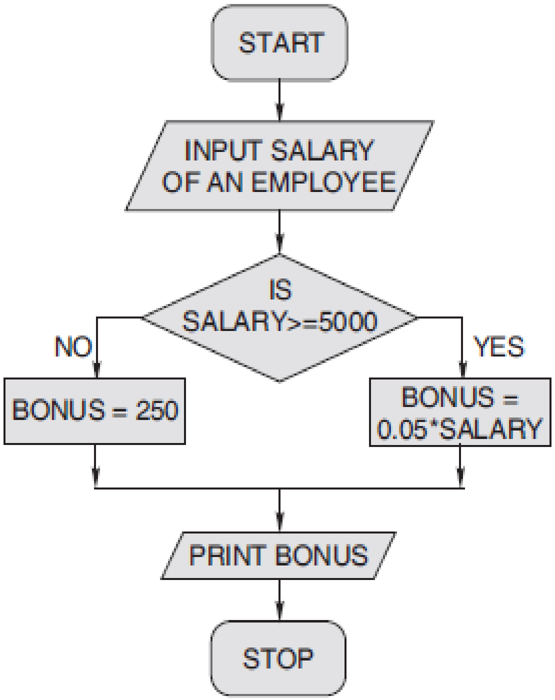
Step 3: Calculate I=P*R*T/100

Step 4: PRINT I

Step 5: STOP

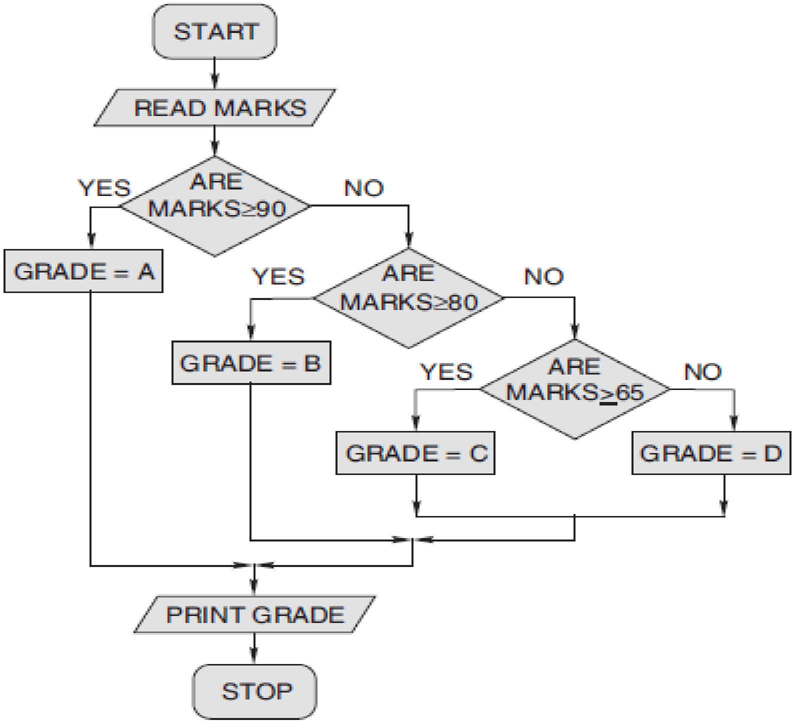
Ex 28. The XYZ Construction Company plans to give a 5% year-end bonus to each of its employees earning Rs 5,000 or more per year, and a fixed bonus of Rs 250 to all other employees. Draw a flowchart and write the step-form algorithm for printing the bonus of any employee.

Solution

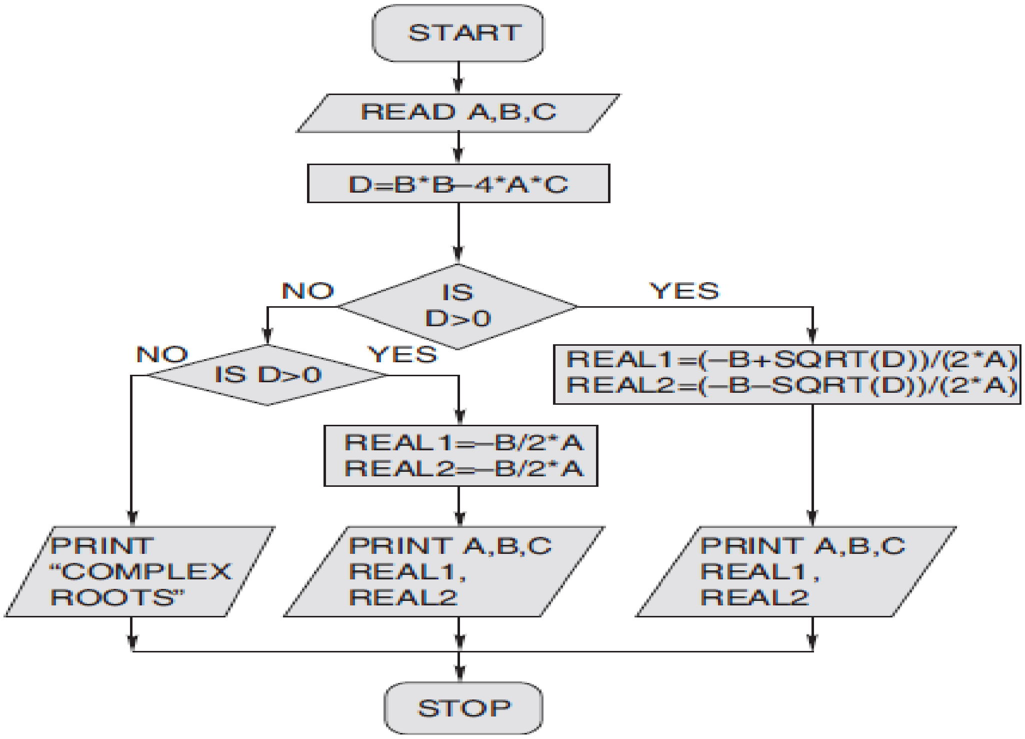


Ex 29. Prepare a flowchart to read the marks of a student and classify them into different grades. If the marks secured are greater than or equal to 90, the student is awarded Grade *A*; if they are greater than or equal to 80 but less than 90, Grade *B* is awarded; if they are greater than or equal to 65 but less than 80, Grade *C* is awarded; otherwise Grade *D* is awarded.

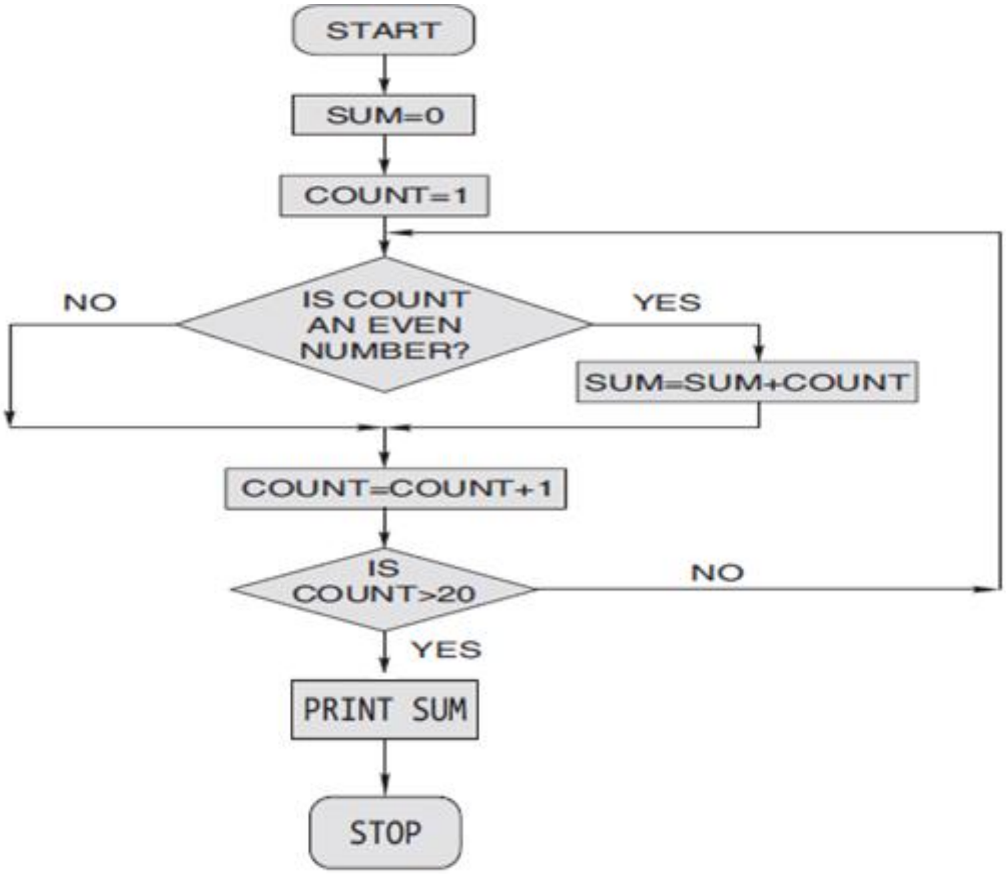
Solution



Ex 30. Draw a flowchart to find the roots of a quadratic equation.
Solution



Ex 31. Draw a flowchart for printing the sum of even terms contained within the numbers 0 to 20.
Solution



1.20 Strategy for designing algorithms

Now that the meaning of algorithm and data has been understood, strategies can be devised for designing algorithms. The following is a useful strategy.

Investigation step

1. *Identify the outputs needed.*
   This includes the form in which the outputs have to be presented. At the same time, it has to be determined at what intervals and with what precision the output data needs to be given to the user.
2. *Identify the input variables available*.
   This activity considers the specific inputs available for this program, the form in which the input variables would be available, the availability of inputs at different intervals, and the ways in which the input would be fed to the transforming process.
3. *Identify the major decisions and conditions*.
   This activity looks into the conditions imposed by the need identified and the limitations of the environment in which the algorithm has to be implemented.
4. *Identify the processes required to transform inputs into required outputs.*
   This activity identifies the various types of procedures needed to manipulate the inputs, within the bounding conditions and the limitations mentioned in step 3, to produce the needed outputs.
5. *Identify the environment available.*
   This activity determines the kind of users and the type of computing machines and software available for implementing the solution through the processes considered in steps.

Top-down development step

1. *Devise the overall problem solution by identifying the major components of the system.*
   The goal is to divide the problem solution into manageable small pieces that can be solved separately.
2. *Verify the feasibility of breaking up the overall problem solution.*
   The basic idea here is to check that though each small piece of solution procedure are independent, they are not entirely independent of each other, as they together form the whole solution to the problem. In fact, the different pieces of solution procedures have to cooperate and communicate in order to solve the larger problem.

Stepwise refinement

1. *Work out each and every detail for each small piece of manageable solution procedure.*
   Every input and output dealt with and the transformation algorithms implemented in each small piece of solution procedure, which is also known as process, is detailed. Even the interfacing details between each small procedure are worked out.
2. *Decompose any solution procedure into further smaller pieces and iterate until the desired level of detail is achieved.*
   Every small piece of solution procedure detailed in step 1 is checked once again. If necessary any of these may be further broken up into still smaller pieces of solution procedure till it can no more be divided into meaningful procedure.
3. *Group processes together which have some commonality*.
   Some small processes may have to interface with a common upper level process. Such processes may be grouped together if required.
4. *Group variables together which have some appropriate commonality*.
   Certain variables of same type may be dealt as elements of a group.
5. *Test each small procedure for its detail and correctness and its interfacing with the other small procedures.*

Walk through each of the small procedures to determine whether it satisfies the primary requirements and would deliver the appropriate outputs. Also, suitable tests have to be carried out to verify the interfacing between various procedures. Hence, the top-down approach starts with a big and hazy goal. It breaks the big goal into smaller components. These components are themselves broken down into smaller parts. This strategy continues until the designer reaches the stage where he or she has concrete steps that can actually be carried out.

It has to be noted that the top-down approach does not actually take into account any existing equipment, people, or processes. It begins with a "clean slate" and obtains the optimal solution. The top-down approach is most appropriate for large and complex projects where there is no existing equipment to worry about. However, it may be costly because, sometimes, the existing equipments may not fit into the new plan and it has to be replaced. However, if the existing equipments can be made to fit into the new plan with very less effort, it would be beneficial to use it and save cost.

1.21 Tracing an algorithm to depict logic

- An algorithm is a collection of some procedural steps that have some precedence relation between them.
- Certain procedures may have to be performed before some others are performed. Decision procedures may also be involved to choose whether some procedures arranged one after other are to be executed in the given order or skipped or implemented repetitively on fulfillment of conditions arising out of some preceding manipulations.
- Hence, an algorithm is a collection of procedures that results in providing a solution to a problem. *Tracing* an algorithm primarily involves tracking the outcome of every procedure in the order they are placed.
- *Tracking* in turn means verifying every procedure one by one to determine and confirm the corresponding result that is to be obtained.
- This in turn can be traced to offer an overall output from the implementation of the algorithm as a whole. Consider Example 26 given in this chapter for the purpose of tracing the algorithm to correctly depict the logic of the solution. Here at the start, the "mark obtained by a student in a subject" is accepted as input to the algorithm. This procedure is determined to be essential and alright. In the next step, the marks entered are compared with 90. As given, if the mark is greater than 90, then the mark obtained is categorized as Grade A and printed, otherwise it is be further compared. Well, this part of the algorithm matches with the requirement and therefore this part of the logic is correct.

1.22 Specification for converting algorithms into programs

By now, the method of formulating an algorithm has been understood. Once the algorithm, for solution of a problem, is formed and represented using any of the tools like step-form or flowchart or pseudo code, etc., it has to be transformed into some programming language code. This means that a program, formed by a sequence of program instructions belonging to a programming language, has to be written to represent the algorithm that provides a solution to a problem. Hence, the general procedure to convert an algorithm into a program is given as follows:

- *Code the algorithm into a program*—understand the syntax and control structures used in the language that has been selected and write the equivalent program instructions based upon the algorithm that was created. Each statement in an algorithm may require one or more lines of programming code.

- *Desk-check the program*—Check the program code by employing the desk-check method and make sure that the sample data selected produces the expected output.
- *Evaluate and modify, if necessary, the program*—Based on the outcome of desk-checking the program, make program code changes, if necessary, or make changes to the original algorithm, if need be.
- *Do not reinvent the wheel*—If the design code already exists, modify it, do not remake it.

Because the reader has not yet been introduced to the basics of the C language, the reader has to accept the use of certain instructions like #include <stdio.h>, int main(), printf(), scanf(), and return without much explanation at this stage in the example program being demonstrated below. However, on a very preliminary level, the general form of a C program and the use of some of the necessary C language instructions are explained briefly as follows:

1. All C programs start with:
   ```
   #include <stdio.h>
   int main ()
   {
   ```
2. In C, all variables must be declared before using them. So the line next to the two instruction lines, given in step 1, should be any variable declaration that is needed.
   For example, if a variable called “a” is supposed to store an integer, then it is declared as follows:
   ```
   int a;
   ```
3. Here, scanf() is used for inputting data to the C program and printf() is used to output data on the monitor screen.
4. The C program has to be terminated with a statement given below:
   ```
   return 0;
   }
   ```

To demonstrate the procedure of conversion from an algorithm to a program in C, an example is given below.

Problem statement Write the algorithm and the corresponding program in C for adding two integer numbers and printing the result.

Solution

Algorithm

```
1. START
2. DECLARE A AND B AS INTEGER VARIABLES
3. PRINT “ENTER TWO NUMBERS”
4. INPUT A, B
5. R = A + B
6. PRINT “RESULT =”
7. PRINT R
8. STOP.
```

Program in C

```
int main( )
{
    int A, B;
    printf(“\n ENTER TWO NUMBERS:”);
    scanf(“%d%d”,&A,&B);
    R = A + B;
    printf(“\n RESULT = ”);
    printf(“%d”,R);
    return 0;
}
```

Exercise

1. What do you mean by a program?
2. Distinguish between system software and application software.
3. State the advantages and disadvantages of machine language and assembly language.
4. Compare and contrast assembly language and high-level language.
5. Differentiate between 3GL and 4GL.
6. What is a translator?
7. What are the differences between a compiler and an interpreter?
8. Briefly explain the compilation and execution of a program written in high-level language.
9. Briefly explain linker and loader? Is there any difference between them?
10. Explain linking loader and linkage editor?
11. Classify the programming languages.
12. What is a functional language?
13. What is object-oriented language? Name five object-oriented programming languages. State the most common features of object-oriented programming.
14. What do you mean by structured programming? State the properties of structured programming.
15. What is top-down analysis? Describe the steps involved in topdown analysis.
16. What is a structured code?
17. What is an algorithm?
18. Write down an algorithm that describes making a telephone call. Can it be done without using control statements?
19. Write algorithms to do the following:
    - i. Check whether a year given by the user is a leap year or not.
    - ii. Given an integer number in seconds as input, print the equivalent time in hours, minutes, and seconds as output. The recommended output format is something like: 7,322 seconds is equivalent to 2 hours 2 minutes 2 seconds.
    - iii. Print the numbers that do not appear in the Fibonacci series. The number of terms to be printed should be given by the user.
    - iv. Convert the binary equivalent of an integer number.
    - v. Find the prime factors of a number given by the user.
    - vi. Check whether a number given by the user is a Krishnamurty number or not. A Krishnamurty number is one for which the sum of the factorials of its digits equals the number. For example, 145 is a Krishnamurty number.
    - vii. Print the second largest number of a list of numbers given by the user.
    - viii. Print the sum of the following series:
        - i. 1 x2/2! + x4/4! + up to n terms where n is given by the user
        - ii. 1 1/2 + 1/3 up to n terms where n is given by the user
        - iii. 1 + 1/2! + 1/3! + up to n terms where n is given by the user
20. Define a flowchart. What is its use?
21. Are there any limitations of a flowchart?
22. Draw a flowchart to read a number given in units of length and print out the area of a circle of that radius. Assume that the value of pi is 3.14159. The output should take the form: The area of a circle of radius __________ units is __________ units.
23. Draw a flowchart to read a number N and print all its divisors.
24. Draw a flowchart for computing the sum of the digits of any given number.
25. Draw a flowchart to find the sum of N odd numbers given.

26. Draw a flowchart to compute the sum of squares of integers from 1 to 50.
27. Write a program to read two integers with the following significance. The first integer value represents a time of day on a 24-hour clock, so that 1245 represents quarter to one mid-day. The second integer represents time duration in a similar way, so that 345 represents three hours and 45 minutes. This duration is to be added to the first time and the result printed out in the same notation, in this case 1630 which is the time 3 hours and 45 minutes after 1245. Typical output might be start time is 1415. Duration is 50. End time is 1505.
28. Fill in the blanks.

    i. A program flowchart indicates the __________ to be performed and the __________ in which they occur.

    ii. A program flowchart is generally read from __________ to__________.

    iii. Flowcharting symbols are connected together by means of __________.

    iv. A decision symbol may be used in determining the __________ or __________ of two data items.

    v. __________ are used to join remote portions of a flowchart.

    vi. __________ Connectors are used when a flowchart ends on one page and begins again on another page.

    vii. A __________ symbol is used at the beginning and end of a flowchart.

    viii. The flowchart is one of the best ways of __________ a program.

    ix. To construct a flowchart, one must adhere to prescribed symbols provided by the __________.

    x. The programmer uses a __________ to aid him in drawing flowchart symbols.