

UNIT - 1.1

INTRODUCTION TO COMPUTER

1.1.1 INTRODUCTION

Definition

A computer is an electronic machine, devised for performing calculations and controlling operations that can be expressed either in logical or numerical terms.

Applications

The applications domain of a computer depends totally on human creativity and imagination it covers a huge area of applications including education, industries, government medicine, scientific research, low and even music and arts.

- Millions of complex calculations can be done in a mere fraction of time
- Difficult decisions can be made with unerring accuracy for comparatively little cost

1.1.2 CHARACTERISTICS OF COMPUTER

Speed

Computer process data at an extremely fast rate – millions of instructions per second in few seconds, a computer can perform a huge task that a normal human being may take days or even years to complete.

The speed of a computer is calculated in Mhz

Accuracy

Besides efficiency, computer are accurate as well. The level of accuracy depends an the instructions and the type of machine being used.

Diligence

Computer being a machine does not suffer form the human trailts of tiredness and lack of concentration

Reliability

Reliability is the measurement of performance of a computer, which is measured against some predetermined standard for operation without any failure.

Storage capability

The main memory of the computer is relatively small and it can hold only a certain amount of information, therefore, the data is stored on secondary storage devices such as magnetic tape or disks.

Versatility

It can perform multiple tasks simultaneously with great ease. For example, at one moment it can be used to draft a letter, another moment it can be used to play music and in between, one can print a document as well.

All this work is possible by changing the program.

Resource sharing

It made the sharing of costly resources like printer possible.

Apart from device sharing data and information can also be shared among group of computers, thus creating a large information and knowledge base.

1.1.3 EVOLUTION OF COMPUTERS

- In the beginning, when the task was simply counting or adding, people used either their fingers or pebbles along lines in the sand.

In order to simplify the process of counting, people in *Asia Minor* built a counting device called *abacus*. This device allowed users to do calculations using a system of sliding beads arranged on a rack.

- With the passage of time, many computing devices such as *Napier bones* and *slide rule* were invented.
- In 1642, a French mathematician, *Blaise Pascal* invented the first functional automatic calculator.

This brass rectangular box, also called a *Pascaline*, used eight movable dials to add sums eight figures long.

- In 1694, German mathematician *Gottfried Wilhelm von Leibniz* extended Pascal's design to perform multiplication, division and to find square root. This machine is known as *Stepped Reckoner*.
- The real beginnings of computers as we know them today, however, lay with an English mathematics professor, *Charles Babbage*.

In 1822, he proposed a machine to perform differential equations, called a *Difference Engine*.

- In 1889, *Herman Hollerith*, who worked for US census bureau, also applied the *Jacquard loom* concept to computing.

Hollerith's method used cards to store data, which he fed into a machine that compiled the results mechanically.

- **Mark I**, which was built as a partnership between *Harvard Aiken and IBM* in 1944.
This electronic calculating machine used relays and electromagnetic components to replace mechanical components
- In 1946, **John Eclcert** and **John Mauchy** of developed **ENIAC** (electronic numerical integrator and calculator)
Thus computer used electronic vacuum tubes to make internal parts of the computer
- Eckert and mauchy also proposed the development of **EDVAC** (electronic discrete variable automatic computer.
It was the first electronic computer to use the stored program concept introduced by **John von Neumann**.
- In 1949, at the Cambridge university, Maurice wilkes developed **EDSAC** (electronic delay storage automatic calculator)
This machine used mercury delay lines for memory and vacuum tubes for logic.
- The Eckert – mauchy corporation manufactured **UNIVAC** (universal automatic computer) in 1951 and its implementation marked the real beginning of the computer era.

1.1.4 COMPUTER GENERATIONS

1.1.4.1 First Generation (1940-56) : Vacuum Tube

First generation computer were vacuum tubes/thermionic value based machines these computers used vacuum tubes for circuitry and magnetic drums for memory.

A magnetic drum is a metal cylinder coated with magnetic iron-oxide material on which data and programs can be stored.

Input was based on punched cards and paper tape and output was in the form of printouts.

For example: ENIAC, EDVAC AND UNIVAC.

Characteristics of First Generation Computers.

- These computers were based on vacuum tube technology.
- These were the fastest computing devices of their time.
- These computers were very large, and required a lot of space for installation.
- These were non-portable and very slow equipments.

1.1.4.2 Second Generations (1956-63): Transistors

A transistor is made up of semiconductor material like germanium and silicon. It usually had three leads and performed electrical functions such as voltage, current or power amplification with low power requirement.

Since transistor is a small device, the physical size of computers was greatly reduced.

1.4 Fundamentals of Computing and Programming

Computers became smaller, faster, cheaper, energy-efficient and more reliable than their predecessors.

Magnetic cores- were used as primary memory and magnetic disks as secondary storage devices. However, they still relied on punched cards for input and printouts for output.

For example: PDP – 8 , IBM 1401 and IBM 7090

Characteristics of Second Generation Computer.

- These machines were based on transistor technology
- These were smaller as compared to the first generation computers.
- These were more portable and generated less amount of heat.

1.1.4.3 Third Generation (1964 – Early 1970), Integrated Circuits

The development of the integrated circuit was the trait of the third generation computer. Also called an ic, an integrated circuit consists of a single chip with many components such as transistors and resistors fabricated on it.

Integrated circuit replaced several individually wired transistor. This development made computers smaller in size, reliable and efficient.

Instead of punched cards and printouts, users interacted with third generation computers through keyboards and monitors and interfaced with operating system.

For example : NCR 395 and B6500

Characteristic of Third Generation Computer

- These computers were based on integrated circuit (ic) technology.
- They were able to reduce computational time from micro seconds to nano seconds.
- Extensive use of high – level language became possible

1.1.4.4 Fourth Generation (Early (1970 – Till Date) Microprocessors)

The technology of this generation was still based on the intergrated circuit, these have been made readily available to use because of the development of the microprocessor.

The fourth generation computers led to an era of large scale integration (LSI) and very large scale integration (vlsi) technology. LSI technology allowed thousands of transistors to be constructed on one small slice of silicon material whereas vlsi squeezed hundreds of thousands of components on to a single ewp

ULTRA – large scale integration (ULSI) increased that number into millions the fourth generation computer became more powerful compact, reliable and affordable.

For example: apple ii, attair 8800 and CRAY-1

Characteristics of Fourth Generation Computers

- Fourth generation computers are microprocessor based systems
- These computers are very small
- GUI and pointing devices enable users to learn to use the computer quickly
- Interconnection of computers leads to better communication and resource sharing

1.1.4.5 Fifth Generation (Present and Beyond): Artificial Intelligence

A computer would learn from its mistakes and possess the skill of experts the starting point for the fifth generation of computers has been set in the early 1990. The expert system it defined as a computer information system that attempts to mimic the thought process and reasoning of experts in specific areas three characteristics can be identified with the fifth generation computer these are.

Mega chips

Fifth generation computers will use super large scale integrated (SLSI) chips, which will result in the production of microprocessor having millions of electronic components on a single chip

Parallel processing

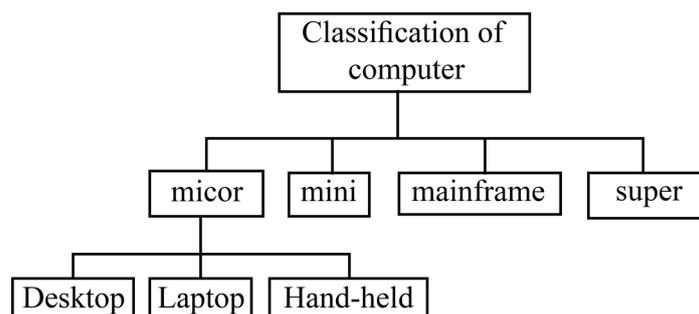
A computer using parallel processing accesses several instructions at once and works on them at the same time through use of multiple central processing units.

Artificial intelligence: (AI)

AI comprises a group of related technologies expert systems (ES), natural language processing (NLP) speech recognition, vision recognition and robotics.

1.1.5 CLASSIFICATION OF COMPUTERS

Four major categories: micro, mini, mainframe and super computers



Micro computers

A micro computer is a small, low cost digital computer, which usually consists of a microprocessor, a storage unit, an input channel and an output channel, all of which may be on one chip inserted into one or several pc boards.

IBM – pc, Pentium 100, ibm-pc Pentium 200 and *Apple Macintosh* are some of the example of micro computers

Micro computers include desktop, laptop and hand – held models such as PDAS (personal digital assistants).

Desktop computer

Desktop computer also known as personal computer (pc) is principally intended per stand alone use by an individual micro computer typically consist of a system unit a display monitor, a keyboard internal hard disk storage and other peripheral devices.

Some of the major personal computer manufactures are *Apple, IBM, Dell* and *Hewlett Packard*.

Laptop

A laptop is a portable computer that is a user can carry it around. Laptops are small computer enclosing all the basic features of a normal desktop computer.

The biggest advantage of this computer is that one can use this computer anywhere and at anytime, especially when are is travelling

Hand held computers

A hard-held, also called personal digital assistant (PDA), is a computer that can conveniently be stored in a pocket and used while the user is holding it.

PDA's are essentially small portable computers and are slightly bigger than the common calculators.

Some example of PDA's are *Apple Newton, Casio Cassiopeia* and *Franklin ebook man*

Mini computers

The mini computer is a small digital computer whose process and storage capacity is lesser than that of a mainframe, but more than that of micro computer.

Its speed of processing data is in between that of a mainframe and a micro computes, generally, it is used as desktop device that is often connected to a mainframe in order to perform the auxiliary operations.

Mini computers are usually multi-user systems, so these are used in interactive applications in industries, research organisations colleges and universities.

High – performance workstations with graphics I/o capability use mini computers

Some of the widely used mini computers are *PDP II, IBM (8000 series)* and *VAX 7500*.

Mainframe computer

A mainframe is an ultra – high performance computer made for high – volume, processor – intensive computing. It consists of a high end computer processor, with related peripheral devices, capable of supporting large volumes of data processing systems and extensive data storage and retrieval.

Mainframes are the second largest of the computer family, the largest being super computers.

Mainframe allows its user to maintain large information storage at a centralised location and be able to access and process this data from different computers located at different locations

It is typically used by large businesses and for scientific purpose.

Examples of mainframe computers are IBM's *E5000*, *VAX8000* and *CDC6600*.

Super computers

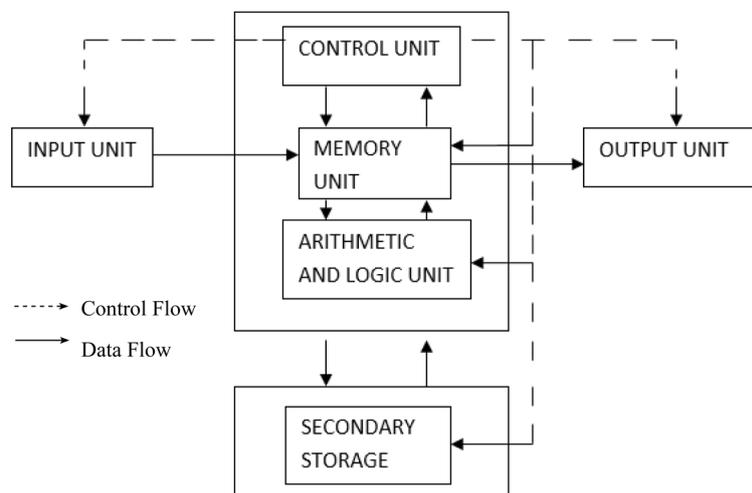
Super computers are the special purpose machine, which are specially designed to maximise the numbers of *FLOPS* (floating point operation per second). Any computer below one *gigaflop/sec* is not considered a super computer. A super computer has the highest processing speed at a given time for solving scientific and engineering problems.

Essentially, it contains a number of cpu, that operate in parallel to make it faster. Its processing speed lies in the range of 400 – 10,000 *MFLOP's* (millions of floating point operation per second).

Super computers are used to solve multivariant mathematical problems of existent physical processes, such as aerodynamics, metrologis, and plasms physics.

The largest commercial use of super computers is in the entertainment advertising industry, *CRAY-3*, *Cyber 205* and *PARAM* are some well known super computers.

1.1.6 BASIC COMPUTER ORGANIZATION



The block diagram of the computer system have the following three units, each functional unit corresponds to their basic operations performed as described in details.

- (a) Input unit
- (b) Central processing unit
- (c) Output unit

(a) Input Unit

- Accept data and instructions from the outside world.
- Convert it to a form that the computer can understand
- Supply the converted data to the computer system for further processing
- The input unit is used to send information or instructions or commands to the computer. The data received from the input unit is immediately stored in main memory and then processed.

Following are some of the input devices.

- (i) Keyboard
- (ii) Mouse
- (iii) Light pen
- (iv) Joystick
- (v) OCR (optical character recognizer)
- (vi) MICR (magnetic ink character recognizer)
- (vii) OMR (optical mark recognizer)

(b) Central Processing Unit (CPU)

- It performs all calculations and all decisions.
- It controls and co-ordinates all units of the computer
- It interprets instructions of a program
- It stores data temporarily and monitors external requests.

The CPU is sub-divided into the following sub-systems.

- (i) Control unit
- (ii) Arithmetic and logical unit
- (iii) Memory unit
 - (a) Primary storage
 - (b) Secondary storage.

(i) Control unit

The control unit instructs the computer how to carry out program instructions. It directs the flow of data between memory and arithmetic logical unit.

The input unit does not know when to receive data and where to put the data in the storage unit after receiving it. Similarly, the control unit instructs the input unit where to store the data after receiving it from the user.

In the same way, it controls the flow of data and instructions from the storage unit to

ALU during program execution the control unit fetches instructions from the primary memory, decodes them to determine the operations required, and then sets up instructions execution.

Eg. To add two numbers or to read a character from a keyboard. A number registers are associated with the control unit.

(ii) Arithmetic And Logical Unit

Arithmetic and logical unit performs all the arithmetic and logical operations. Arithmetic operations like addition, subtraction, multiplication and logical operations, such as comparisons are performed in ALU.

All calculations are performed in the arithmetic and logical unit (ALU) of the computer ALU also does comparisons and take decision .

Example: it can check if the number A is less than equal to or greater than the number B. once the calculations or the logical operation is performed by ALU, then the result is transferred to the storage unit.

(iii) Memory unit

Memory is the part of computer which holds data for processing and other information it is also called as **main memory** or **primary memory**.

A device that stores program instructions or data used by the cpu when performing a given function.

Memory is a device, which is used to store information temporarily/permanently, it is the place where the information is safekept. Secondary memory, such as disk storage, is functionalty considered I/O because it is accessed through the I/O system.

(a) Primary storage

The primary storage is also called as “**main memory**” stores and access information very fastly. This is generally used to hold the program being currently executed in the computer, the data being received from the input unit, the intermediate and final results of the program.

Primary storage is also known as **system memory, internal, temporary** and “**RAM**”

- Installed on the main computer board (motherboard)
- Typically comprised of ICs (integrated circuits)
- Fast access – usually in the order of nano seconds

(b) Secondary storage

The secondary storage is also known as **Auxiliary Storage** it may store several programs, documents, databases etc.

The program that we want to run on the computer is first transferred to the primary memory before it can run. Similarly, after running the program if need to save the result, we will transfer them to the secondary storage.

The secondary memory is slower and cheaper than the primary memory. Some of the commonly used secondary memory devices are *Floppy diskette, Zip diskette, Hard disk and Magnetic disks* and *Tapes* etc.

(c) Output unit

Devices used to get the response or result of a process from the computer is called output output unit is the communication between the user and the computer.

The output unit of a computer provider the information and results of a computation to the outside world.

Computers do not work in the decimal system, they work in the binary system. Therefore if required, the output unit also converts the binary data into a form that users can understand.

Commonly used output devices are.

- Visual display unit (*VDU*) or monitor
- Printer
- Computer output microfilm
- Plotter.

1.1.7 NUMBER SYSTEMS

Introduction

A number is required for counting or to express the amount of some quantity it consists of a group of symbols called digits, which are arranged in a definite manner. There can be many ways in which the digits can be arranged to form a number. This gives rise to what we call a number system.

The *decimal number* system which has ten digits (0, 1, 2,.....9), the *octal system* has eight digits (0,1,2,....7) the *hexadecimal* system has sixteen digits (0,1,2,.....9,a,b,c,d,e,f), the *binary number* system has only two (0,1)

The number of digits in a system is called *radix* or *base*. Thus the decimal system may be called as radix-10 system, the binary system as radix – 2, system.

The number system are basically two types

- (i) Non – positioning number system
- (ii) Positioning number system.

(i) Non –positional number system

Using this system, the symbols i.e, I for 1, II for 2, III for 3, IIII for 4, iiii for 5 etc, are used

In non-positioning number system each symbol represents the same values regardless of its position in the number. The symbols are simply added to find out the value of a particular number.

(ii) Positional number system

The positional number user only few symbol called digits

Example:

The decimal number 3977.39(10)

3×10^3	+	9×10^2	+	7×10^1	+	7×10^0	+	3×10^{-1}	*	9×10^{-2}
1000 th position		100 th position		10 th position		Unit position		1/10 th position		1/100 th position

The following four positional number system are commonly used.

- (1) Decimal number system
- (2) Binary number system
- (3) Octal number system
- (4) Hexadecimal system

(1) Decimal number system

In this number system the base or radix is 10 and there are altogether ten number i.e, 0,1,2,3,4,5,6,7,8,9.

A number in a radix system would be written as $(a_n a_{n-1} a_{n-2} \dots \dots a_0 a_{-1} a_{-2} \dots \dots a_{-m})$

In this representation a_n is called the 'Most-Significant Digit' (MSD) of the number and a_{-m} is called the 'Least Significant Digit' (LSD).

Example:

In decimal number system $3967_{(10)}$ or 3967 Consists the digits

7 in the unit position (7×10^0)

6 in the unit position (6×10^1)

9 in the unit position (9×10^2)

3 in the unit position (3×10^3)

It values can be written as

$$3 \times 10^3 + 9 \times 10^2 + 6 \times 10^1 + 7 \times 10^0$$

$$3 \times 1000 + 9 \times 100 + 6 \times 10 + 7 \times 1$$

$$3000 + 900 + 60 + 7$$

$$3967$$

$$3967_{(10)} = 3967$$

(2) Binary number system

The binary system uses only two digits i.e., 0's and 1's. The base or radix of binary number system is 2 because it contains only two numbers.

User

- The circuits in computer have to be handled by two binary digits or bits rather than decimal number
- The computer only identifies signals in the form of digital pulses, which represent either high or low voltage.
- Everything that can be done with decimal numbers can also be done using binary numbers.

Example:

Convert the binary number $100111_{(2)}$ to decimal numbers.

Solution: $100111_{(2)} = ?_{(10)}$

6 th	5 th	4 th	3 rd	2 nd	1 st	Position
1	0	0	1	1	1	Given Binary Number
$\times 2^5$	$\times 2^4$	$\times 2^3$	$\times 2^2$	$\times 2^1$	$\times 2^0$	Weights
$1 \times 32 + 0 \times 16 + 0 \times 8 + 1 \times 4 + 1 \times 2 + 1 \times 1$						
32	+	0	+	0	+	4
+	2	+	1	=	$39_{(10)}$	
$100111_{(2)}$						= $39_{(10)}$

Decimal number	Binary equivalent
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

(3) Octal number system

The number system with base or radix digit (8) is known as octal number system. There are only eight digits i.e., 0,1,2,3,4,5,6,7

Each position in this number system represents a power of the base(8)

The decimal equivalent of the octal number $3077_{(8)}$ is

$$\begin{aligned}
 3077_{(8)} &= 3 \times 8^3 + 0 \times 8^2 + 7 \times 8^1 + 7 \times 8^0 \\
 &= 3 \times 512 + 0 \times 64 + 7 \times 8 + 7 \times 1 \\
 &= 1536 + 0 + 56 + 7 = 1599_{(10)} \\
 3077_{(8)} &= 1599_{(10)}
 \end{aligned}$$

Decimal	Octal	Binary Equivalent
0	0	000
1	1	001
2	2	010
3	3	011
4	4	100
5	5	101
6	6	110
7	7	111

(4) Hexadecimal number system

The base 16 suggests choices of 16 single character digits or symbol

The first 10 digits are digits of decimal system (0 to 9) and the remaining 6 digits are denoted by (A to F) representing decimal value (10 to 15) where a = 10, B = 11, C = 12, D = 13, E=14 and F = 16. The largest single digit is F or 15 (one less that the base).

Since number (0 to 9) and alphabets (A to F) are used to represent the digit in hexadecimal numbers system, it is also called the alphanumeric number systems.

Decimal	Hexadecimal	Binary Equivalent
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010

11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

Example:

Convert the hexadecimal number 39a (16) To decimal number

Solution

$$\begin{aligned}
 39A_{(16)} &= ?_{(10)} \\
 &= 3 \times 16^2 + 9 \times 16^1 + A \times 16^0 \\
 &= 3 \times 256 + 9 \times 16 + A \times 1 \\
 &= 3 \times 256 + 9 \times 16 + 10 \times 1 \\
 &= 768 + 144 + 10 \\
 &= 922_{(10)} \\
 39A_{(16)} &= 922_{(10)}
 \end{aligned}$$

1.1.8 CONVERSION OF NUMBER SYSTEM

- Conversion of Decimal Number System
- Conversion of Binary
- Conversion of Octal
- Conversion of Hexadecimal

1.1.8.1 Conversion of Decimal Number System

- ⇒ Decimal to binary system
- ⇒ Decimal to octal
- ⇒ Decimal to hexadecimal

(1) Conversion of Decimal to Binary System

Example:

Convert the 37.8125(10) decimal number to its binary equivalent

2	37	
2	18	- 1
2	9	- 0

$$\begin{array}{r}
 2 \quad \overline{4} \quad - 1 \\
 2 \quad \overline{2} \quad - 0 \\
 \quad \quad 1 \quad - 0
 \end{array}$$

Therefore, $37_{(10)} = 100101_{(2)}$

<u>Fraction</u>	<u>Radix</u>	<u>Result</u>	
0.8125 *2	=1.625	=0.625	with carry as 1
0.625 *2	=1.25	=0.25	with carry as 1
0.25 *2	=0.5	=0.50	with carry as 0
0.5 *2	=1.0	=0.00	with carry as 1

$$0.8125_{(10)} = 0.1101$$

Therefore $37.8125_{(10)} = 100101.1101_{(2)}$

Exercise:

Convert $(68)_{10}$ to binary

$$\text{Answer} = (1000100)_2$$

(2) Conversion of Decimal to-Octal System

Example:

Convert the $35.45_{(10)}$ decimal number to its octal equivalent

$$\begin{array}{r}
 8 \quad \overline{35} \\
 \quad \quad 4 \quad - 3
 \end{array}$$

Therefore, $35_{(10)} = 43_{(8)}$

<u>Fraction</u>	<u>Radix</u>	<u>Result</u>	
0.45 *8	=3.6	= 0.6	with carry as 3
0.6 *8	=4.8	= 0.8	with carry as 4
0.8 *8	=6.4	= 0.4	with carry as 6
0.4 *8	=3.2	= 0.2	with carry as 3
0.2 *8	=1.6	= 0.6	with carry as 1

$$0.45_{(10)} = 0.34631_{(8)}$$

Therefore, $35.45_{(10)} = 43.34631_{(8)}$

Exercise:

Convert decimal number 214 to its octal equivalent

Answer = 326

(3) Conversion of Decimal to Hexadecimal System

Example:

Convert the $22.64_{(10)}$ decimal number to its hexadecimal equivalent.

$$\begin{array}{r} 16 \overline{) 22} \\ \underline{16} \\ 6 \end{array}$$

Therefore, $22_{(10)} = 16_{(16)}$

<u>Fraction</u>	<u>Radix</u>	<u>Result</u>	
0.64 *16	=10.24	=10.24	with carry as 10
0.24 *16	=3.84	=0.84	with carry as 3
0.84 *16	=13.44	=0.44	with carry as 13
0.44 *16	=7.04	=0.04	with carry as 7
$0.64_{(10)} = 0.A3D7_{(16)}$			

Therefore, $22.64_{(10)} = 16.A3D7_{(16)}$

Exercise:

Convert decimal number 3509 to its hexadecimal equivalent

Answer = 13115

=DB5₍₁₆₎

1.1.8.2 Conversion of Binary System

⇒ Binary to decimal conversion

⇒ Binary to octal

⇒ Binary to hexadecimal

(1) Binary to Decimal Conversion

Example:

Convert $1101.101_{(2)}$ to its decimal equivalent

$$\begin{aligned} &= 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2 + 1 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} \\ &= 8 + 4 + 0 + 1 + 0.5 + 0 + 0.125 \\ &= 13.625_{(10)} \end{aligned}$$

Exercise:

Convert $(1000100)_2$ to its decimal equivalent

$$\text{Answer} = (68)_{10}$$

(2) Binary to Octal Conversion**Example:**

Convert the $111101100_{(2)}$ to its octal equivalent.

⇒ Group the number by 3

$$111 \quad 101 \quad 100$$

⇒ Specify the octal equivalent

$$111 \quad 101 \quad 100$$

$$7 \quad 5 \quad 4$$

Therefore, the octal number is $754_{(8)}$

Exercise:

Convert $111 \ 01 \ 111 \ 0_{(2)}$ to its equivalent octal number

$$\text{Answer} = 735_{(8)}$$

(3) Binary to Hexadecimal Conversion**Example:**

Convert the $1101100010011011_{(2)}$ to its hexadecimal equivalent.

⇒ Group the number by 4

$$1101 \quad 1000 \quad 1001 \quad 1011$$

⇒ Specify the hexadecimal equivalent

$$1101 \quad 1000 \quad 1001 \quad 1011$$

$$D \quad 8 \quad 9 \quad B$$

Therefore, the octal number is $D89B_{(16)}$

Exercise:

Convert $11111101 . 0001 \ 0011_{(2)}$ to its equivalent hexadecimal number.

$$\text{Answer} = FD13_{(16)}$$

1.1.8.3 Conversion of Octal Number System

⇒ Octal to Decimal Conversion

⇒ Octal to Binary Conversion

⇒ Octal to Hexadecimal

(1) Octal to Decimal Conversion

Example:

Convert the $51.63_{(8)}$ to its decimal equivalent.

$$\begin{aligned}
 &= 5 \times 8^1 + 1 \times 8^0 + 0.6 \times 8^{-1} + 0.3 \times 8^{-2} \\
 &= 40 + 1 + (0.075 + 0.0046875) \\
 &= 41.0796875_{(10)}
 \end{aligned}$$

Therefore, $51.63_{(8)} = 41.0796875_{(10)}$

Exercise:

$$516_{(8)} = 324$$

(2) Octal to Binary Conversion

Example:

Convert the $51.63_{(8)}$ to its binary equivalent.

$$\begin{array}{cccc}
 5 & 1 & \cdot & 6 & 3 \\
 \downarrow & \downarrow & & \downarrow & \downarrow \\
 101 & 001 & & 110 & 011
 \end{array}$$

$$101001.110011_{(2)}$$

$$51.63_{(8)} = 101001.110011_{(2)}$$

Therefore, the binary number is $101001.110011_{(2)}$

Exercise:

convert 5613 to its binary equivalent

$$\text{Answer} = 1010011110011_{(2)}$$

(3) Octal to Hexadecimal Conversion

Example:

Convert the $345.30_{(8)}$ to its hexadecimal equivalent.

Octal	↔	Binary	↔	Hexadecimal
⇒ 3	4	5	•	3 0
⇒ 011	100	101	•	<u>011</u> 0,0 0
	↓	↓		↓
⇒ 0000	1110	0101	•	0110 0000
⇒ 0	E	5	•	6 0

Therefore, the hexadecimal number is $E5.60_{(16)}$

Exercise

34530(8)

Answer = E560(16)

011	1,00	101	011	0000
↓	↓	↓	↓	↓
E	5	6	0	

1.1.8.4 Conversion of Hexadecimal Number System

⇒ Hexadecimal to Decimal Conversion

⇒ Hexadecimal to Binary

⇒ Hexadecimal to Octal

⇒ Hexadecimal to Decimal Conversion

(1) Hexadecimal to Decimal Conversion**Example:**Convert the $5B.2E_{(16)}$ to its decimal equivalent.

$$=5 \times 16^1 + B \times 16^0 + 2 \times 16^{-1} \times E \times 16^{-2}$$

$$=80 + 11 + (0.125 + 0.0546875)$$

$$=91.1796875_{(10)}$$

$$5B.2E_{(16)} = 91.1796875_{(10)}$$

Exercise $5B^2(16)$

Answer = 1458

(2) Hexadecimal to Binary ConversionConvert the $8B2F.9A_{(16)}$ to its binary equivalent.

8	B	2	F	•	9	A
↓	↓	↓	↓	•	↓	↓
1000	1011	0011	1111	•	1001	1010

$$8B2F.9A_{(16)} = 1000101100111111.10011010_{(2)}$$

Therefore, the binary number is $1000101100111111.10011010_{(2)}$ **Exercise**

8B2F

Answer = $1000101100101111_{(2)}$

(3) Hexadecimal to Octal Conversion

Example:

Convert the $BDAF.AC9_{(16)}$ to its octal equivalent.

⇒	B		D		A		F		•		A		C		9			
	↓		↓		↓		↓		•		↓		↓		↓			
	1011		1101		1010		1111		•		1010		1100		1001			
⇒	1		011		110		110		101		111		•		101	011	001	001
	↓		↓		↓		↓		↓		↓		•		↓	↓	↓	↓
	1		3		6		6		5		7		•		5	3	1	1

Therefore, the hexadecimal number is $136657.5311_{(8)}$

UNIT - 1.2

COMPUTER SOFTWARE

1.2. COMPUTER SOFTWARE

Definition

Software is a generic term for organised collection of computer data and instructions. It is responsible for controlling, integrating and managing the hardware components of a computer and to accomplish specific tasks.

For example:

Software instructs the hardware what to display on the user's screen, what kinds of input to take from the user and what kinds of output to generate.

1.2.1 Types of software

Software can be categorised as system software and application software.

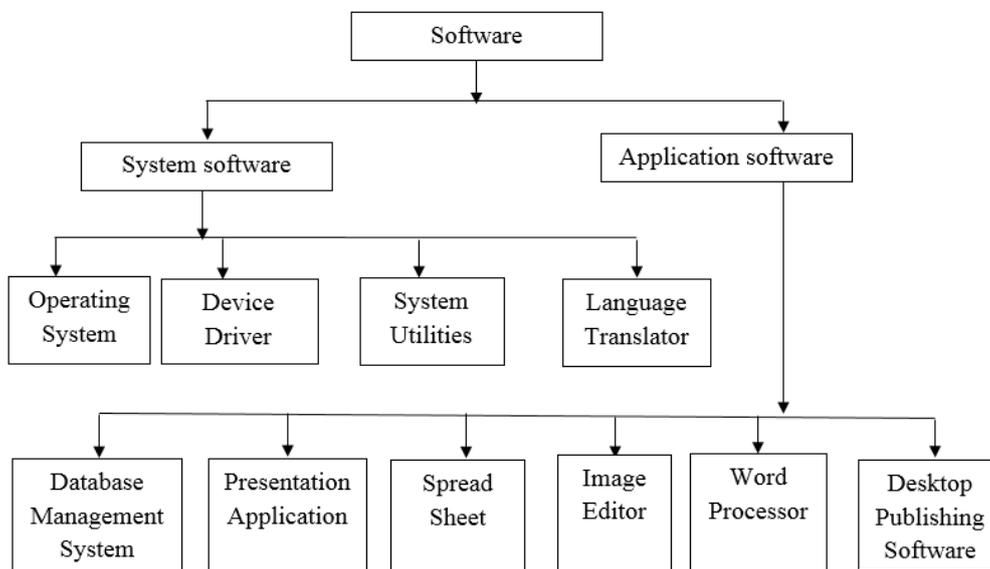


Fig 1.1 Types of software

1.2.2 SYSTEM SOFTWARE

System software as an interface system software consists of several programs which are directly responsible for controlling, integrating and managing the individual hardware components of a computer system.

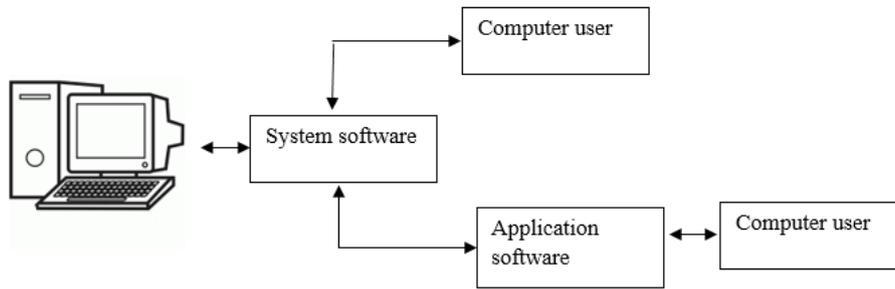


Fig 1.2: System software as an interface

This software provides a programming environment in which programmers can create applications to accommodate their needs. System software acts as an interface between the hardware of the computer and the software applications.

System software makes the computer functional; they provide basic functionality like file management, visual display, and keyboard input and are used by application software to accomplish these functions.

Some examples of system software are operating systems, device drivers, language translators, and system utilities.

Operating system

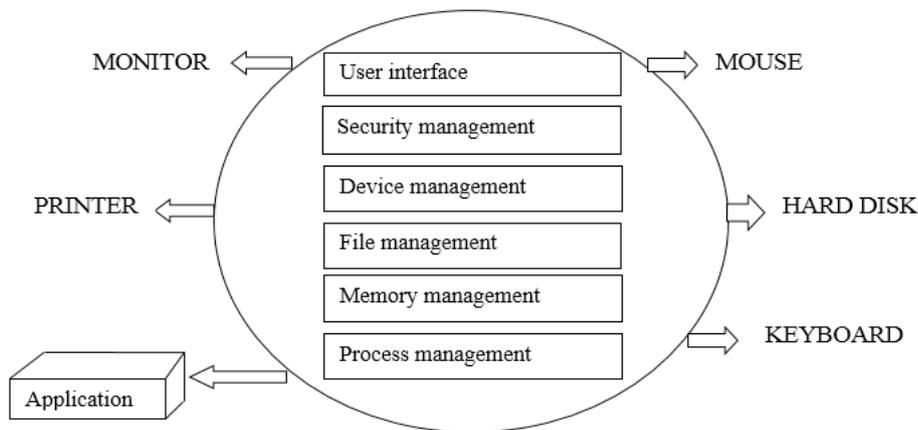


Fig 1.3: Operating system

Operating system is the first layer of software loaded into computer memory when it starts up.

As the first software layer, all other software that gets loaded after it depends on it for various common core services.

These common core services include disk access, memory management, task scheduling, and user interfacing.

The operating system organizes and controls the hardware.

Examples of operating systems are windows *XP*, *UNIX* and *LINUX*.

Device Drivers

Device drivers are system programs, which are responsible for proper functioning of devices every device, whether it is a printer, monitor, mouse or keyboard, has a driver associated with it per its proper functioning

In modern operating systems, most hardware drivers, such as the keyboard drivers, come with the operating system.

Language Translators

Computer only understand a language consisting of 0s and 1s called ***machine language***.

Depending on the programming language used language translators are divided into three major categories. Computer interpreter and assembles

Language Translators	Description
Compiler	The programs written in any high-level programming language (C or Pascal) are converted into machine language using a compiler.
Interpreter	An interpreter analyses and executes the source code in line-by-line Manner, without looking at the entire program.
Assembler	Compared to all the types of programming languages, assembly Language is closest to the machine code. An assembler converts the assembly codes into machine codes, making the assembly program ready for execution.

System Utility

System utility programs perform day to day tasks related to the maintenance of the computer system they are used to support enhance, and secure existing programs and data in the computer system.

1.2.3 APPLICATION SOFTWARE

Application software may consist of a single program, such as Microsoft notepad it may also consist of a collection of programs often called a ***software package*** which work together to accomplish a task, such as database management software.

Application software ranges from games, calculators and word processors document creating programs to programs that “paint” images on screen (image editors) some of the most commonly used application software are discussed below.

Word processors

A word processor is a software used to compose, format, edit and print electronic documents. It involves not only typing, but also checking the spelling and grammar of the text and arranging it correctly on a page.

1.24 Fundamentals of Computing an Programming

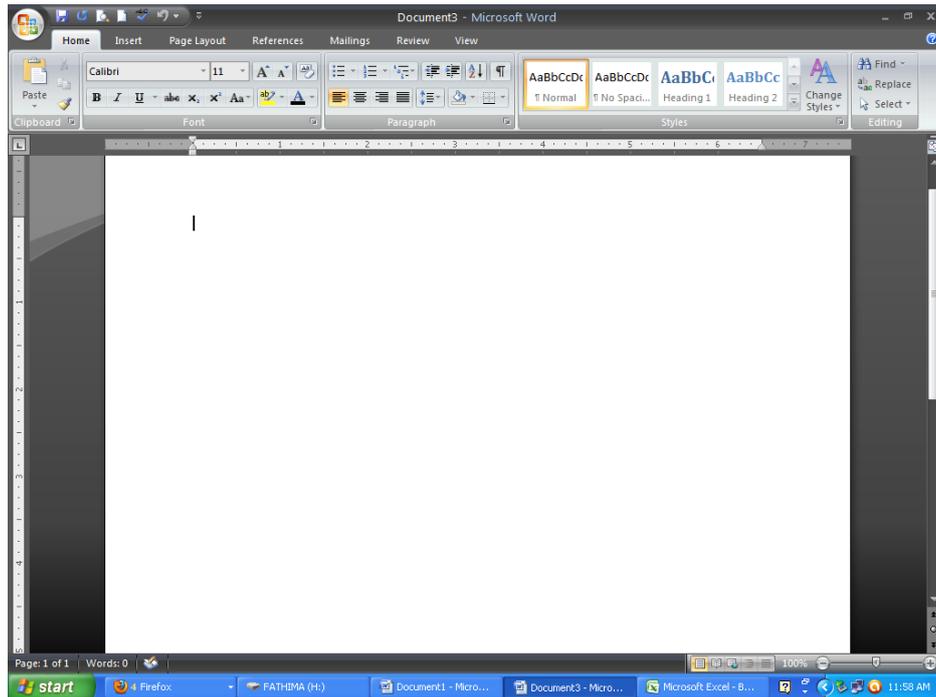


Fig 1.4: Microsoft word

It is possible to include pictures, graphs, charts and fonts and colour. Nowadays, virtually all personal computer are equipped with a word or other document and printing

Example of some well known word processors are Microsoft word and word perfect.

Spread sheets

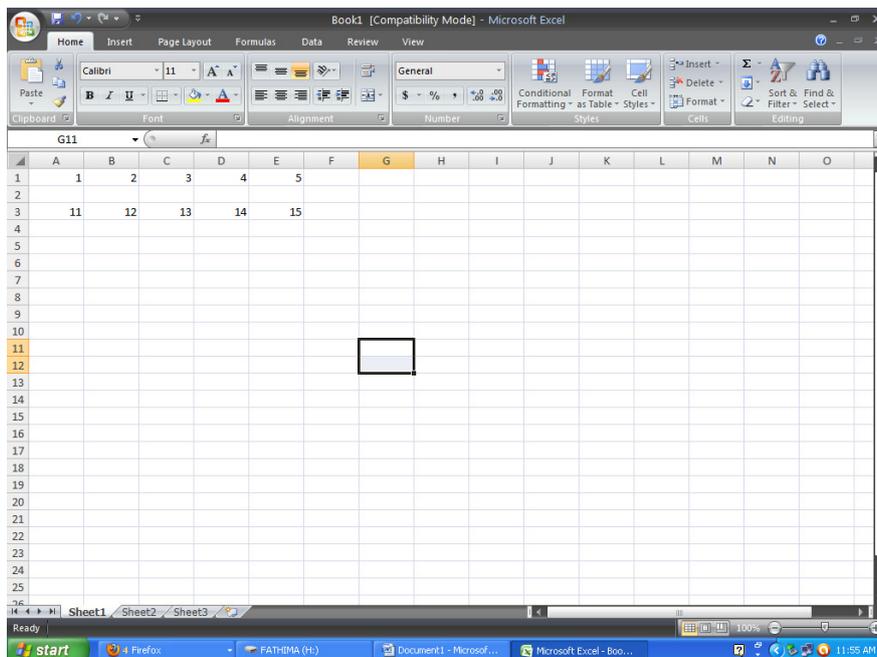


Fig 1.5: Microsoft excel

One of the first commercial uses of computers was in processing payroll and other financial records. A spreadsheet application is a rectangular grid, which allows text, number and complex functions to be entered into a matrix of thousand of individual cells.

The spreadsheet provides sheets containing cells each of which may contain text and/ or number

Cells may also contain equations that calculate results from data placed in other cells or series of cells.

Microsoft excel and lotus 1-2-3 are examples of spreadsheet applications

Image Editors

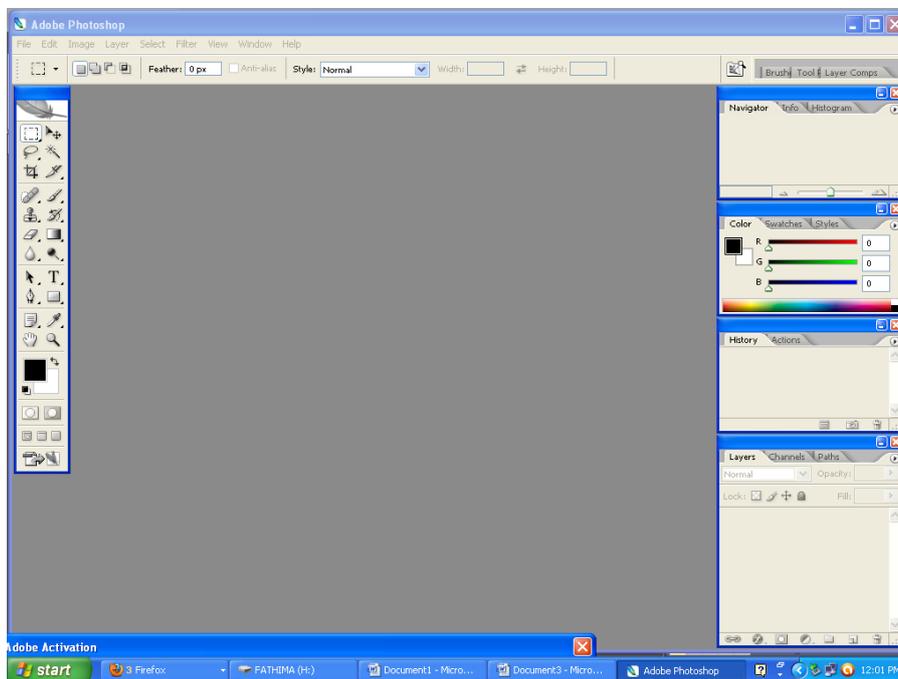


Fig 1.6: Adobe Photoshop

Image editor programs are designed specifically for capturing, creating, editing and manipulating images

These graphics programs provide a variety of special features per creating and altering images.

In addition to offering a host of filters and image transformation algorithms, some editors also enable the user to credit and superimpose layers.

With image editing software, one can darken or lighten an image, rotate it, adjust its contrast, crop out extraneous detail and much more.

Examples of these programs are adobe photoshop, adobe illustrator and corel draw.

Database management systems



Fig 1.7: Microsoft access

Database management software is a collection of computer programs that allow storage modification and extraction of information form a database in an efficient manner.

It provides tools for data input, verification storage, retrieval, query and manipulation.

New categories of data can be added to the database without disrupting the existing system.

Forpro and *oracle* are database management systems.

Presentation applications

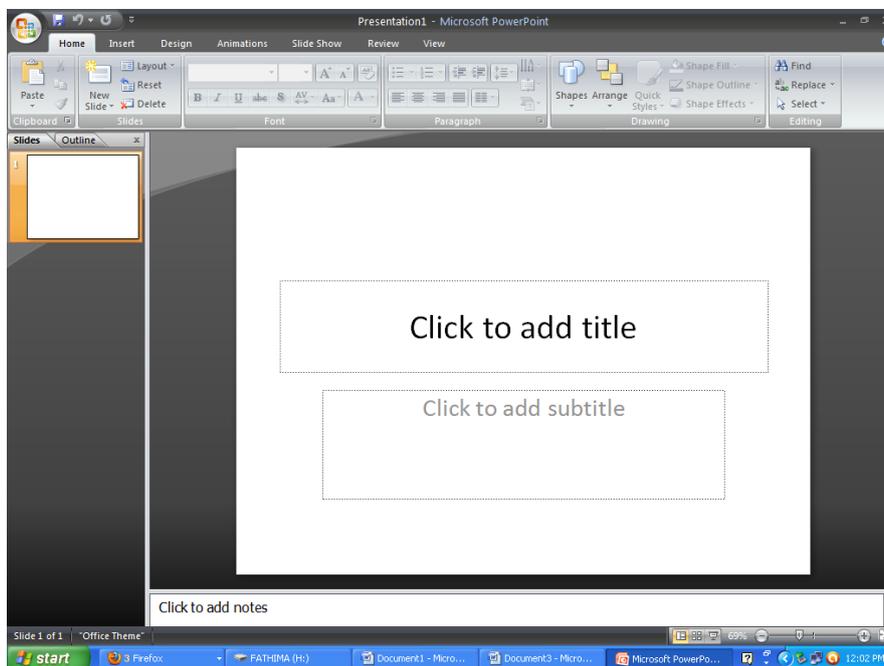


Fig 1.8: Microsoft PowerPoint

A presentation is a means of assessment which requires presentation providers to present their work orally in the presence of an quidience

It combines both visual and verbal elements presentation software allows the user to create presentations by producing slides or handouts per presentation of projects.

Microsoft powerpoint is one of the most famous presentation application.

Desktop publishing software:

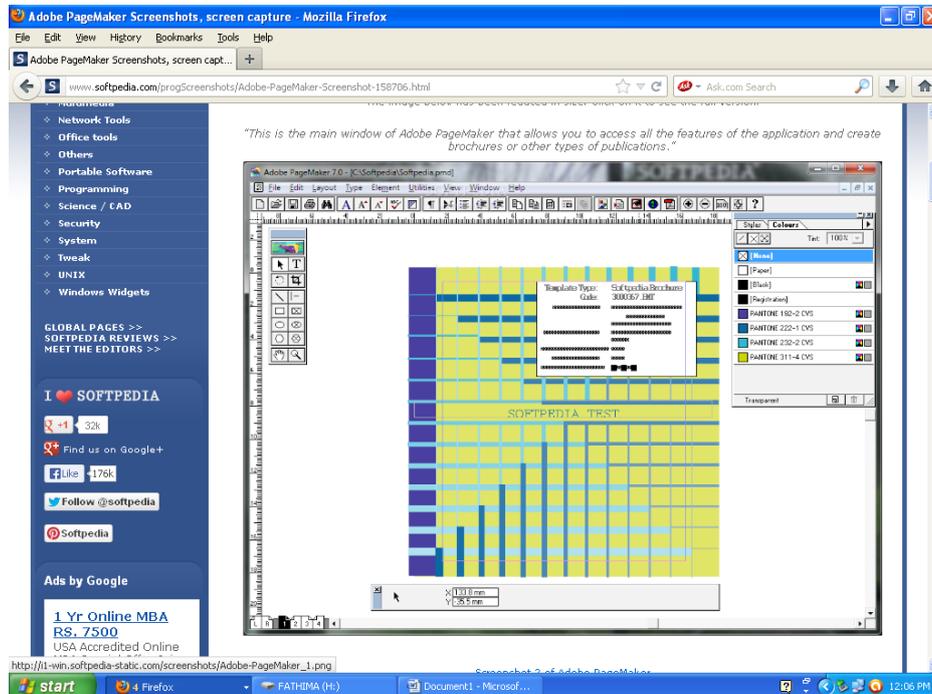


Fig 1.9: Adobe PageMaker

The term desktop publishing is usually used to describe the creation of printed documents using a desktop computer.

It is a technique of using a personal computer to design images and pages, and assemble type and graphics, then using a laser printer or image setter to output the assembled pages onto paper, film or printing plate.

Quark Express and *adobe page maker* are desktop publishing software.

1.2.4 SOFTWARE DEVELOPMENT STEPS

Software development is the set of activities that results in software produces. Software development may include research, new development, modification, reuse, re-engineering, maintenance or any other activities that result in software product

Software development life cycle (SDLC)

The various phases of software development involves the performance of many people.

(a) User

The user is a person who will use the software.

(b) System Analyst

A system analyst is the person who meets the users and gathers the information

(c) System Designer

System designer is the specialist who designs the system or software.

(d) Programmer

A programmer is a person who writes the code that implements the user requirements.

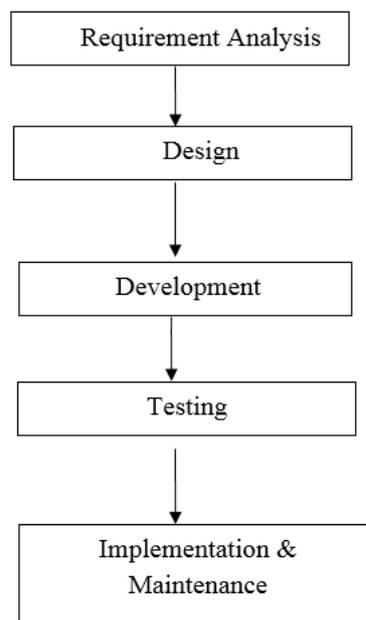
(e) Project manager

Project manager is a person who manages the entire team

(f) Testing team

Testing the software is done by this testing team. Bugs are identified and reported.

The following are the flow of software development life cycle.



Requirement – Analysis

The objective of the requirement analysis is to identify and document the user requirements for the proposed system.

The main objective of the requirements analysis is to produce a document that properly specifies all requirements of the customers. That is called the **Software Requirement Specifications (SRS)** document is the primary output of this phase.

This process involves analysis which includes meeting with interview and observing knowledge, users to understand what the requirements are, in addition existing system, processes, documents and procedures are also reviewed.

Design Process

It defines specifically how the software is to be written including an object model with properties and methods for each object, analysis and design are very crucial in the whole development cycle.

Development or Coding

The design must be translated into machine - readable form. The coding or development step performs this task.

Different high level languages like **C, C++, JAVA, COBOL** etc. are used for coding, with respect to the type of application, the right programming language is chosen.

Testing

Testing is the process of executing the proposed software with sample or test data and put it into regular use.

Once the code is generated, the program testing begins.

Different testing tools and methodologies are already own testing tools that are tailor made per their own development operations.

Implementation and Maintenance

It involves installation and initial training and may involve training and may involve hardware and network upgrades.

Software will definitely undergo changes once it is delivered to the customer.

In addition, the changes in the system could directly affect the software operations.

The software should be developed to accommodate changes that could happen during the post implementation period.

The maintenance phase of the project is the last component and it continues as long as a warranty, extended warranty or support contract is in place.

PROBLEM SOLVING AND OFFICE APPLICATION SOFTWARE

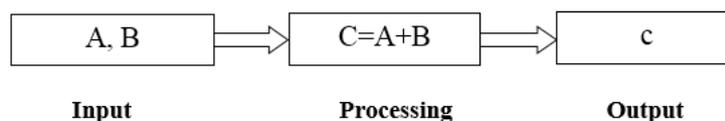
2.1 PLANNING THE COMPUTER PROGRAM

2.1.1 Program Definition

A program is a set of instruction written to carryout a particular task, so that computer can follow them

To solve the problem using the computers we must follow the steps below.

- (a) Problem must be analysed thoroghly,
- (b) Solution method is broken down into a sequence of small tasks.
- (c) Based on this analysis an algorithm must be prepared to solve the problem
- (d) The algorithm must be expressed in precise notation
- (e) In viewing algorithm, design a computer program in any high level language.
- (f) The computer program is fed into the computer.
- (g) The instruction in the program, executes one after another and outputs the expert result.



Calculate the sum of two numbers, A and B and store the sum in c, here a and b are the input addition is the process and c is the output of the program

2.2 PROBLEM SLOVING TECHNIQUES

2.2.1 Algorithms

An algorithm is defined as a finite sequence of explicit instructions that, when provided with a set of input values, produces an output and then terminates.

In algorithm, after a finite number of steps, solution of the problem is achieved.

Algorithms can have steps that repeat (iterate) or require decision (logic and comparison) until the task is completed.

2.2 Fundamentals of Computing an Programming

For example, to determine the largest number out of three numbers A,B and C the following algorithm may be used. Step 1: start

Step 1: Start

Step 2: Read three numbers say A, B, C

Step 3: Read the larger number between A and B and store it in MAX_AB.

Step 4: Find the larger number between MAX_AB, C and store it in MAX.

Step 5: Display MAX.

Step 6: Stop.

2.2.2 Algorithm properties

- (1) There must be no ambiguity in any instruction
- (2) There should not be any uncertainty about which instruction is to be executed next.
- (3) The algorithm should conclude after a finite number of steps an algorithm cannot be open ended
- (4) Then algorithm must be general enough to deal with any contingency.

2.3 FLOWCHART

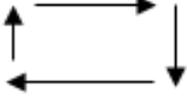
A flowchart is a pictorial representation of an algorithm in which the step are drawn in the form of different shapes of boxes and the logical flow is indicated by interconnecting arrow

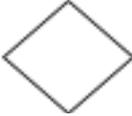
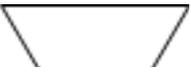
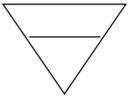
The boxes represent operations and the arrows represent the sequence in which the operation are implemented.

The primary purpose of the flowchart is to help the programmes in understanding the logic of the program.

Standard flowchart symbols prescribed by American nations standard institute (ANSI)

2.3.1 Flowchart symbols

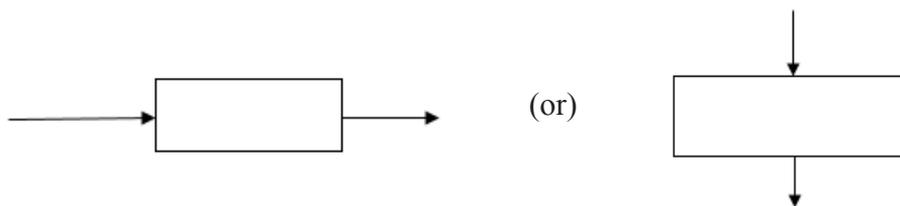
Symbol	Symbol name	Description
	Flow lines	Flow lines are used to connect symbols. These lines indicate the sequence of steps and the direction of flow of control.
	Terminal	This symbol is used to represent the beginning (star), the termination (end) or halt (pause) in the program logic.

	Input / Output	It represents information entering or leaving the system, such as customer order(input) and servicing(output)
	Processing	Process symbol is used for representing arithmetic and data movement instructions.
	Decision	Decision symbol denotes a decision (or branch). The program should continue along one the two routes (IF/ELSE). This symbol has one entry and two exit paths. The path chosen depends on whether answer to a question is yes or no.
	Connector	Connector symbol is used to join different flow lines.
	Off-page connector	This symbol is used to indicate that the flowchart continues on the next page.
	Document	Document is used to represent a paper document produced during the flowchart process.
	Annotation	It is used to provide additional information about another flowchart symbol. the content may be in the form of descriptive comments remarks or explanatory notes.
	Manual Input	Manual input symbol represents input to be given by a developer programmer
	Manual Input	It represents input to be given by a developer/ programmer
	Online storage	This symbol represents the online data storage such as hard disk, magnetic drums, or other storage.
	Offline storage	This symbol represents the offline data storage such as sales on OCR and data on punched cards
	Communication Link	Communication link symbol is used to represent data received or to be transmitted from an external system.

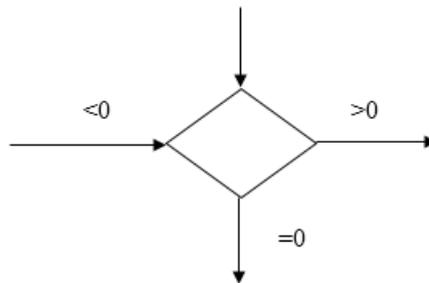
	Magnetic Disk	This symbol is used to represent data input or output from and to a magnetic disk.
---	---------------	--

2.3.2 Guidelines for preparing flowcharts

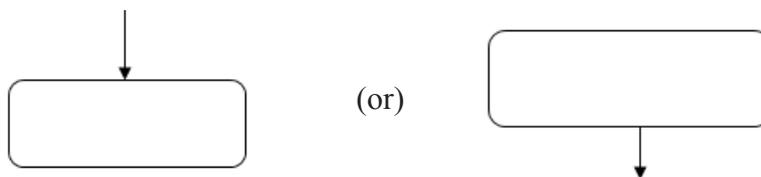
- The flowchart should be clear, neat and easy to follow
- The flowchart must have a logical start and finish
- In drawing a proper flowchart, all necessary requirements should be listed in logical order.
- Only one flow line should come out form a process symbol



- Only one flow line should enter a decision symbol. However, two or three flow lines may leave the decision symbol



- Only one flow line is used with a terminal symbol



- Within standard symbol write briefly if necessary use the annotation symbol to describe data or process more clearly



- In case of complex flowchart, connector symbols are used to reduce the number of flow lines

- Intersection of flow lines should be avoided to make it a more effective and better way of representing communication
- It is useful to test the validity of the flowchart with normal/ unusual test data.

2.3.3 Benefit of Flowchart

Makes Logic Clear

The main advantage of using a flowchart to plan a task is that it provided a pictorial representation of the basic which makes the logic easier to follow

Communication

Being a graphical representation of a problem – solving logic, flowcharts are better way of communicating the logic of a system to all concerned.

Effective Analysis

With the help of a flowchart, the problem, can be analysed in an effective way.

Useful in Coding

The flowcharts act as a guide or blueprint during the analysis and program development phase.

Proper Testing and Debugging

By nature, a flowchart helps in detecting the errors in a program, as the developers know exactly what the logic should do.

Appropriate Documentation

Flow charts serve as a good program documentation tool.

2.3.4 Limitations of Flowcharts

Complex

The major disadvantage in using flowcharts is that when a program is very large, the flowcharts may continue per many pages, making them hard to follow.

Costly

Drawing flowcharts are viable only if the problem solving logic is straight forward and not very lengthy.

Difficult to Modify

Due to its symbolic nature, any changes or modification to a flowchart usually requires redrawing the entire logic again, and redrawing a complex flowchart is not a simple basic.

No Update

Usually programs are updated regularly

2.4 PSEUDOCODE

Pseudocode is made up of two words *pseudo* and *code*. Pseudo means limitation and code refers to instructions, written in a programming language.

Pseudocode used plain English statements rather than symbols to represent the processes of a computer program. It is also known as **PDL** (Program Design Language)

For example, the pseudocode given below calculated the area of a rectangle.

```
PROMT the user to enter the height of the rectangle
PROMT the user to the enter the width of the rectangle
COMPUTE the area by multiplying the height with width
DISPLAY the area STOP
```

Pseudocode uses some keywords to denote programming processes. Some of them are:

- Input : READ, OBTAIN, GET and PROMPT
- Output: PRINT, DISPLAY AND SHOW
- Compute : XOMPUTE, CALCULATE AND DETERMINE
- Inittalise: SET AND INITIALISE
- Add One: INCREMENT

2.4.1 Pseudocode Guideline

- Statements should be written in simple English and should be programming language independent.
- Steps must be understandable and when the steps Are followed the must produce a solution to the specified problem
- Pseudocodes should be concise.
- Each instruction should be written in a separate line an each statement in pseudocode should express just one action per the computer.
- Capitalise keywords such as READ, PRINT and so on.
- Each set of instructions is written from top to bottom, with only one entry and one exit.
- It should allow for easy transition from design to coding in programming language

2.4.2 Benefits of Pseudocode

- Since it is language independent, it can be used by most programmes, it allows the developer to express the design in plain natural language.
- It is easier to develop a program from a pseudocode than with a flowchart
- Often, it is easy to translate pseudocode into a programming language a step which can be accomplished by less experienced programmes.

- The use of words and phrases in pseudocode, which are in line with basic computer operations, simplifier the translation from the pseudocode algorithm to a specific programming language.
- Its simple structure and readability makes it easier to modify.

2.4.3 Limitations of Pseudocode

- The main disadvantage of using pseudocode is that it does not provide visual representation of the program's logic
- Pseudocode can not be compiled nor executed and more are not real formatting or syntax rules

Problem Solving through Algorithms, Flowcharts and Pseudocode

1. Write a program to convert the celcius into fahrenheit.

Algorithm:

Step 1: Start

Step 2: Read the Celsius value

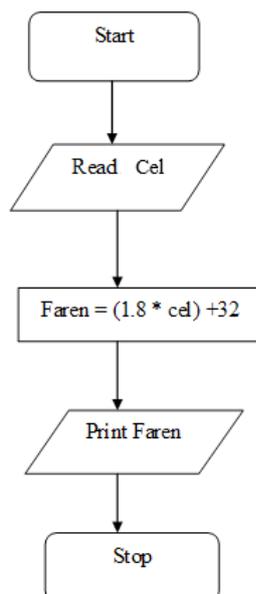
Step 3: Calculate the fahrenheit value by using the formula

$$\text{Fahrenheit} = (1.8 * \text{Celsius}) + 32$$

Step 4: Print the fahrenheit value

Step 5: Stop

Flowchart:

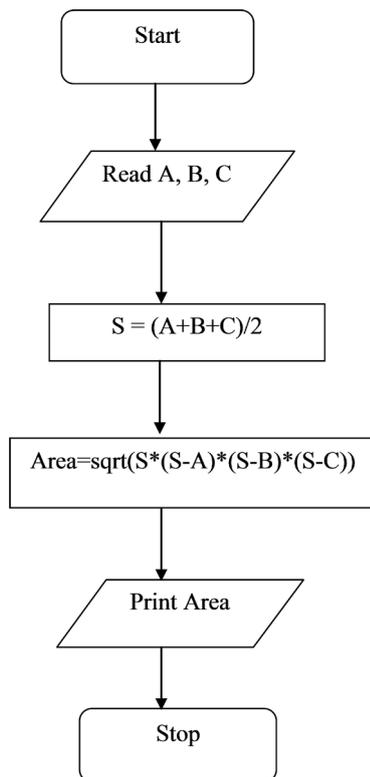


Pseudocode:

Set initial celsiue
READ the value of Celsius
Calculate faren = (1.8*Celsius) +32
Write the output Fahrenheit
Stop

2. Write a program to find area of the manage

Flowchart



Algorithm:

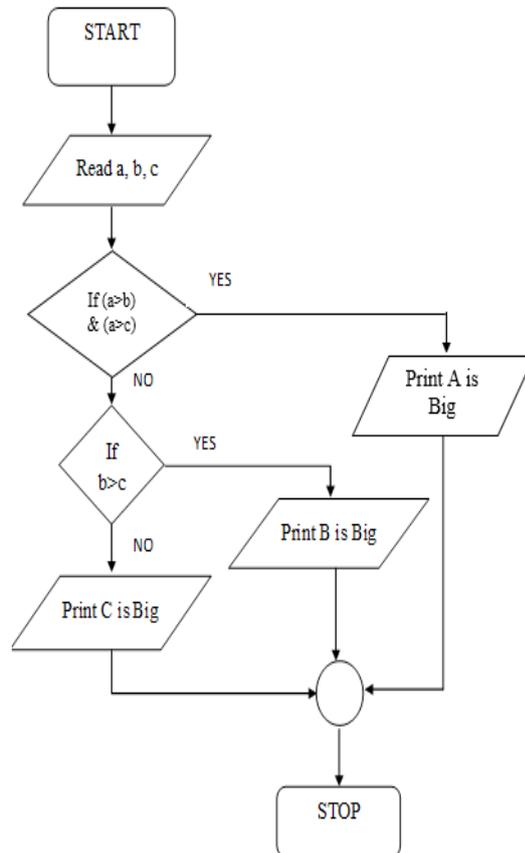
Step 1: Start
Step 2: Read the value of a,b,c
Step3: To calculate three sides of the triangle using formula
 $S \leftarrow (a+b+c)/2$
Step 4: To Find area of a triangle apply formula
 $\text{Area} \text{ sqrt } (s*(s - a)*(s-b)*(s-c))$
Step 5 : Print the area
Step 6: Stop

Pseudocode:

Set initial a,b,c
 Read the value of a,b,c
 To calculate three sides of triangle using formula
 $s = (a+b+c)/2$
 To find area of a triangle using formula
 $\text{Area} = \sqrt{s*(s-a)*(s-b)*(s-c)}$
 WRITE the output area
 Stop

3. Write a program to find the largest of three numbers.

Flowchart:



Algorithm:

Step1 : Start
 Step2 : READ a, b, c
 Step3 : IF (a>b) and (a>c) Then print 'A is Big'.

2.10 Fundamentals of Computing an Programming

Step 4 : Else, IF (b>c) Then print 'B is Big'.

Step 5 : Else print 'C is Big'

Step 6 : Stop.

Pseudo code:

Set initial a, b, c

READ the value for a, b, c

IF (a>b) and (a>c) THEN

 WRITE 'A is Big'

Else, IF (b>c) THEN

 WRITE 'B is Big'

ELSE

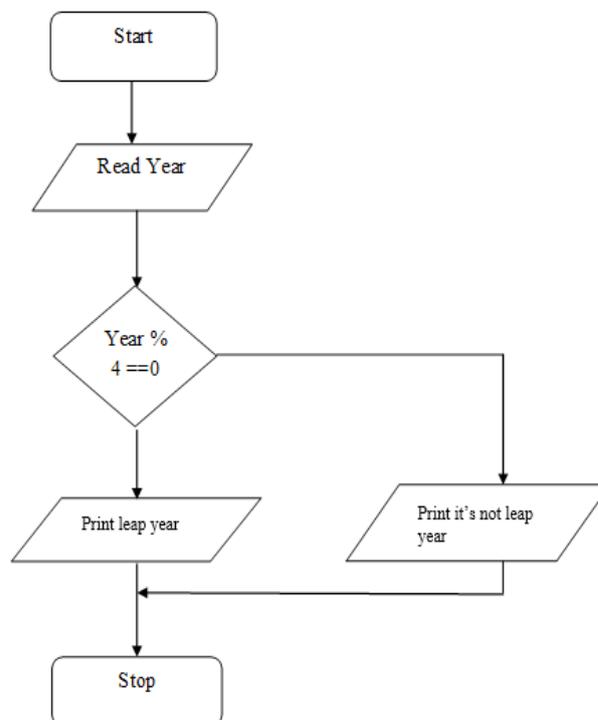
 WRITE 'C is Big'

END IF

Stop.

4. Write a program to find the given year is a leap year or not.

Flowchart



Algorithm:

Step 1 : Start.
 Step 2 : Read the value of year.
 Step 3 : if ((year mod 4) = 0) then print "It is a Leap Year".
 ELSE print "it is not Leap Year"
 Step 4 : Stop.

Pseudo code:

Set initial value of year.
 READ the value of year.
 IF (year % 4 == 0) Then
 WRITE the year is leap year
 ELSE
 WRITE the year is not leap year.
 ENDIF
 STOP

5. Write a program to find the roots of the quadratic equation.

Algorithm:

Step 1 : Start.
 Step 2 : Read the value of A, B, C.
 Step 3 : Find the value of 'D' of using the formula $D=B^2-4*A*C$
 Step 4 : If D is greater than or equal to zero then find two roots are
 Root1 $(-B+ \text{sqrt}(D)) / (2*a)$
 Root2 $(-B- \text{sqrt}(D)) / (2*a)$
 Step 5 : Print two roots Root1, Root2
 Step 6 : If 'D' is not greater than or equal to zero, then print the roots are Imaginary
 Step7 : Stop.

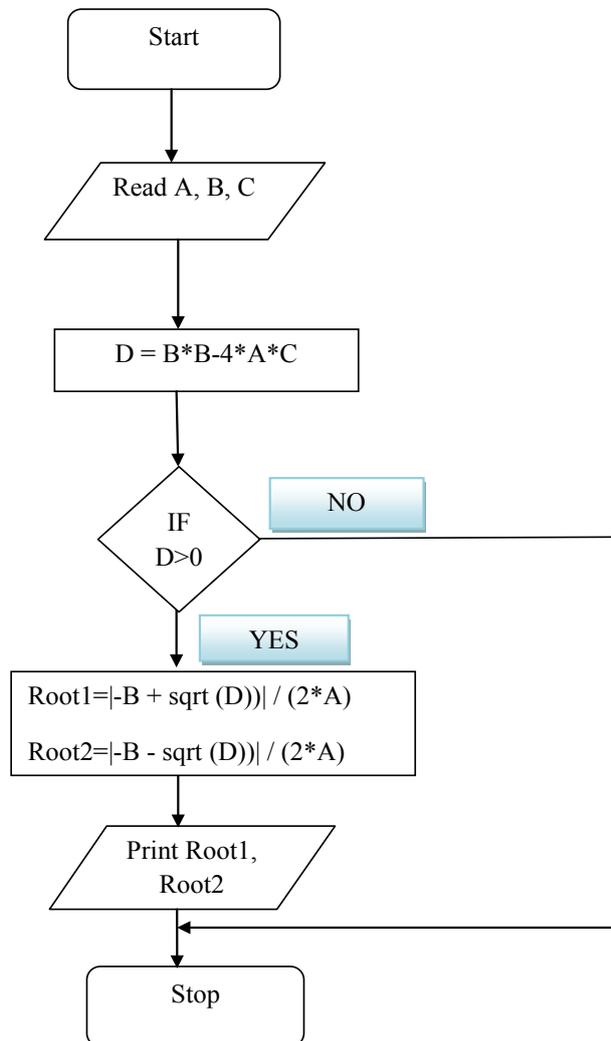
Pseudo code:

Set initial zero to D, Root1 and Root2.
 READ the value of A, B, C.
 Find Discriminate
 $D=B^2-4*A*C$
 IF $D \geq 0$ THEN

```

Calculate Root1  (-B+ sqrt (D)) / (2*a)
                Root2  (-B- sqrt (D)) / (2*a)
ELSE
Roots are imaginary
ENDIF
WRITE Root1, Root2.
STOP
    
```

Flowchart:



3.5 INTRODUCTION TO OFFICE PACKAGES

3.5.1 Word Processing

Word processing in the computer is the process of creating, editing, retrieving, storing and printing test material.

Features of word processes.

- (a) fast
- (b) editing features
- (c) permanent storage
- (d) formatting features
- (e) graphics
- (f) OLE object linking and embedding
- (g) Spell check
- (h) Tables
- (i) Mail merge

Introduction to MS –word

MS - word is a word processing program that lets us to create documents such as letter, reports, manuals and newsletters etc.

The option provided by ms-word are explained below

Working with document

(i) Entering Text

Type the text in the document press enter key, when we want to start new paragraph. Word underlines misspelled words in red and grammar mistakes in green, the red and green underlines will not appear in printing the document.

Word automatically corrects common typing mistakes that we type such as and (adn) and new (new)

(ii) Saving a document

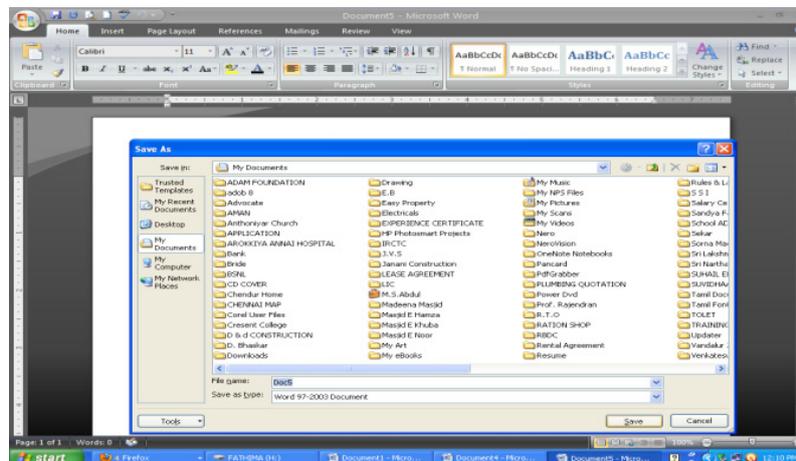


Fig 2.1. Saving a document

2.14 Fundamentals of Computing an Programming

- (1) Click the **Save** button on the **Standard Toolbar** the **save as Dialog Box** appears as shown fig
- (2) Type the name of the file in the **filename** text box
- (3) Select the appropriate disk drive and folder in the **save in** list box
- (4) Click the **save** button.

(iii) Opening a document:

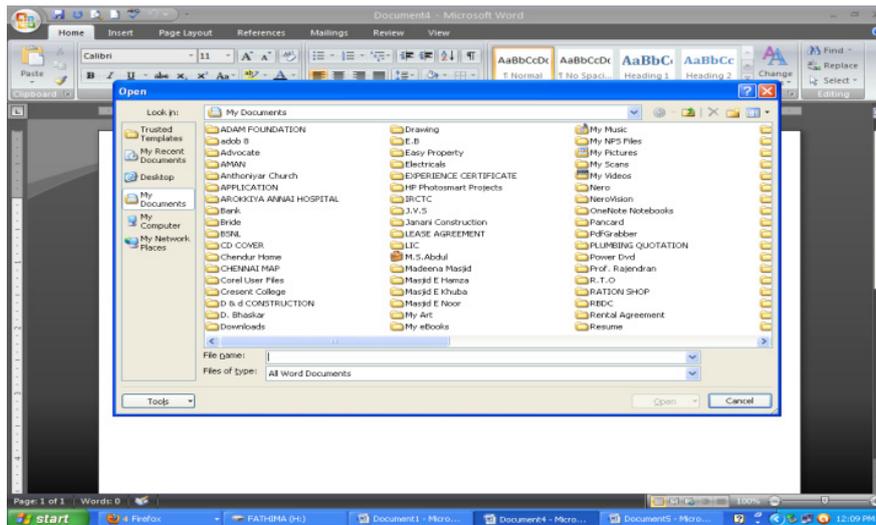


Fig 2.2. Opening a document

- (1) Click on the **open** button on the **standard toolbar**. The **open dialog box** appear as shown fig
- (2) Select the appropriate disk drive and folder where your file exists in the **Look in** : list box
- (3) Type the name of the file or click on the file name from the list which we want to open in the **file name** text box.
- (4) Click the **open** button now the specified file is opened.

(iv) Copying text

- (1) Select the text, which we want to copy
- (2) Click on **copy** button on the formatting tool bar or choose **copy** option form the **Edit** menu or press **CTRL+C**
- (3) Now the selected text disappears and it immediately places in the clip board
- (4) Place the cursor at desired location i.e, Where we want to insert the text.
- (5) Click on the **paste** butto in the formatting tool bar or click on the **paste** option from the **Edit** menu or press **CTRL+V**
- (6) Now, the text will appears at the cursor position

2.5.2 Finding and Replacing Text

(i) Finding text

Suppose we want to search the word “sales”. Choose the **Find and Replace** option from the **Edit** menu or press **CTRL+F**.

The find and replace dialog box appears.

- (1) Type the word **sales** in the **find what** text box.
- (2) Click the **find whole words only** check box
- (3) Click **find next** button
- (4) Repeat the process until confirmation dialog box opens informing we that word has finished searching the document
- (5) Click **ok**

(ii) Replacing text

Suppose we want to replace the word "**Invoice**" with the word "**sales**" in the document do the following

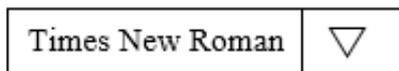
- (1) Click **Replace** tab in the **Find and Replace** dialog box.
- (2) Type the word **Invoice** in the **Find what** text box
- (3) Type the word **Sales** in the **Replace with** text box
- (4) Click **Replace** push button it replaces all the occurances of "**Invoice**" with "**Sales**"

2.5.3 Formatting Documents

(1) Changing font types and font size

Using the **font dialog box** we can change font style, font size, underlying blod italic styles and color of selected text

Font Tab: Click the **font** tab to view the different **fonts**. We can also select a font by clicking the list arrow in the **font type** box on the following toolbar.



(2) Applying superscript and subscript format

The **superscript** format places text slightly above a line of normal printed text. The **subscript** format places text slightly below a line of normal printed text.

Eg. Suppose if we want to type A^2+B^2

- (1) Type the formulae as A^2+B^2
- (2) Select the first 2 in A^2+B^2 then choose **font** option from the **format** menu then choose **superscript** check box

(3) Now we can get A^2+B^2 like $A^2 + B^2$

(4) This is ^{superscript} and this is _{subscript}

(3) Bullets and numbering

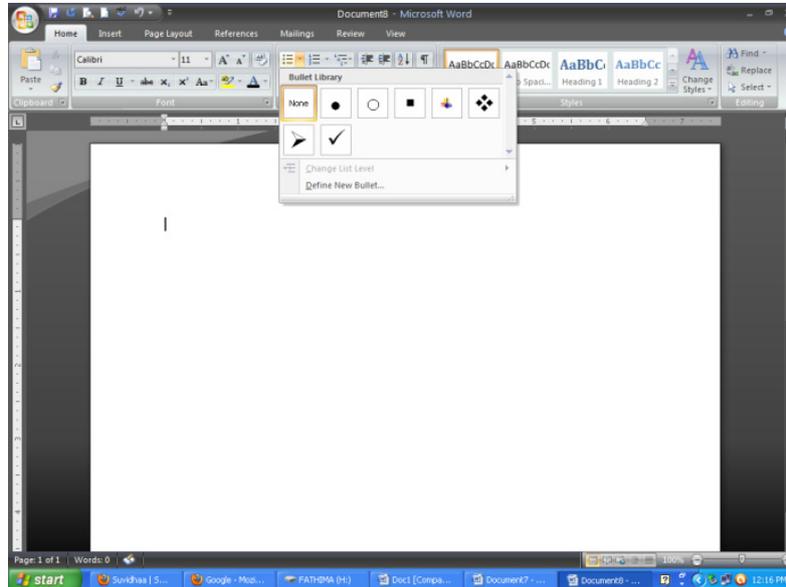


Fig 2.3. Bullets and numbering

To add **numbering** to a selected text, do the following

- (1) Select the text from beginning line to ending line.
- (2) Choose **Bullets** and **Numbering** option from the format menu, now the **Bullets and Numbering dialog box** appears as shown in figure below.
- (3) Choose the required format for Numbering, then click OK.

To add Bullets, do the following

- (1) Select the **paragraph**.
- (2) Click the **Bullets** button on the formatting tool bar, choose required bullets and then press **OK** button. Bullets are now inserted in the paragraph.

(4) Inserting Symbols.

- (1) Select **symbol** option from the insert menu.
- (2) The symbols menu appears as shown figure.
- (3) There are so many symbols available to from the different types of fonts to view all the symbols click on the **font** list box.
- (4) Then choose phone symbol.
- (5) Then click on **insert** button and then click **cancel** button.

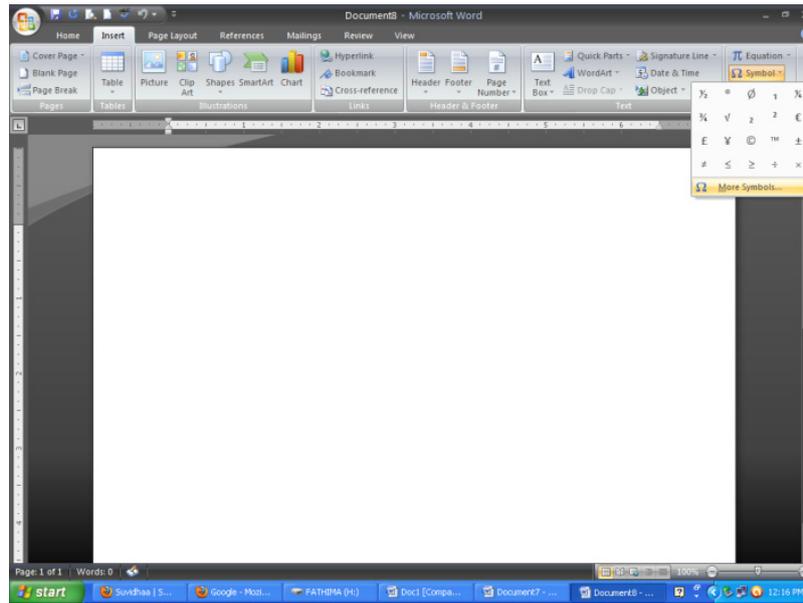


Fig 2.4. Inserting symbols

2.5.4 Working with tables

A table is simply information arranged in rows and column

- Insert table button
- Insert table dialog box (*Table* → *Insert* → *Table menu selecting*)
- Tables and border button
- *Table* → *Convert Text to Table* menu selection

(1) *Using the Insert Table Button*

To use the insert table button, drag the desired number of rows and columns and release the button.

INTRODUCTION TO C

3.1 INTRODUCTION

C is a popular general purpose programming language. C language has been designed and developed by **Dennis Ritchie** at bell laboratories in **1972**.

The source code for the linux operating system is coded in C. C runs under a number of operating systems including MS – DOS

C language is a middle - level computer language. It reduces the gap between high level language and low – level language it is known as **middle level language**.

C is a structured language. It is simily in many ways to otha structural languages such as pascal and fortran. A structured language allows variety of programs in small modules.

C provides loop constructs like **while**, **do-while** and **for**.

3.1.1 Structure of a C program

```
Include header file section
Global declaration section
/* comments */
Main() /* Function name */
{
/* comments */
Declaration part
Executable part
}
User-defined functions
{
Statements
}
```

(a) Include header file section

C program depends upon some header files for function definition that are used in program. Each header file by default is extended with **.h** the header file should be included using **# include** directive as given here.

```
For example  # include <stdio.h> or
              # include "stdio.h"
```

In this example **<stdio.h>** file is included, that is, all the definitions and prototypes of function defined in this file are available in the current program.

(b) Global declaration

This section declares some variables that are used in more than one function. Those variables are known as global variable.

(c) Function main

Every program written in c language must contain main c function. Empty parathensis after main are necessary. The function mains is a starting point of every c program. The execution of the program always begins with are function mains.

Except the main c) function, other sections may not be necessary. The program execution starts form the opening ({) and ends with the closing brace (}) between these two braces the program should declare the declaration and executable part.

(d) Declaration part

The declaration part declares the entire variables that are used in executable part initialisation means providing initial value to the variable.

(e) Executable part

This part contains the statements following the declaration of the variable this part contains a set of statements or a single statement. These statements are enclosed between the braces.

(f) User – defined function

The functions defined by the user are called user defined functions these functions are generally defined after the main () function. They can also be defined before main() function this portion is not compulsory.

(g) Comments

Comments are not necessary in the program comments are to be inserted by the programmer.

Comments are nothing but same kind of statement which are placed between the delimiters logo. The compiler does not execute comments.

For example:

```
/* this is single comment */
```

3.1.2 Programming Rules

- (1) All statements should be written in lower case letter. Upper case letter are only used for symbolic constants.
- (2) Blank spaces may be inserted between the words.
- (3) The user can also write one or more statement in one line separating them with a semi-colon (;)

```
a=b + c;
```

```
d=b * c;
```

or

```
a=b + c; d=b*c;
```

3.1.3 Executing the program

(a) Creation of program

Programs should be written in editor.

(b) Compilation and linking of a program.

The source program statements should be translated into object programs which is suitable for execution by the computer. The translation is done. After correcting each statement if there is no error, compilation proceeds and translated program are stored in another file, with the same. File name with extension". Obj"

If any errors. Are there the programmer should correct them. Linking is also an essential process. It puts all other program files and function together that are required by the program.

(c) Executing the program:

After the compilation the executable object code will be loaded in the computer's main memory and the program is executed.

3.2 THE C CHARACTER SET

The character used to form words, numbers and expressions depend upon the computer in which the program runs.

- (1) Letter
- (2) Digits
- (3) White spaces
- (4) Special characters

Character set

Letters	Digits	White space
Capital A to Z	All decimal digits 0 to 9	Blank Space
Small a to z		Horizontal Tab
		Vertical Tab
		New Line
		Form Feed

List of Special Characters

Special Character	Character Name		Special Character	Character Name
,	Comma		+	Plus
.	Period or dot		<	Less than
;	Semicolon		>	Greater than
:	Colon		()	Parenthesis left/right
`	Apostrophe		[]	Bracket left/right
“	Quotation mark		{}	Braces left/right
!	Exclamation mark		~	Tilde
	Vertical bar		_	Under score
/	Slash		\$	Dollar
\	Back slash		?	Question mark
&	Ampersand		%	Percent
^	Caret		#	Number sign or hash
*	Asterisk		=	Equal to
-	Minus		@	At the rate

Delimiters

Delimiters	Use
: (Colon)	Useful for label
; (Semicolon)	Terminates statement
() (parenthesis)	Used in expression and function
[] (Square Bracket)	Used for array declaration
{ } (Curly brace)	Scope of statement
# (Hash)	Preprocessor directive
, (Comma)	Variable separator

The C keywords

The c keywords are reserved words by the compiler. All the c keywords have been assigned a fixed meaning.

C keywords.

Auto	Double	int	struct
break	Else	long	switch
Case	Enum	register	typedef
Char	Sxtern	return	Union
Const	Float	short	unsigned
Continue	For	signed	Void
Default	Goto	sizeof	volatile
Do	If	static	While

Additional keywords for Borland c are as follows

Asm	Cdecl	Far	Huge	interrupt	Near	pascal
-----	-------	-----	------	-----------	------	--------

Identifiers

Identifiers are names of variables, function and arrays. They are user-defined names consisting sequence of letters and digits, with the letter as the first character. The (-) underscore symbol can be sued as an identifier.

For example:

- (a) # define n 10 here 'n' and 'a' are
- (b) # define a15 user – defined identifiers

3.3 CONSTANTS

The constants I care applicable to the values which do not charge during the execution of a program.

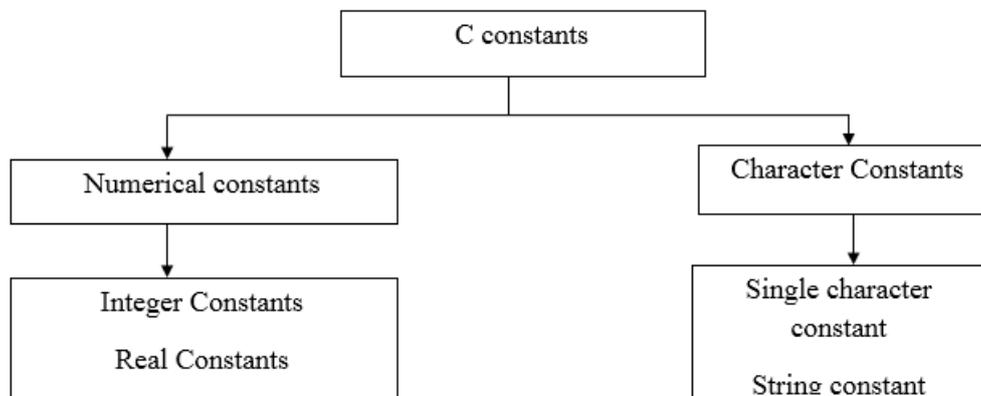


Fig 3.1. C constants

3.3.1 Numerical Constants

(a) Integer constants

These are the sequence of numbers from 0 to 9 without decimal points or fractional part or any other symbols. It requires minimum two bytes and maximum four bytes integer constants could either be positive or negative or may be zero.

For example: 10, 20, +30, -15, etc.

(b) Real constants

Real constants are often known as floating point constants. For example, length, height, prize, distance and so on are measured in real numbers.

For example: 2.5, 5.321, 3.14 etc.

3.3.2 Character Constant

(a) Single character constants:

A character constant is a single character. Characters are also represented with a single digit or a single special symbol or white space enclosed within a pair of single quote marks.

For example: 'a', '8', ' ', etc.

(b) String constants

String constants are sequence of characters enclosed within double quote marks. The string may be a combination of all kinds of symbols

For example: "Hello", "India", "444", "a".

(c) Variables

Variables can be different data types the data types are integer real or character, constants. In C, a variable is a data name used for storing a data value its value may be changed during the program execution.

A variable name may be declared based on the meaning of the operation same meaningful variable names are as follows.

For example: height, average, sum etc.

3.4 DATA TYPES

Data is represented using numbers or characters. The numbers may be integers or real. C completely supports different data types.

3.4.1. Integers data types

(1) Integer, short and long

All C compilers offer either short or long integer data types.

- *Difference between short and long integers.*

Short Integer	Long Integer
Occupies 2 bytes in memory	Occupies 4 bytes in memory
Range: -32,768 to 32,767	Range : -2147483648 to 2147483647
Program runs faster	Program runs slower
Format specifies is %d or %i	Format specifies is %ld
Example : int a=2; Short int b=2;	Example: long b; Long int c;

(2) Integers, signed and unsigned integers

- *Difference between signed and unsigned integers*

Signed Integer	Unsigned Integer
Occupies 2 bytes in memory	Occupies 2 bytes in memory
Range: -32,768 to 32,767	Range : -0 to 65535
Format specifies is %d or %i	Format specifies is %u
Example : int a=2; long int b=2;	Example: unsigned long b; Unsigned short int c;

(3) Character, signed and unsigned

Signed Integer	Unsigned Integer
Occupies 1 bytes in memory	Occupies 1 bytes in memory
Range: -128 to 127	Range : 0 to 255
Format specifies is %c	Format specifies is %c
Example : Char ch='b';	Example: Unsigned char='b';

(4) Floats and Doubles

Float	Double float
Occupies 4 bytes in memory	Occupies 8 bytes in memory
Range: -32,768 to 32,767	Range : 1.7e-308 to 1.7e+308
Format specifies is %f	Format specifies is %lf

Example : Float a	Example: Double y; Long double k;
----------------------	---

(5) Enter Data types in C

Data Types	Size (Bytes)	Range	Format specifiers
Char	1	-128 to 127	%c
Unsigned char	1	0 to 255	%c
Short or int	2	-32,768 to 32,767	%i or %d
Unsigned int	2	0 to 65535	%u
Long	4	-2147483648 to 2147483647	%ld
Unsigned long	4	0 to 4294967295	%u
Float	4	3.4e-38 to +3.4e+38	%f or %g
Double	8	1.7e-308 to 1.7e+308	%lf
Long double	10	3.4e-4932 to 1.1e+4932	%lf

3.4.2 Declaring variables

Declaration provides two things, (1) compiler obtains the variable name, (2) it tells the compiler the data type of the variable being declared and helps in allocating the memory.

Syntax:

Data type variable – name

For example:

```
int age;
char m;
float s;
double k;
int a,b,c;
```

The int, char, float and double are keywords to represent data types.

Data Types	Keywords
Character	char
Signed character	signed char
Unsigned character	unsigned char
Integer	int
Signed integer	signed int
Unsigned integer	unsigned int

Unsigned short integer	unsigned short int
Signed long integer	signed long int
Unsigned long integer	unsigned long int
Floating point	float
Double floating point	double
Extended double floating point	long double

3.4.3 Initializing variable

Variable declared can be assigned or initialized using an assignment operator

Syntax:

Variable – name – constant

Or

Data-type variable – name – constant

For example

X = 2, where x is an integer variable

Int y = 2;

Int x,y,z

3.4.4 Constant and volatile variables

(a) Constant variable

The keyword const is then added before the declaration it tells the compiler that the variable is a constant

For Example:

Const int m=10;

Where, const is keyword, m is a variable name and 10 is a constant value. Where, const is keyword, m is a variable name and 10 is a constant value.

(b) Volatile variable:

The volatile variables are those variables

That are changed at any time by other external program or the same program.

Syntax:

Volatile int d:

It the value of a variable in the current program is to be maintained constant and desired not to be changed by any other external operation, than the declaration of variable will be as follows

Volatile const d = 10

3.5 OPERATORS AND EXPRESSION

Priority of operators and their clubbing

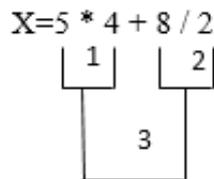
Operators	Operation	Clubbing	Priority
() [] .	Function call Square bracket Structure operator Structure operator	Left to Right	1st
+ - ++ -- ! * & sizeof <type>	Unary plus Unary minus Increment Decrement Not operator One's complement Address operator Size of an object Type cast	Right to Left	2nd
* / %	Multiplication Division Modular division	Left to Right	3rd
+ -	Addition Subtraction	Left to Right	4th
<< >>	Left shift Right shift	Left to Right	5th
< <= > >=	Less than Less than or equal to Greater than Greater than or equal to	Left to Right	6th
== !=	Equality Inequality	Left to right	7th
&	Bitwise and	Left to right	8th
^	Bitwise xor	Left to right	9th
	Bitwise or	Left to right	10th
&&	Logical And	Left to right	11th
	Logical or	Left to right	12th

?:	Conditional operator	Right to left	13th
=, *=, -, =, +=, ^=, \=, <<=, >>=	Assignment operator	Right to left	14th
,	Comma operator	Left to right	15th

- (1) When two operators of the same priority are found in the expression, precedence is given to the extreme left operator.

For example:

$$X = 5 * 4 + 8 / 2$$

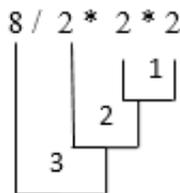


Here $5 * 4$ is solved first. Though $*$ and $,$ have same priorities. The operator $*$ occurs before

- (2) If there are more sets of parenthesis in the expression, the innermost parenthesis will be solved first, followed by the second and soon

For example:

$$X = (8 / (2 * (2 * 2)));$$



Here,

- Inner most bracket is evaluated first, i.e. $2 * 2 = 4$
- Second inner most brackets are evaluated. 2 are multiplied with result of inner most brackets. The answer of $2 * 4 = 8$.
- Then the outer most is evaluated 8 is divided by 8 and gives result 1.

3.5.1 Operators

An operator is a symbol that specifies an operation to be performed on the operands

- Two operands called binary operators
- One operand called unary operator.

For example:

$$A + B, \text{ where '+' is a operator and 'A', 'B' are the operand}$$

Types of Operators

- (a) Arithmetic operators
- (b) Relational operators
- (c) Logical operators
- (d) Assignment operators
- (e) Increment and decrement operator
- (f) Conditional operators
- (g) Bitwise operators
- (h) Special operator

3.5.1.1 Arithmetic Operator

Operator	Meaning	Example
+	Addition	2+9=11
-	Subtraction	9-2=7
*	Multiplication	2*9=18
/	Division	9/3=3
%	Modulo division	9%2=1

All the above operator are called binary operator as they acts upon two operands at a time

The following table shows how the division operator operates an various data types

Operation	Result	Example
int/int	Int	2/5=0
real/int	Real	5.0/2=2.5
int/real	Real	5/2.0=2.5
real/real	Real	5.0/2.0=2.5

Arithmetic operator can be classified as

- Unary arithmetic operator
- Binary arithmetic operator
- Integer arithmetic operator

(a) Unary Operator:

It requires only one operand.

Example: +x, -y

(b) Binary Arithmetic:

It requires two operands.

Example: a+b, a-b, a*b, a/b, a%b.

(c) Integer arithmetic:

It requires both operands are integer values for arithmetic operation

Example: A=5, B=4

Expression	Result
A + B	9
A - B	1

(d) Floating point arithmetic:

It requires both operands are float type for arithmetic operation

Example: A=6.5, B=3.5

Expression	Result
A + B	10.0
A - B	3.0

Example

Program to illustrate the usage of arithmetic operator.

```
#include<stdio.h>
#include<conio.h>
Void main ()
{
    int i,j, k ;
    i=10;
    j=20
    k=i+j;
    printf ("value of k is %d\n",k);
    getch();
}
```

Output: value of k is 30.

3.5.1.2 Relational Operators

Syntax: AE, relational operator AE2

Description: AE, &AE2 are constants or an expression variables.

Example: a>2 yields true, which is equal to 1.

Operators	Meaning	Example	Return value
<	is less than	2<9	1
<=	is less than or equal to	2<=2	1
>	is greater than	2>9	0
>=	is greater than or equal to	3>=2	1
==	is equal to	2==3	0
!=	is not equal to	2!=2	0

Expression	Interpretation	Value
a<b	True	1
(a+b)>=c	True	1
(b+c)>(a+5)	False	0
c!=3	False	0
b==2	True	1

The value of relational expression is either one or zero.

Example:

Program to use various relational operators and display their return value.

```

/*Program to use various relational operators*/
#include<stdio.h>
#include<conio.h>
Void main ()
{
/*Statements*/
clrscr();
printf("\n condition : Return value \n");
printf("\n 5!=5 : %5d", 5!=5);
printf("\n 5==5 : %5d", 5==5);
printf("\n 5>=5 : %5d", 5>=5);
printf("\n 5<=50 : %5d", 5<=50);
printf("\n 5!=3 : %5d", 5!=3);
getch();
}

```

Output:

Condition	:	Return value
5!=5.	:	0
5==5	:	1
5>=5	:	1
5<=50	:	1
5!=3	:	1

3.5.1.3 Logical Operators

Operator	Meaning	Example	Return value
&&	Logical AND	(9>2)&&(17>2)	1
	Logical OR	2<=2	1
!	Logical NOT	2>9	0

Logical operators are used to combine the results of two or more conditions.

Program on demonstrate logical operator.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int c1,c2,c3;
clrscr();
printf("Enter the values of c1, c2 and c3:");
scanf("%d%d%d", &c1,&c2,&c3);
if((c1<c2)&&(c1<c3))
printf("\n c1 is less than c2 and c3");
if(!(c1<c2))
printf("\n c1 is greater than c2");
if((c1<c2)||c1<c3)
printf("\n c1 is less than c2 or c3 or both");
getch();
}
```

Output:

```
Enter values of c1, c2 and c3: 45 32 98
c1 is greater than c2
c1 is less than c2 or c3 or both
```

3.5.1.4 Assignment operator

Assignment operator are used to assign a value or an expression or a value of a variable to another variable.

Syntax : variable=expression (or) value.

Example : x=10;
 x=a+b;
 x=y;

Program to demonstrate assignment operator.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int i,j,k;
clrscr();
k=(i=4,j=5);
printf("\n k is %d",k);
getch();
}
```

Output:

K=5

3.5.1.5 Increment and Decrement Operator

Operators	Meaning
++x	Pre increment
--x	Pre decrement
x++	Post increment
x--	Post decrement

‘++’ adds one to the variables.

‘—’ subtract one from the variables.

Program using increment and decrement.

```
#include<stdio.h>
#include<conio.h>
void main()
```

```

{
int i,j,k;
clrscr();
i=3;
j=4;
k=i++ + --j;
printf("\n i=%d, j=%d, k=%d",i,j,k);
getch();
}

```

Output:

i=4, j=3, k=6

3.5.1.6 Conditional operator (or) ternary operator syntax:

Syntax : Condition? exp1:exp2;

Description : The ‘?:’ operator act as a ternary operator, it first evaluate the condition, if it is true then the ‘exp1’ is evaluated, if the condition is false then the ‘exp2’ is evaluated. It checks the condition logic if it is true, then the value of ‘a’ is stored in ‘big’ otherwise the value of ‘b’ is stored in ‘big’

Program using conditional operator.

```

#include<stdio.h>
#include<conio.h>
void main()
{
int a=5, b=3, big;
big=a.b?a:b;
printf("big is....%d", big);
}

```

Output:

big is...5

It check the condition ‘a>b’ if it is true, then the value of ‘a’ is stored in ‘big’ otherwise the value of ‘b’ is stored in ‘big’

3.5.1.7 Bitwise operator

Operators	Meaning
&	Bitwise AND
	Bitwise OR
^	Bitwise XOR
<<	Shift left
>>	Shift right
~	One's complement

- **Truth Table for '&' Bitwise AND**

&	0	1
0	0	0
1	0	1

Example:

X=7 = 0000 0111

Y=8 = 0000 1000

X&Y = 0000 0000

- **Truth Table for '|' Bitwise OR**

	0	1
0	0	1
1	1	1

Example:

X=7 = 0000 0111

Y=8 = 0000 1000

X|Y = 0000 1111

- **Truth Table for '^' Bitwise XOR**

^	0	1
0	0	1
1	1	0

Example:

X=13 =0000 1101

Y=8 =0000 1000

X^Y =0000 0101

If either of the operand bit is high (1) then it gives high (1) result. If both operand bits are same then it gives low (0) result

A	B	a b	a&b	a^b	~a
0	0	0	0	0	1
0	1	1	0	1	1
1	0	1	0	1	0
1	1	1	1	0	0

3.5.1.8 Special operator

Operators	Meaning
,	Comma operator
sizeof	sizeof operator
& and *	Pointer operator
. and →	Member selection operator

(a) Comm operator (,):

Example: val= (a=3, b=9, c=77, a+c);

Where, first assign the value 3 to a.

Then assign the value 9 to b.

Then assign the value 77 to c.

Finally assign 80 to the val.

The comma operator is used to separate the statement elements such as variables, constants and expression.

(b) The size of () operator:

Syntax : sizeof(var)

Description : var is the variable for which to find the size.

Example:

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int a;
    printf("size of variable a is .%d", sizeof(a));
}
```

Output_: size of variable a is 2

The sizeof() operator is a compile time operator.

(c) **Pointer operator:**

& : the address of the variable

* : the value of the variable

(d) **Member selection operators:**

And \rightarrow : these symbols used to access the elements from a structure.

3.5.2 Expressions

An expression represents data item such as variables, constants and are interconnected with operators as per the syntax of the language an expression is evaluated using assignment operator.

Syntax : variable=expression

Description : Any 'c' valid variable and expression.

Example : x=a*b-c

In the above statement the expression evaluated first from left to right. After the evaluation of the expression the final value is assigned to the variable from right to left

Example of algebraic expression and C expression

Algebraic Expression	C Expression
$a+b \times c$	$a+b*c$
ax^2+bx+c	$a*x*x+b*x+c$
$\left[\frac{4ac}{2a} \right]$	$(4*a*c)/(2*a)$
$\left[\frac{2x^2}{b} \right] - c$	$((2*x*x)/b)-c$
$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$	$(-b+\text{sqrt}(b*b-4*a*c))/(2*a)$ (or) $(-b-\text{sqrt}(b*b-4*a*c))/(2*a)$
πr^2	$3.14*r*r$

3.6 MANAGING INPUT AND OUTPUT OPERATORS.

Input output statements:

- (a) Unformatted I/O statements
- (b) Formatted I/O statements.

3.6.1 Unformatted I/O statements

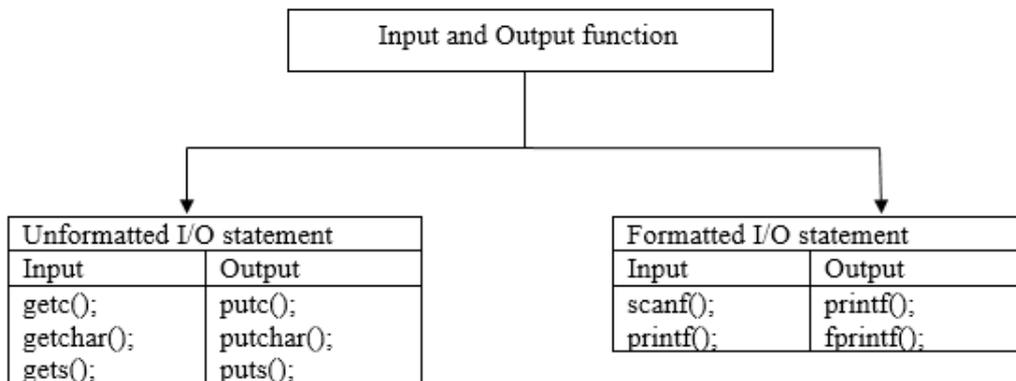


Fig 3.2. Unformatted I/O statement

The following are the unformatted I/O statements available in 'C'.

Input	Output
getchar()	Putchar()
get()	putc()
gets()	puts()

Single Character Output – Putchar() Function

Syntax : putchar (character variable)

Description: character variable is the valid 'c' variable of the type of char data type

Example : char x; x=getchar();

It reads a single character from a standard input device.

Example:

Program to print the character standard input device.

```

#include<stdio.h>
#include<conio.h>
void main()
{
char ch;
printf("Enter any character/digits....");
ch=getchar();
if(isalpha(ch)>0)
printf("it is a alphabet");
else if(isdigit(ch)>0)

```

```
printf("it is a digit");  
else  
printf("it is alphanumeric");  
}
```

Output:

1. Enter any character/digits.....a
It is a alphabet
2. Enter any character/digit.....9
It is a digit
3. Enter any character/digit..... #
It is a alphanumeric

Single character Output-putchar () function:

Syntax : Putchar(character variable);

Description: Character variable is a valid 'c' variable of the type of char data type.

Example : char x; putchar(x);

The Putchar () function is used to display one character at a time on the standard output device.

The get () function:

This is used to accept a single character from the standard input to a character variable.

Syntax: character variable = get ()

Description : character variable is the valid 'c' variable of the type of char data type

Example : char c; c=getc();

The put() function:

This is used to display a single character in a character variable to standard output device.

Syntax : put (character variable)

Description: character variable is the valid 'c' variable of the type of char data type.

Example : char c; putc(c);

The gets() and put () function;

The gets() function is used to read the string from the standard input device (keyboard)

Syntax: gets(char type of array variable)

Description: valid 'c' variable declared as one dimension char type.

Example : gets(s)

The *put-s()* function is used to display / write the string to the standard output device (monitor).

Syntax: puts (char type of array variable);

Description: valid 'c' variable declared as one dimension char type

Example:

Print the output from standard device.

```
#include<stdio.h>
#include<conio.h>
void main()
{
char scientist[40];
puts("Enter Name");
gets(scientist);
puts("print the name");
gets(scientist);
}
```

Output:

```
Enter name      : abdul kalam
Print the name  : abdul kalam
```

3.6.2 Formatted I/O Statement

Input	Output
scanf()	printf()
fscanf()	fprintf()

The scanf() function

(a) Syntax:

```
scanf("control string", &var1, &var2, ..., &varn);
```

(b) Description:

The control string consists of character groups. Each character group must begin with a percentage sign '%' followed by conversion character as specified. Var1 var2 varn – are the arguments or variable in which the data is going to be accepted.

(c) Example:

```
int n;  
scanf("%d",&n);
```

The printf () function

(a) Syntax:

```
printf( control string “, var1, var2, varn);
```

(b) Description:

Control string is any of the following

- (a) Format code character
- (b) Execution character set
- (c) Character/string that will be displayed

Var1, var2, varn are the arguments or variable from which the data is going to output

(c) Example:

```
printf(“the result is .....%d”,n);  
printf(“%f”,&f);  
printf(“%s”,s);
```

Example:

program to print the sum of two number

```
#include<stdio.h>  
#include<conio.h>  
void main()  
{  
int a, b, c;  
a=6;  
b=8;  
c=a+b;  
printf(“sum of two number is %d\n”,c);  
getch();  
}
```

Output:

Sum of two numbers is 14.

3.7 DECISION MAKING

Control statement

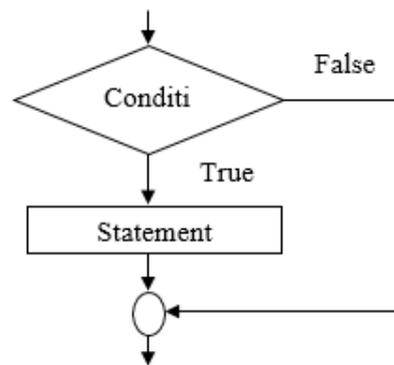
'C' language provides the following conditional (decision making) statements

- If statement
- If Else statement
- Nested ifelse statement
- If...else ladder

The if statement

Syntax

```
if(condition is true)
{
    True statements;
}
```



Description:

If the condition is true, then the true statement are executed. The true statement may be a single statement or group of statement. If the condition is false then the true statements are not executed instead. The program skip past it.

The condition is given by the relational operator like ==, !=, <=, >=, etcExample:

Example:

The following Program illustrate the useof IF with multiple statement.

```
#include<stdio.h>
void main()
{
    int i;
    printf("enter the number less than 10...");
    scanf("%d",&i);
    if(i<=10)
    {
        printf("%d", i);
        printf("\n the entered number %d is less than 10",i);
    }
}
```

```

}
}

```

Output:

Enter the number less than 10...5
 The entered number 5 is less than 10.

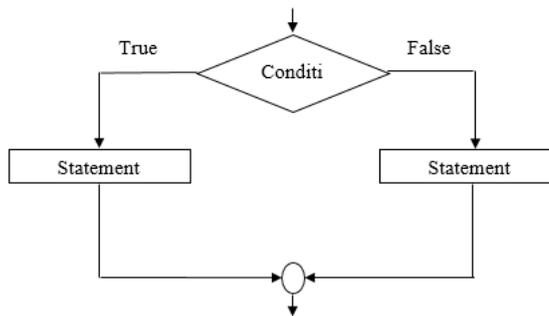
The if ...else statement

Syntax

```

if(condition)
{
    True statements;
}
else
{
    false statement;
}

```



Description:

It is used to execute some statement when the condition is true and execute some other statement when the condition is false.

Example

Program to find biggest among two number

```

#include<stdio.h>
#include<conio.h>
void main()
{
    int i,j,big;
    printf("enter two values:");
    scanf("%d%d",&i,&j);
    big=i;
    if(big<j)
    {
        big=j;
    }
    printf("biggest of two numbers is %d\n",big);
}

```

```

if(i<j)
{
big=j;
}
else
{
big=i;
}
printf("biggest of two numbers (using else) is %d\n",big);
getch();
}

```

Output:

```

Enter two values:45  78
biggest of two numbers is  78
biggest of two numbers(using else) is 78

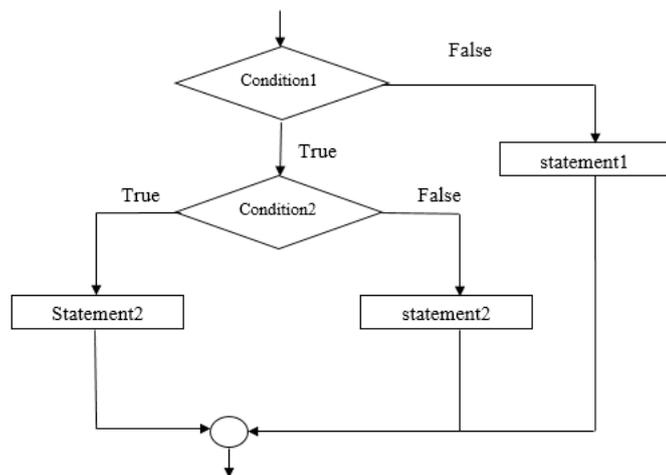
```

Nested if Else statement**Syntax:**

```

if(condition 1)
{
    if(condition 2)
    {
        True statement 2;
    }
    else
    {
        False statement 2;
    }
}
else
{
    False statement 1;
}

```



Description:

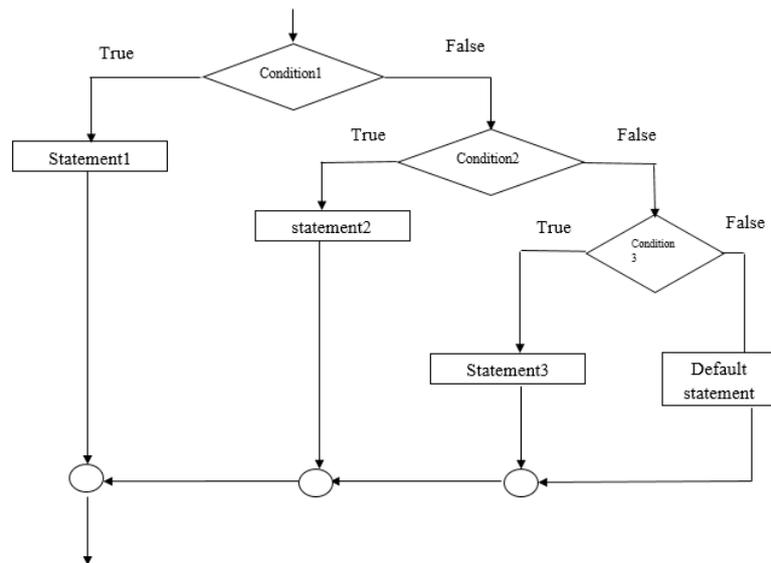
when a series of if ...else statement are occurred in a program, we can write an entire if ... else statements in another if ...else statement called nesting, and the statement is called nested if.

The if ...else ladder**Syntax:**

```

if(condition 1)
{
statement 1;
}
elseif(condition 2)
{
statement 2;
}
elseif(condition 3)
{
statement 3;
}
else
{
Default-statement;
}

```

**Description:**

Nested if statement can become quite complex. If there are more than three alternatives and indentation is not consistent, it may be different for you to determine the logical structure of the if statement. In situation you can use the nested if as the else if ladder.

Example:

Program to find biggest among two number.

```

#include<stdio.h>
#include<conio.h>
void main()
{
char ch;
clrscr();

```

```

printf("Enter a single character:");
scanf("%c",&chr);
if((chr>='a' && chr<='z' || (chr>='A' && chr<='Z'))
printf("Entered character is an alphabetic.\n);
else
if(chr>='0' && chr<='9')
printf("Entered character is an digit.\n);
else
printf("Entered character is an special character.\n);
getch();
}

```

Output:

```

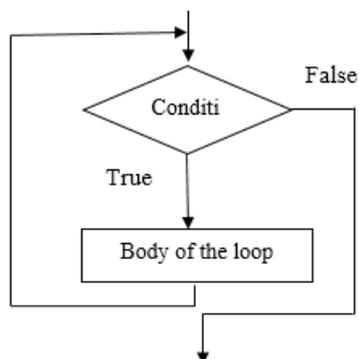
Enter a single character: M
Entered character is an alphabetic
Enter a single character: 5
Entered character is a digit
Enter a single character: &
Entered character is a special character\n/p:

```

3.8 BRANCHING AND LOOPING

The following loop structures available in 'c'

- While
- Do...while
- For....

The while loop

Syntax:

```
While (condition)
{
.....
Body of the loop;
.....
}
```

Description:

It is a repetitive control structure, used to execute the statements within the body until the condition becomes false.

The while loop is an entry controlled loop statement, means the condition is evaluated is executed after executing the body of the loop the condition is once again evaluated and if it is true, the body is executed once again, the process of repeated execution of the body of the loop continuous until the condition finally becomes false and the control is transferred out of the loop.

Example:

Program to find simple interest using while loop structure.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int p, n, count=1;
float r,si;
clrscr();
while(count<=3)
{
printf("Enter the value of P, N and R :");
scanf("%d%d%f", &p, &n, &r);
si=(p*n*r)/100;
printf("simple interest=%f\n",si);
count++;
}
getch();
}
```

Output:

```
Enter the value of P, N and R: 2000 3 5
    Simple interest = 300.00000
Enter the value of P, N and R: 3500 6 1.5
    Simple interest = 315.00000
Enter the value of P, N and R: 6000 2 3.5
    Simple interest = 420.00000
```

The do...while loop**Syntax:**

```
do
{
.....
Body of the loop;
.....
}
While(condition);
```

Description:

The while loop makes a test of condition before the loop is executed. Therefore, the body of the loop may not be executed at all, if the condition is not satisfied at first attempt. In some situations it may be necessary to execute the body of the loop before the test condition is performed, such as a situation the do...while loop is useful

Example:

Program to print n consecutive integers with infinite limit using do...while loop.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int i, n;
clrscr();
printf("Enter the number:");
scanf("%d",&n);
i=0;
do
```

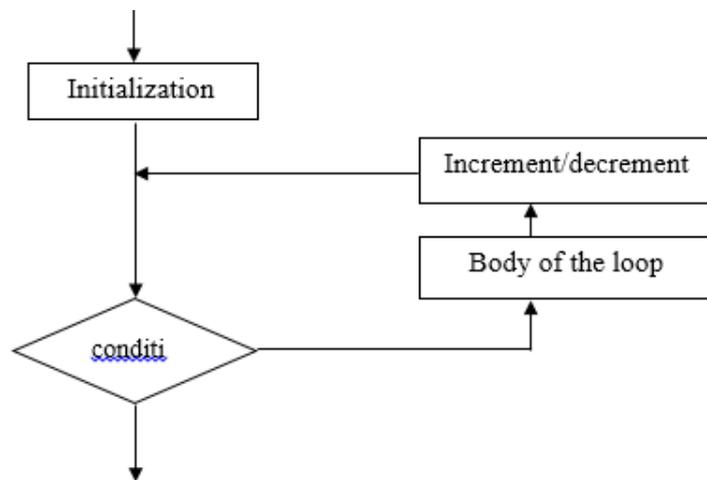
```

{
printf("the numbers are %d\n",i);
i=i+1;
}
while(i<n);
getch();
}
    
```

Output:

Enter the number: 6
 The numbers are 0
 The numbers are 1
 The numbers are 2
 The numbers are 3
 The numbers are 4
 The numbers are 5

The for loop



Syntax:

```

For(initialize counter; test condition; increment/ decrement counter)
{
.....
Body of the loop;
.....
}
    
```

Description:

The for loop is another repetitive control structure, and is used to execute set of instructions repeatedly until the condition becomes false.

The assignment, incrementation or decrementation and condition checking is done in for statement only, where as other control structures are not offered all these features in one statements.

For loop has three parts.

(a) **Initialise** counter is used to initialize counter variable.

(b) **Test condition** is used to test the condition

(c) **Increment/decrement** counter is used to increment or decrement counter variable.

Example:

Program to print n numbers using for.....loop structure.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int i, n;
clrscr();
printf("Enter the number:");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("the numbers are %d\n",i);
}
getch();
}
```

Output:

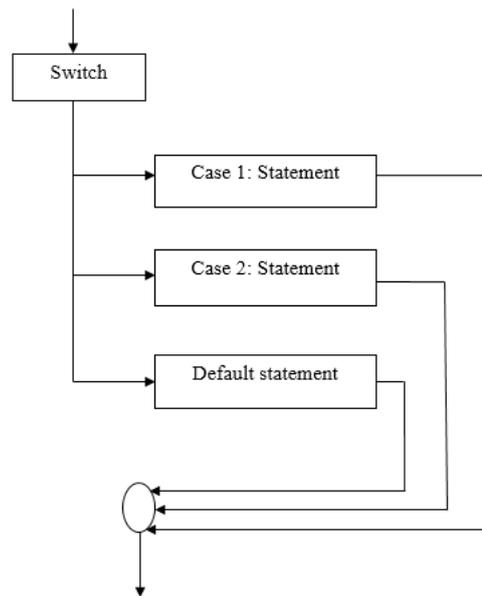
Enter the number: 3

The numbers are 0

The numbers are 1

The numbers are 2

The switch() Statement



Syntax:

```

switch(expression_s)
{
    case constant 1:
        block1;
        break;
    case statement 2:
        block2;
        break;
    default:
        default block;
        break;
}
  
```

The switch (expression)

In the block the variable or expression can be a character or an integer. The integer may be any value 1,2,3,etc. in case of character constant, the values may be with alphabets such as 'x','y','z' etc.

The switch() organisation

The switch () expression should neither be terminated with (;) semicolon nor with any other symbol. The entire case structure following switch () should be enclosed with curly braces.

The keyword `case` is followed by a constant. Every constant terminates with a colon (`:`) here, the keyboard case `&` and `break` performs respectively the job of opening and closing curly braces.

The `switch()` execution

When one of the cases satisfies, the statements following it are executed. In case there is no match, the default case is executed.

The `break` statement used in `switch()` passes control outside the `switch()` block. By mistake if no `break` statements are given all the cases following it are executed. Program, to use the computer as calculation the user input a code `+`, `-`, `*`, `/` and values that are to be computed.

Example:

Program to use the computer as calculator. The user input a code `+`, `-`, `*`, `/` and values that are to be computed.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a,b,c=0;
char op;
clrscr();
printf("\n + ADD \n - SUB \n * MUL \n / Div \n");
printf("Enter code.....");
scanf("%c",&op);
printf("Enter values.....");
scanf("%d%d",&a,&b);
switch(op)
{
Case '+':
c=a+b;
break;
Case '-':
c=a-b;
break;
Case '*':
```

```

    c=a*b;
    break;
    Case '/':
    c=a/b;
    break;
}
printf("Result is %d\n",c);
getch();
}

```

Output:

```

Calculation code
+ ADD
- SUB
* MUL
/ DIV
Enter code.....+
Enter values..... 20 10
Result is.....30

```

The Break statement

The keyword `break` allows the programmers to terminate the loop. The `break` skips from the loop or block in which it is defined.

The control then automatically goes to the first statement after the loop or block the `break` can be associated with all conditional statement.

The Continue statement

The `continue` statement is exactly opposite to `break`. The `continue` statement is used for continuing next of loop statements it is useful when we want to continue the program without executing any part of the program.

The goto statement

This statement passes control anywhere in the program, that is, control is transferred to another part of the program without testing any condition.

```
Goto label;
```

ARRAYS AND STRUCTURES

4.1 ARRAY

4.1.1 Introduction

Consider the following example:

```
main()
{
int a=2;
a=4;
printf(“%d”, a);
}
```

Output: 4

Declaration of an array is done as follows

```
int a[5];
```

It tells the compiler that a is an integer type of array and can store 5 integers.

4.1.2 Array Initialization

```
int a[5] = { 1,2,3,4,5};
```

Here, 5 elements are stored in an array a. calling array elements.

a[0] refer to 1st element i.e. 1

a[1] refer to 2nd element i.e. 2

a[2] refer to 3rd element i.e. 3

a[3] refer to 4th element i.e. 4

a[4] refer to 5th element i.e. 5

4.1.3 Definition of array

An array is a collection of similar data types in which each element is located in separate memory locations.

(a) One-Dimensional Array

The collection of data items can be stored under a one variable name using only one subscript, such a variable is called the one-dimensional array.

The element of an integer array `a[5]` are stored in continuous memory locations it is assumed that the starting memory location is 2000 each integer element requires 2 bytes.

Integer data types and their memory locations.

Element	A[0]	A[1]	A[2]	A[3]	A[4]
Address	2000	2002	2004	2006	2008

Data type and their required bytes.

Data type	Memory Requirement
Character	1 byte
Integer	2 bytes
Floating Point	4 bytes
Long Integer	4 bytes
Double Float	8 bytes

Example:

Program to display character array with their address.

```
#include<stdio.h>
#include<conio.h>
void main()
{
char name[10]={'A', 'R', 'R', 'A', 'Y'};
inti=0;
clrscr();
printf("\n character memory location \n");
while(name[i]!='\0')
{
printf("\n [%c] \t\t [%u]", name[i], &name[i]);
i++;
}
getch();
}
```

Output:

Character	Memory location
[A]	4054
[R]	4055
[R]	4056
[A]	4057
[Y]	4058

The elements of an array are stored in continuous memory location. The elements of one-dimensional array A, R, R, A, Y are stored from location 4054 to 4058.

The element of an array are stored in continuous memory location. The element of one-dimensional array A,R,R,A,Y, are stored from location 4054 to 4058

(b) Two Dimensional Array

A two dimensional array can be thought of as a rectangular display of element with rows and columns. For example, elements of `int x[3][3]`

Array elements in matrix form

	Col 1	Col2	Col3
Row 1	X[0][0]	X[0][1]	X[0][2]
Row 2	X[1][0]	X[1][1]	X[1][2]
Row 3	X[2][0]	X[2][1]	X[2][2]

A two-dimensional array is a collection of a number of one-dimensional arrays, which are placed one after another. For example, in the above table each row of a two-dimensional array can be thought of as a single- dimensional array.

Example

Write a program to display the elements of a two-dimensional array.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    inti,j;
    int a[3][3]={{1,2,3},{4,5,6},{7,8,9}};
    clrscr();
    printf("\n Elements of an array \n\n");
    for(i=0;i<3;i++)
    {
```

```

    for(j=0;j<3;j++)
        printf("%5d", a[i][j]);
        printf("\n");
    }
    getch();
}

```

Output:

Elements of an array

1	2	3
4	5	6
8	9	

(c) Three or Multi Dimensional Arrays

The C program allows array of two or multi dimensions.

Syntax:

Array name [s1] [s2] [s3]...[sn]

For examples are, int mat[3][3]

Three dimensional array can initialized as follows.

```

int mat[3][3][3]={ {
    {1,2,3}
    {4,5,6}
    {7,8,9}
};
{
    {1,4,7}
    {2,5,8}
    {3,6,9}
};
{
    {1,4,4}
    {2,4,7}
    {6,6,3}
};
}

```

Example

Write a program to explain the working of a three- dimensional array.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int array_3d [3][3][3];
inta, b, c;
clrscr();
for(a=0;a<3;a++)
for(b=0;b<3;b++)
for(c=0;c<3;c++)
array_3d[a][b][c]=a+b+c;
for(a=0;a<3;a++)
{
printf("\n");
for(b=0;b<3;b++)
{
for(c=0;c<3;c++)
printf("%3d", array_3d[a][b][c]);
}
getch();
}
```

Output:

```
0  1  2
1  2  3
2  3  4
1  2  3
2  3  4
3  4  5
2  3  4
3  4  5
4  5  6
```

4.2 HANDLING OF CHARACTER STRINGS

4.2.1 String manipulation

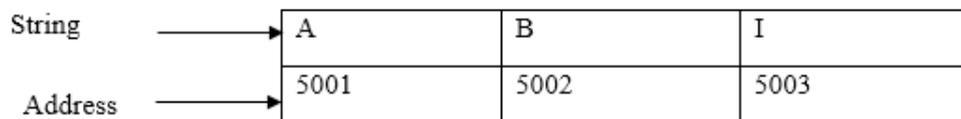
In 'C' language the group of character, digits and symbols enclosed within quotation marks are called as string otherwise string strings are array of characters.

Null character ('\0') is used to mark the end of the string

Example:

```
Char name [ ] = {'A', 'B', 'I', '\0'};
```

Each character is stored in one bytes of memory and successive character of the string are stored in successive byte.



4.2.2 Initialization of string

The string can be initialized as follows

```
Char name[ ] = "ABI";
```

The characters of the string are enclosed within a pair of double quotes

Reading and writing string

The "%s" control string can be used in scanf, statement to read a string from the terminal and the same may be used to write string to the terminal in printf() statement

Example:

```
char name[10];
scanf("%s",name);
scanf("%s",name);
```

There is no address (&) operator used in scanf() statement.

The commonly used string manipulation functions are follows:

(1) The strlen() function:

This function is used to count and return the number of character present in a string

Syntax	Description
Var=strlen(string);	Var: Is the integer variable, which accepts the length of the string
	String: Is the string constant or stringvariable, for which the length is going to be found.

Example:

Write a program using strlen() function.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int len1,len2;
char name[]="ABI";
clrscr();
len1=strlen(name);
len2=strlen("SHEK");
printf("\n String length of %s is %d",name,len1);
printf("\n String length of %s is %d","SHEK",len2);
}
```

Output:

String length of ABI is 3
String length of SHEK is 4

(2) The strcpy() function

This function is used to copy the contents of one string to another and it almost works like string assignment operate

Syntax	Description
strcpy(string1,string2);	String 1 is the destination string. String 2 is the source string.

i.e., The contents of string 2 are assigned to the contents of string 1. Where string 2 may be character array variable or string constant.

Example:

```
char str1[]="ABI";
char str2[]="SHE";
strcpy(str1, str2);
```

Where the contents of str2 are copied into the str1 and the contents of str1 is replaced with new one.

Example:

Write a program using strcpy() function.

```
#include<stdio.h>
#include<conio.h>
void main()
{
char target[10];
char source="ABI";
clrscr();
strcpy(target,source);
printf("\n Source string is %s",source);
printf("\n Target string is %s",target);
}
```

Output:

```
Source string is ABI
Target string is ABI
```

(3) The strcat() function:

The *strcat()* function is used to concatenate or combine, two strings together and forms a new concatenated string

Syntax	Description
strcat(string1,string2);	String 1 and String 2 are character type arrays or string constants.

when the above logo function is executed string 2 is combined with string

Example

```
strcat("ABI", "SHEK");
yields ABISHEK.
char str1="ABI"
```

```
char str2="SHEK"
strcat(str1,str2)
yields ABISHEK
```

(4) The strcmp() function:

This is a function which compares two strings a comparison of two strings can be made up to certain specified length

```
Strcmp(source, target, argument);
```

where, argument is number of characters up to which the comparison is to be made write a program to compare two string up to specified length.

Example:

Write a program to compare two strings upto specified length.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
charsr[10],tar[10];
intn,diff;
clrscr();
printf("Enter string(1):");
gets(sr);
printf("Enter string(2):");
gets(tar);
printf("\n Enter length up to which comparison is to be made");
scanf("%d",&n);
diff=strcmp(sr,tar,n);
printf("The two strings are identified up to %d characters :",n);
else
puts("The two strings are different");
getch();
}
```

Output:

Enter string (1): HELLO
 Enter string (2): HE MAN
 Enter length up to which comparison is to be made: 2
 The two strings are identified up to 2 characters

(5) The *strrev()* function:

The *strrev()* function is used to reverse a string

Syntax	Description
<code>strrev(string);</code>	String are character type arrays or string constants.

Example

Write a program to reverse a string

```
#include<stdio.h>
main ()
{
char y[30];
printf("Enter the string:");
gets(y);
printf("the string reversed is:%s",strrev(y));
}
```

Output:

Enter the string: book
 The string reversed is:koob

4.3 POINTERS**4.3.1 Definition of pointer**

A pointer is a memory variable that stores a memory address. It can have any name than is legal for another variable and it is declared in the same fashion like other variable but it is always denoted by '*' operator.

Pointer Declaration

For example:

```
int *x;
float *f;
char *y;
```

- (1) In the first statement x is an integer pointer and it tells the compiler that it holds the address of any integer variables. In the same way f is a float pointer, which stores the address of any float variable and y is a character pointer that stores the address of any character variable.
- (2) The indirection operator (*) is also called the dereference operator.
- (3) The indirection operator (*) is used in two distinct ways with pointer, declaration and dereference.
- (4) When a pointer is declared, the star indicates that is a pointer, not a normal variable.
- (5) When a pointer is dereferencing, the indirection operator indicates that the value stored in the pointer at that memory location is to be accessed rather than address itself.
- (6) The '&' is the address operator and it represents the address of the variable. When a pointer is dereferenced, the indirection operator indicated that the value stored in accessed rather than the address itself

Example:

Write a program to add two numbers through variables and their pointers.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
inta,b,c,d,*ap,*bp;
clrscr();
printf("Enter Two Numbers:");
scanf("%d%d", &a,&b);
ap=&a;
bp=&b;
c=a+b;
d=*ap+*bp;
printf("\n sum of A & B using variables:%d", c);
printf("\n sum of A & B using pointer:%d", d);
getch();
}
```

Output:

Enter two numbers: 8 4
 sum of A & B using variables: 12
 sum of A & B using pointer: 12

4.3.2 Arithmetic Operations with Pointer

Increase decrease prefix and postfix operations can be performed with the help of the pointer.

Pointer and arithmetic operation

Data Type	Initial address	operation		Address after operation		Required bytes
int i=2	4046	++	--	4048	4044	2
Char c='x'	4053	++	--	4054	4052	1
Float f=2.2	4058	++	--	4062	4054	4
Long l=2	4060	++	--	4064	4056	4

Example:

Write a program to perform different arithmetic operations using pointers.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a=25,b=10,*p,*j;
p=&a;
j=&b;
clrscr();
printf("\n Addition a+b=%d",*p+b);
printf("\n Subtraction a-b=%d",*p-b);
printf("\n Product a*b=%d",*p**j);
printf("\n Division a/b=%d",*p/*j);
printf("\n a mod b=%d",*p%*j);
getch();
}
```

Output:

Addition a+b=35
 Subtraction a-b=15
 Product a*b=250
 Division a/b=2
 A mod b=5

4.4 STRUCTURE AND UNION**Structure*****Declaration and Initialization of Structures:***

structures can be declared as given below

```
struct struct_type
{
type variable1;
type variable2;
};
```

Structure declaration always starts with struct keyword. Here, struct- type is known as tag. The struct declaration is enclosed within a pair of curly braces.

Using struct and tag, the user can declare structure variables like variable, variable 2 and so on. Those are the members of the structure.

After defining structure we can create variable as given below

```
struct struct_type v1,v2,v3
```

Here, v1, v2 and v3 are variables of structure struct type

```
int v1, v2, v3
```

Here, v1, v2 and v3 are variable of integer data type

Example:

```
struct book;
{
char book[30];
int pages;
float price;
};
struct book1 bk1;
```

A structure of type book is created. It consists of three members, book [30] of char data type, pages of type and price of float data type.

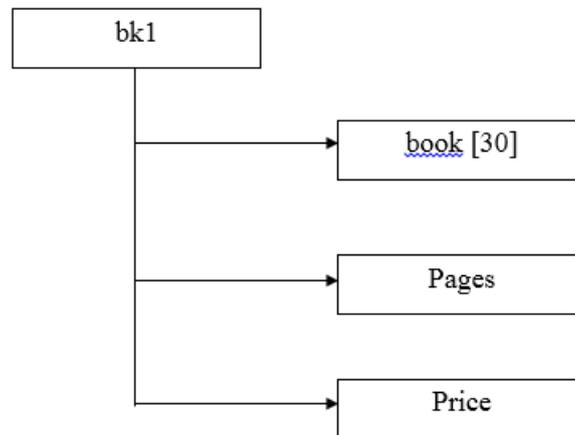


Fig 5.2. Declaration and initialization of structure

```
struct book1 bk1;
```

The above line creates variable bk of type book and it reserves a total of 36 bytes (30 bytes for book (30), 2 bytes for integer and 4 bytes for float).

Through bk1, all the three members of structure can be accessed.

In order to initialize structure elements with certain values, the following statement is used.

```
struct book1 bk1={"abishek",500,385.00};
```

All the member of structure is related to variable bk1.

```
structure_variable.member or bk1.Book
```

The period (.) sign is used to access the structure members. We can directly assign values to members as given below.

```
bk1.book="abishek";
```

```
bk1.pages=500;
```

```
bk1.price=385.00;
```

Example:

Write a program to display size of structure elements. Use sizeof() operator.

```
#include<stdio.h>
```

```
#include<conio.h>
```

```
#include<string.h>
```

```
void main()
```

```

{
struct book1
{
char book[30];
int pages;
float price;
};
struct book1 bk1;
clrscr();
printf("\n size of structure elements:");
printf("\n Book:%d", sizeof(bk1.book));
printf("\n pages:%d",sizeof(bk1.pages));
printf("\n price:%d",sizeof(bk1.price));
printf("\n total bytes:%d",sizeof(bk1));
getch();
}

```

Output:

```

Size of structure elements
Book: 30
Pages: 2
Price: 4
Total bytes: 36

```

4.4.1 UNION**Syntax:**

```

Union union_name
{
    union member1;
    union member2;
    .....
    .....
    union member3;
};
union union_variable

```

Union is a variable, which is similar to the structure. It contains a number of members like structure but it holds only one object at a time union also contains members of types int, float, long, arrays, pointers and so on.

Example:

Write a program to find size of union and number of bytes reserved for it.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
union result()
{
int marks;
char grade;
};
struct res
{
char name;
int age;
union result perf;
}data;
clrscr();
printf("Size of union :%d\n",sizeof(data.perf));
printf("\n Size of structure :%d\n", sizeof(data));
getch();
}
```

Output:

```
Size of union      :      2
Size of structure  :     19
```

4.4.1.1 Union of structures

One structure can be nested within another structure. A union can be nested within another union. We can also create structure in a union or vice versa

Example:

Write a program to use structure within union. Display the contents of structure element.

```
#include<stdio.h>
#include<conio.h>
void main()
{
struct x
{
float f;
char p[2];
};
union z
{
struct x set;
};
union=st;
st.set.f=5.5;
st.set.p[0]=65;
st.set.p[1]=66;
clrscr();
printf("\n %f",st.set.f);
printf("\n %c",st.set.p[0]);
printf("\n %c",st.set.p[1]);
getch();
}
```

Output:

5.5

A

B

4.5 USER DEFINED FUNCTIONS

Definition of function

A function is a self – contained block or a sub-program of one or more statement that performs a special task when called

4.5.4 Declaration of Function and Function Prototypes

(a) *Declaration of functions*

Function is declared as per format give below

```
function_name(argument/permanent list)
argument declaration;
{
local variable declaration;
statement1;
statement2;
return(value);
}
```

Working of functions:

```
main()
{
.....
.....
abc(x, y, z);           Function call
                        |
                        |> Actual Argument
.....
.....
}

abc(l, k, j)           Function definition
{
                        |
                        |> Return Value
.....
.....
return l;
}
```

Actual Arguments

The arguments of calling functions are actual arguments variable x,y,z are actual arguments

Formal arguments

The arguments of called function are formal arguments. Variable l,k and j are formal arguments

Function name

```
sum (int a, int b)
```

where, sum () is a user – defined function and a and b are integer variable arguments the function name must be ended by a semi colent(;

Argument / Parameter list

The argument list means variable names enclosed within the parenthesis. They must be separated by a comma(,)

Function call

A compiler executes the function when a semicolon (;) is followed by function name a function can be called simply using its name like other (statement, terminated by semicolon

Example:

Write a program to show how user_defined function is called.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int x=1,y=2,z;
z=add(x,y);
printf(“z=%d”,z);
}
/* function definition */
add(a,b)
{
return(a+b);
}
```

Output:

```
z=3.
```

Local and global variable

There are two kinds of variables

- 1) local and 2) global

4.6 TYPES OF FUNCTION

- Depending upon the arguments present return value sends the result back to the calling function
- Based on this the function are divided into four types

4.6.1 Without arguments and return values

Calling function	Analysis	Called function
<pre>main() { abc(); }</pre>	<p>No arguments are passed</p> <p>No values are send back</p>	<pre>abc() { }</pre>

- (1) Neither the data is passed through the Function nor the data is sent back from the called function, that is , there is no data transfer between calling and the called functions
- (2) The function is only executed and nothing is obtained
- (3) If such functions are used to perform any operation they act independently they read data values and print result in the same block
- (4) Such functions may be useful to print some messages, draw a line or split the live etc.

Example:

Write a program to display message using user-defined functions.

```
#include<stdio.h>
#include<conio.h>
void main()
{
void message();
message();
}
void message()
{
puts("Have a nice day");
}
```

Output:

Have a nice day

4.6.2 With arguments and without return value:

Calling function	Analysis	Called function
main() { abc(x); }	Arguments are passed No values are send back	abc(y) { }

- (1) In the above functions, arguments are passed through the calling function. The called function operates on the values. But no result is sent back
- (2) 2. such functions are partly dependent on the calling function the result obtained is utilised by the called function and there is no gain to the main()

Example:

Write a program to send value to user-defined function and display result.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int dat(int, int, int)
int d,m,y;
return();
printf("Enter Date dd/mm/yyyy");
scanf("%d%d%d",&d,&m,&y);
dat(d,m,y);
return 0;
}
dat(int x, int y, int z)
{
printf("Date=%d%d%d",x,y,z);
}
```

Output:

```
Enter Date dd/mm/yyyy 12 12 2001
Date : 12/12/2001.
```

4.6.3 With arguments and return value:

Calling function	Analysis	Called function
<pre>main() { int z; z=abc(x); }</pre>	<p>Arguments are passed</p> <p>Values are send back</p>	<pre>abc(y) { y++; return(y); }</pre>

- (1) In the above example the copy of actual argument is passed to the formal argument, that is value of is assigned to y
- (2) The return statement returns the increased value of y. the returned value is collected by z
- (3) Here data is transferred between calling and the called functions, that is, communications, between functions is made

Example:

Write a program to pass value to user-defined function collect and display the values returned by the called function.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int dat(int, int, int)
int d,m,y,t;
return();
printf("Enter Date dd/mm/yyyy");
scanf("%d%d%d%t",dat(d,m,y));
t=dat(d,m,y);
printf("\n Tomorrow=%d%d%d",t,m,y);
return 0;
}
dat(int x, int y, int z)
{
printf("Today=%d%d%d",x,y,z);
```



```

sum()
{
char x;
int y,z;
printf("Enter three values:");
scanf("%d%d%d",&x,&y,&z);
return(x+y+z);
}

```

Output:

```

Enter three values: 3    5    4
Sum = 13

```

4.7 CALL BY VALUE AND REFERENCE

There are two ways in which we can pass argument to the function (a) call by value(b) *call by reference*.

4.7.1 Call By Value

In this type, value of actual argument are passed to the formal argument and the operation is done on the formal arguments

Any change made in the formal argument does not effect the actual arguments because formal arguments are photo copies of actual argument

Hence, when function is called by the call or by value method, it does not affect the actual contents of the actual arguments

Changes made in the formal arguments are local to the block of the called function once control returns back to the calling function the changes made vanish.

Example:

Write a program to send two integer values using “call by value”.

```

#include<stdio.h>
#include<conio.h>
change(int, int);
void main()
{
int x,y;
clrscr();

```

```

printf("Enter values of x & y:");
change(x,y);
printf(" In main() x=%d y=%d",x,y);
return 0;
}
change(int a, int b)
{
int k;
k=a;
a=b;
b=k;
printf("\n In change() x=%d",a,b);
}

```

Output:

```

Enter values of  x & y:  5      4
In change()      x=4    y=5
In main()        x=5    y=4

```

4.7.2 Call By Reference

In this type, instead of passing values, addresses (reference) are passed function operate an addresses rather than values

Here the formal arguments are pointer to the actual arguments. In this type formal arguments point to the actual argument

Hence changes made in the arguments are permanent

Example:

Write a program to send a value by reference to the user-defined functions.

```

#include<stdio.h>
#include<conio.h>
void main()
{
int x,y,change(int *,int *);
clrscr();

```

```
printf("\n Enter values of x &y:");
scanf("%d%d",&x,&y);
change(&x,&y);
printf("\n in main() x=%d y=%d",x,y);
return 0;
}
void change(int *a, int *b)
{
int *k;
*k=*a;
*a=*b;
*b=*k;
printf("\n change() x=%d y=%d", *a,*b);
}
```

Output:

```
Enter values of x&y : 5 4
In change() x=4 y=5
In main() x=4 y=5.
```

4.8 RECURSION

The C language supports recursive feature, that is a function is called repetitively by itself recursion can be used directly or indirectly

Direct recursion function calls itself till the condition is true

In indirect recursion, a function calls another function, then the called function calls the calling function

Example

write a program to (a) main() function recursively and perform sum of 1 to 5 numbers

```
#include<stdio.h>
#include<process.h>
int x, s;
main(x)
{
s=s+x;
```

```

print("\n x=%d s=%d", x,s);
if (x==5)
exit(0);
main(++x);
}

```

Output

```

x= 1      s = 1
x = 2     s = 3
x= 3     s = 6
x = 4     s =10
x = 5     s = 15

```

Steps of Recursive Function

Function call	Value of x	Value of s (sum)
main (1)	x=1	s=1 (0+1) =1
main (2)	x=2	s=3(2+1+0) =3
main (3)	x=3	s=6(3+2+1+0) =6
main (4)	x=4	s=10(4+3+2+1+0) =10
main (5)	x=5	s=15(5+4+3+2+1+0) =15

INTRODUCTION TO C++

5.1 OVERVIEW OF C++

- C++ is an object – oriented programming language
- C++ was developed by bjarne stroustrup at AT & T bell laboratories in murray hill, new jersey, usa
- The idea of C++ comes from the c increment operator ++, thereby suggesting that C++ is an incremented version of C
- Most of what we already know about applies to C++ also
- Therefore, almost all c programs are also C++ program
- The three most important facilities that C++ adds on o c are classes, function overloading and operator overloading

5.2 APPLICATIONS OF C++

C++ is a versatile language for handling very large programs. It is suitable for virtually any programming tasks including development of editors, compilers, databases, communication systems and any complex real – life application system

Since C++ allows us to create hierarchy related objects, we can build special object- oriented libraries which can be used later by many programmers

While C++ is able to map the real world problem property, the c part of C++ gives the language the ability to get close to the machine – level details

When a new features need to be implemented, it is very easy to add to the existing structure of an object.

A simple C++ program

Example: Printing a string

```
#include<iostream.h>           // include header file
main()
{
count<<"C++ is better C";      // C++ statement
}
```

Program Features

- The C++ program is a collection of functions. The example contains only are function, main()
- Every C++ program must have a main()

Comments

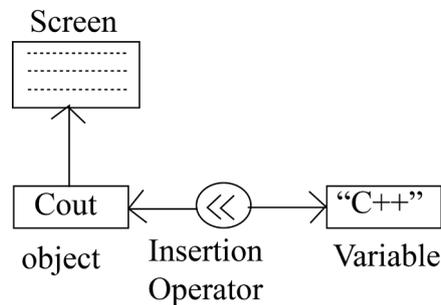
- C++ introduces a new comment symbol // (double slash)
- A comment may start anywhere in the line

C Comment Symbols	C++ Comment
/* CProgram */	//C++ Program

Output Operator

Example: cout <<"C++ is better C.";

- This statement introduces two new C++ features, cout and <<
- The identifier cout is a predefined object that represent the standard output steam in c++
- Here, the standard output stream represents the screen
- The operator << is called the insertion or put to operator.



- The object cout has a simple interface if string represents a string variable
Cout<<String;
- << → bitwise left shift operator this concept is known as operator over loading

C	C++
printf	Count<<

The iostream.h file

include < iostream .h>

- Some old versions of C++ use a header file called stream.h
- The header file iostream.h should be included at the beginning of all program that use input/output statements

Return Statement

- Every main() in c++ should end with a return (0) statement;
- Otherwise a warning or an error might occur
- Turbo c++ gives a warning and then compiles and executes the program

Average of two numbers

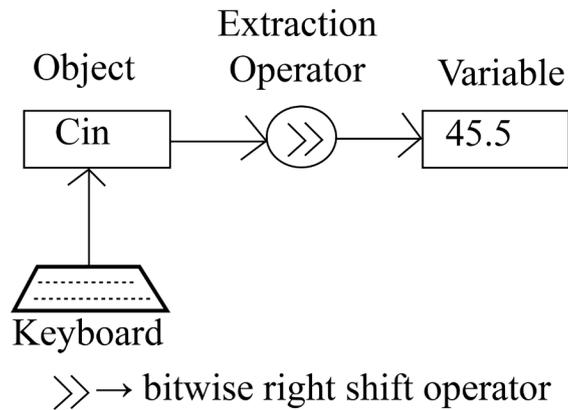
```
#include<iostream.h>
main()
{
float number1, number2,sum, average;
Cout<<"Enter two number:";
Cin>>number1;
Cin>> number2;
Sum = number 1+ number2;
average = sum/2;
Cout<<"sum="<<sum<<"\n";
Cout<<"Average = "<<average<<"\n";
}
```

Output

```
Enter two number : 6.5 7.5
Sum = 14
Average = 7
```

Input Operator

- The statement
Cin>>number1;
- The identifier Cin is a predefined object in c++ that corresponds to the standard input stream
- Here, this stream represents the keyboard
- The operator >> is known as extraction or get from operator



5.3 OOPS CONCEPTS

Object oriented remains a term which is interpreted differently by different people.

The following general concepts

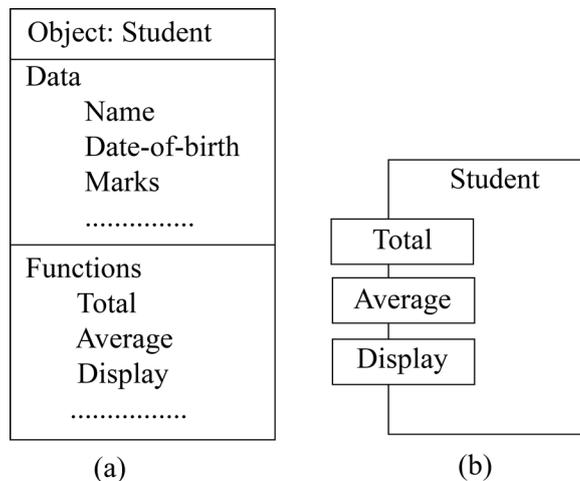
- | | | |
|------------------------|---------------------|----------------------|
| (a) Objects | (b) Classes | (c) Data abstraction |
| (d) Data encapsulation | (e) Inheritance | (f) Polymorphism |
| (g) Dynamic binding | (h) Message passing | |

Objects

Objects are the basic run – time entities in an object – oriented system they may represent a person, a place, a bank account, a table of data or any item that the program must handle, when a program is executed, the objects interact by sending messages to one another.

For example, if “customers” and “account” are two objects in a program, then the customer object may send a message to account object requesting for the bank balance.

Each object contains data and code to manipulate the data



Two ways representing an object

Classes

Object contain data and code to manipulate that data

The entire set of data and code of an object can be made a user – defined data type with the help of a *class*.

Once a class has been define, we can create any number of objects belonging to that class.

Example: fruit mango;

Will create an object *mango* belonging to the class *fruit*.

Data Abstraction and Encapsulation

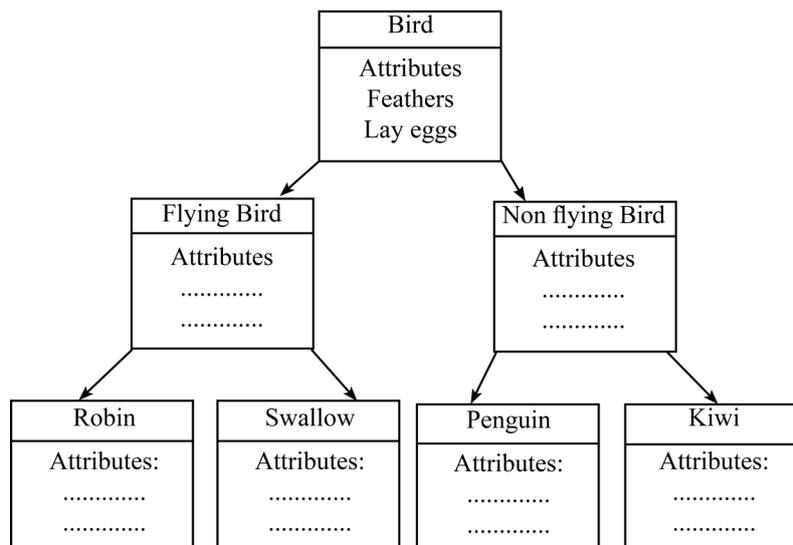
The wrapping up of data and functions into a single unit (called class) is known as encapsulation

These functions provide the interface between the objects data and the program this insulation of the data from direct access by the program is called data hiding. classes use the concept of abstraction and are define as a list of abstract attributes such as size, weight and lost and functions to operate on these attributes

Since the classes use the concept of data abstraction, they are known as abstract data types (ADT)

Inheritance

Inheritance is the process by which objects of one class acquire the properties of objects of another class it supports the concept of hierarchical classification.



For example,

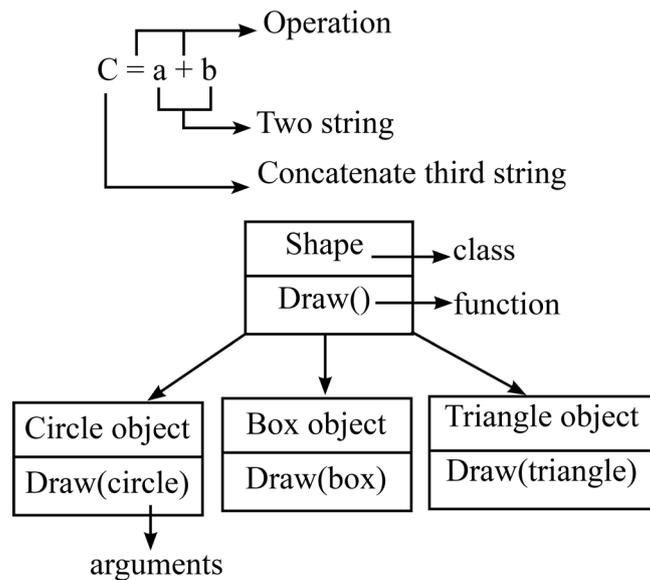
The bird *robin* is a part of class *flying bird* which is again part of the class *bird*.

Polymorphism

Polymorphism means the ability to take more than one form

For example, consider the operation of addition for two numbers, the operation will generate a sum

If the operands are strings, then the operation would produce a third string by concatenation



A single function name can be used to handle different number and different types of arguments

Dynamic Binding

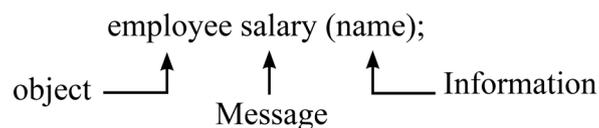
It is associated with polymorphism and inheritance

Message Communication

An object – oriented program consists of a set of object that communicate with each other

- (1) Creating classes that define objects and their behaviour
- (2) Creating objects from the class definitions
- (3) Establishing communication among objects

Example



Message passing involves specifying the name of the object, the name of the function (message) and the information to be sent.

5.4 CLASSES

A class is a way to bind the data and its associated functions together

Class specification has two parts

- (1) Class declaration
- (2) Class function definitions

The *class declaration* describes the type and scope of its members

The *class function* definitions describe how the class functions are implemented

General form of a class declaration is

```

Class class-name
{ private:
  Variable declaration;
  Function declarations;
  Public:
  Variable declarations;
  Function declarations;
}

```

The *class* declaration is similar to a *struct* declaration. The keyword *class* specified that what follows is an abstract data of type

5.4.1 Class – name

- The body of a class is enclosed within braces. The class body contains the declaration of variable and functions.
- These functions and variable are collectively called members
- They are usually grouped under two sections, names, private and public
- The keywords private and public are known as visibility labels. Keywords are followed by a colon
- The members that have been declared as *private* can be accessed only from within the class
- *Public* members can be accessed from outside the class also

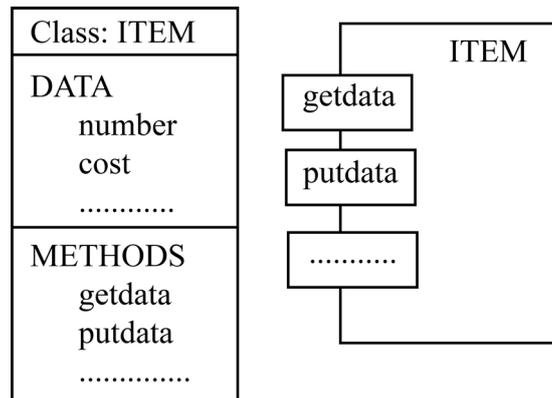
Example:

```

Class item
{
  int number;      //variable declaration
}

```

```
float cost; //private by default
public:
void getdata (int a, float b); //functions declaration
void putdata (void); //using prototype
};
```



(a) (b)
Representation of a class

The class *item* contains two data member and two function members

The data members are private by default while both the functions are public by declaration

The function `getdata ()` can be used to assign values to the member variables `number` and `cost`

And `putdata ()` for displaying their values

5.4.2 Creating Objects

Remember that the delaration of `item` as shown above does not define any objects of `item`. Once a class has been declared, we can create variables of that type by using the class – name

For example:

`item x ;` // memory for `x` is created creates a variable `x` of type `item` in `c++`, the class variable are known as objects

Therefore, `x` is called an object of type `item`

we may also declare more than one object in one statement

Example:

```
item x,y,z
```

Objects can also be created when a class is defined by placing their names immediately after

the closing brace, as we do in the case of structures.

```

Class item
{
.....
.....
.....
} x,y,z;

```

would create the objects x,y and z of type item.

Example: Class Implementation

```

#include <iostream.h>

Class item                                //class declaration
{
int number;                               //private by default
float cost;                               //private by default
public:
void getdata (int a, float b);           //function defined here
{
cout<<"number:","number,","\n";
cout<<"cost:"<<cost<<"\n";
}
}

//.....Member Function Definition.....

Void item :: getdata (int a, float b)
    ↓      ↳Scope resolution ↳argument declaration
Class name
{
number = a;
cost =b;
}

//..... Main Program.....

main ()
{

```

```
item x;          //create object x
cout<<"\n object x" <<"\n";
x.getdata (100,299.95);          //Call member function
x.putdata(0;                    //Call member function
item y;
cout<<"\n object y"<<"\n";
y.getdata (200,175.50);
y.putdata ();
}
```

Output

Object x

Number : 100

Cost: 299.950012

Object y

Number: 200

Cost : 175.5

5.5 FRIEND FUNCTIONS

C++ allows the common function to be made friendly

To make an outside function “friendly” to a class, we have to simply declare this function as a friend of the class as shown below

```
class ABC
{
.....
.....
public:
.....
.....
friend void xyz (void);          // declaration
};
```

The function declaration should be preceded by the keyword friend

The function that are declared with the keyword friend are known as friend functions.

A function can be declared as a friend in any number of classes.

A friend function, although not a member function, has full access rights to the private members of the class.

Friend function

```
#include<iostream.h.
class sample
{ int a;
  int b;
public:
  void setvalue() [a=25; b=40;]
  friend float mean (sample s);           // friend declared
};
float mean (sample s)
{
  return float (s.a+s.b)/2.0;
}
main()
{
  sample x;           // object x
  x.setvalue ();
  cout<<"Mean value = ",mean (x) << "\n";
}
```

Output

Mean value : 32.5

5.6 FRIEND CLASS

All the member functions of one class as the friend functions of another class

The class is called a friend class

```
Class z
{...
  Friend class x;
};
```

Friend class

```
#include<iostream.h.
class ABC;           // Forward declaration
class xyz
{
int x;
public:
void setvalue(inti) {x=i;}
friend vois max (XYZ, ABC);
};
class ABC
{
int a;
public:
void setvalue(int i) {a=i;}
friend void max (XYZ, ABC);
};
void max (XYZm, ABC n) //Definition of Friend
{ if(m.x>=n.a)
cout<<m.x;
else cout<<n.a;
}
main()
{
ABC abc;
abc.setvalue (10);
XYZ xyz;
xyz.setvalue (20);
max(xyz, abc);
}
```

Output: 20

5.7 CONSTRUCTIONS

A constructor is a special member function whose task is to initialize the objects of its class.

The constructor is invoked whenever an object of its associated class is created.

A constructor is declared and defined as follows:

```
// class with a constructor
Class integer
{
int m,n;
public:
integer (void);          // constructor declared
.....
.....
};
integer :: integer (void) //constructor defined
{
m=0; n=0;
}
```

Class with Constructor

```
#include<iostream.h>
class integer
{
int m,n;
public:
integer(Int, int);      //constructor declared
void display(void)
{
cout<<"m="<<m<<"\n";
cout<<"n="<<n<<"\n";
}
};
integer :: integer (int x, int y)          //constructor defined
```

```

{
m=x; n=y;
}
main()
{
integer Int1 (0,100);
integer Int2 (25, 75);
cout<<"\n object1 '<<"\n";
Int1.display ();
cout<<"\n object 2'<<"\n";
Int2.display();
}

```

Output

```

Object 1   Object 2
m=0        m=25
n=100      n=75

```

5.8 DESTRUCTORS

A destructor, as the name implies, is used to destroy the objects that have been created by a constructor

Like a constructor, the destructor is a member function whose name is the same as the class name but is preceded by a tilde. For example, the destructor for the class integer can be defined as shown below

```
~integer () {}
```

Example

```

#include<iostream.h>
int count =0;
class alpha
{
public:
alpha()
{
count++

```

```
cout<<"\n No. of object created"<<count;
}
~alpha()
{
cout<<"\n No. of object destroyed"<<count;
count --;
}
};
main()
{
cout<<"\n\n Enter main \n";
alpha A1, A2, A3, A4;
{
cout<<"\n\n Enter Block1 \n";
alpha A5;
}
{
cout<<"\n\n Enter Block 2 \n";
alpha A6;
}
cout<<"\n\n Re-enter main\n";
}
```

Output

```
Enter main
No. of object created1
No. of object created 2
No. of object created 3
No. of object created 4
Enter block1
No of object created 5
No . of object destroyed 5
Enter block2
```

No.of object created 5

No. of object destroyed 5

Re-enter main

No.of object destroyed 4

No. of object destroyed 3

No. of object destroyed 2

No. of object destroyed 1

As he objects are created and destroyed, they increase and decrease the count. After the first group of objects is created, a5 is created, and the destroyed, a6 is created and then destroyed.

Note that the objects are destroyed in the reverse order of creation.