

# C Programming

## Notes

By Jobs Internships wale

For more such helpful notes and regular  
jobs updates, join our telegram channel

<https://telegram.me/jobsinternshipswale>

e

# B.tech Sem 1+2 Unit 1

Unit - I

Date: 26/08/19 Page: 1.

## INTRODUCTION OF COMPUTER

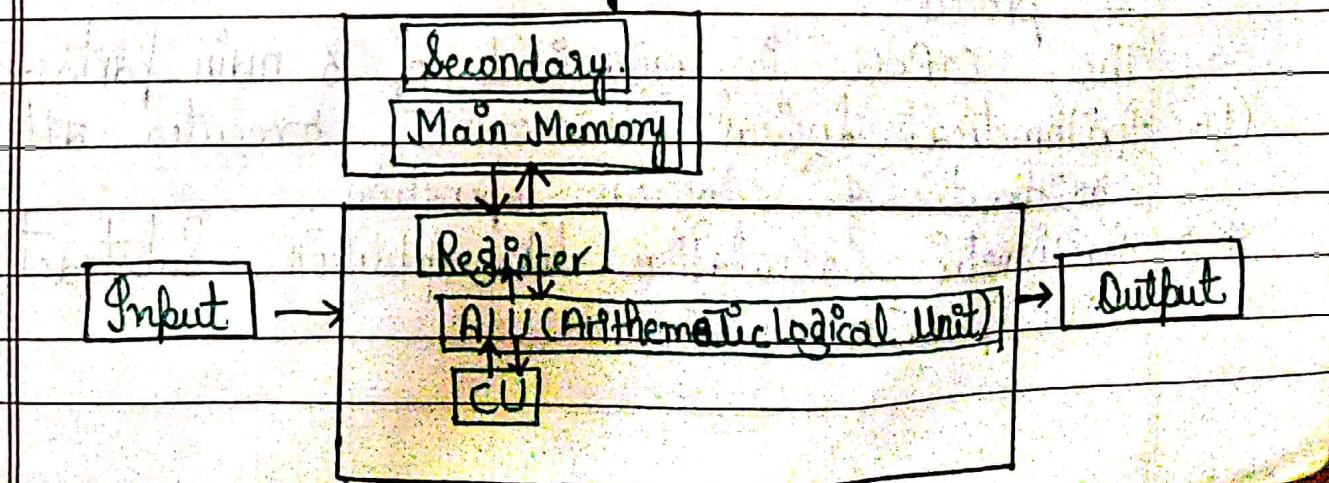
- \* Computer :- If Comp is an electric device, operating under the control of instruction stored in it's own memory that can accept (input), process the data according to specified rules, produce information (output) & store the information for future use.

Data (input) → Processing → Information (Output).

- \* Functionality of a Computer :- Any digital comp carries out 5 functions in gross term.

1. Takes data as a input
2. Stores the data/instructions in it's memory & use them when required.
3. Process the data & converts it into useful informat
4. Generates the Output
5. Controls all the above 4 steps.

- \* Components of a Comp system :-



→ They kind of comp, consist of a hardware & software.

- \* HARDWARE :- Comp hardware is the collection of physical element that complete a Computer System.
- Comp hardware refers to the physical parts or components of a computer such as the monitor, mouse, key board, comp data storage, hard drive, disk & system units as a graphic cards, sound cards, Mother-board & chips & so on.
- All of which are physical objects that can be touch.

1. Input Devices :- I.D is any any peripheral input device. That translate data, from that humans Trans Beings understand to one that comp can work with.

Most Common I.D are keyboard, Mouse, Microphone, Touch Screen, speaker, scanner, web cam, touch pad, light pen & joystick.

- Central Processing Unit (CPU) :- The CPU is the Brain of a Computer. It is responsible for all function of process.

The CPU is organised of 3 main parts -

(1). Arithmetical logical Unit :- It executes all arithmetic & logical operation.

\* → Arithmetic Calculation like Addition, Subtraction, Multiplication & Division.

\* → logical operation like Comparing nos, letters or special characters.

(2). Control Unit :- It controls & co-ordinates comp components.

(3). Registers :- Registers stores the data i.e to be executed next. "Very fast storage Area".

→ \* 1<sup>o</sup> Memory / Main Memory / Internal Memory - The memory that is directly connected to the processor & is called 1<sup>o</sup> Memory. It is used to store the data i.e currently in use. It is the fastest memory. There are two major types of 1<sup>o</sup> memory -

1. RAM (Random Access Memory) :- This memory is referred to as the read & write memory. The input data & instructions are first entered in RAM & then transferred to the 2<sup>o</sup> memory if required. It is a volatile in nature.

\* Types of RAM :- Types of RAM are as followed.

(i) Dynamic RAM (DRAM) & (ii) Static RAM (SRAM).

(2). ROM (Read Only Memory) :- This memory reads only the programme stored in it during manufacturing. It is a non-volatile in a nature.

\* Types of ROM :- Types of ROM are as followed.

- (1) Programmable Read Only Memory (PROM)
- (2) Erasable Programmable Read Only Memory (EPROM) & erase the information in the form of UV rays
- (3) Electricity Erasable Programmable Read Only Memory (EEPROM)

→ \* Cache Memory :- It is the fastest memory. It is used to store frequently stored data in the memory structure. It comes in b/w CPU register & RAM. The size of cache memory is very small in comparison with RAM & it is very high cost.

→ \*  $2^{\text{nd}}$  Memory or Auxiliary Memory / External Memory :- The memory i.e. is not directly connected with the processor is called  $2^{\text{nd}}$  memory.  $2^{\text{nd}}$  Memory is used to store the data i.e. is currently not in use.  $2^{\text{nd}}$  memory is of 4 types - magnetic tape, magnetic disc, optical disc, Flash drive.

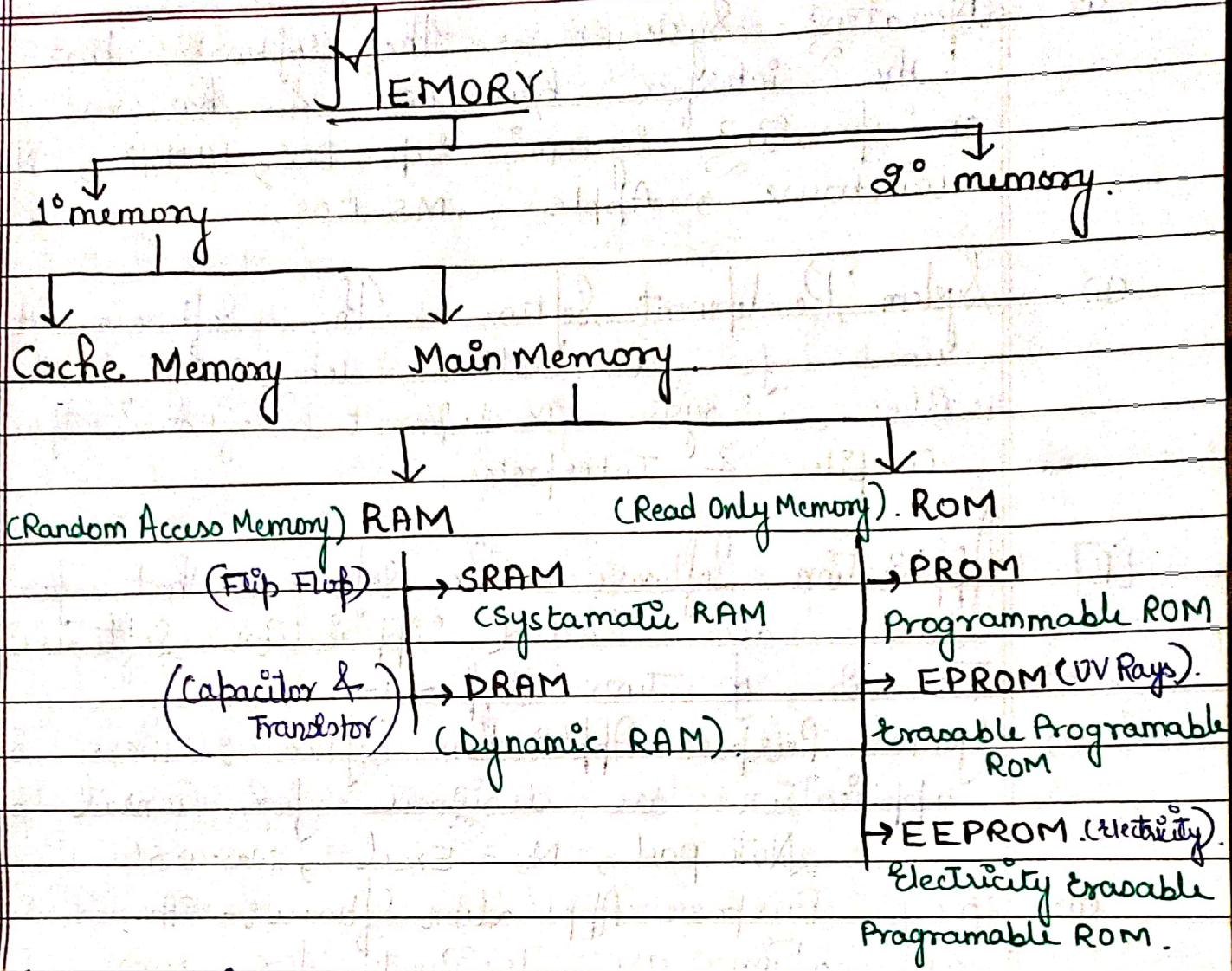
\* Output Devices :- An output device is any piece/part of an computer hardware equipment used to communicate the result of data by an information processing system. e.g. - Speaker, Projectors

Types of Output Device :-

1. CRT (Cathode Ray Tube)
2. LCD (Liquid Crystal Display)
3. LED (Light Emitting Diode)

\* Printers :- There are of following types :-

- (1) Laser Printer
- (2) Ink jet Printer
- (3) Dot matrix printer



\* Software - Software is the collection of programs or instructions that allow the hardware to do its task. This is of two types :-

1. System Software
2. Application Software.

[A]. System Software :- The software that supports the functionality of the comp system & manage the hardware resources & perform required processing task is called a system software.

→ It has two sub-types :-

- (1) Operating System
- (2) Software development is system based.

(i) **Operating System** :- The software that provides the interface b/w user & hardware is called an 'Operating System'. Ex:- DOS, LINUX, UNIX, Windows, Apple, MS-DOS.

(ii) **System Development Software** :- The software that is used for translation, debugging & so on is called a 'System Development Software' for ex:- Compiler & Interpreter.

[B] **Application Software** :- The software that facilitates the user are called 'Application Software'.

→ It is of two types. 1-

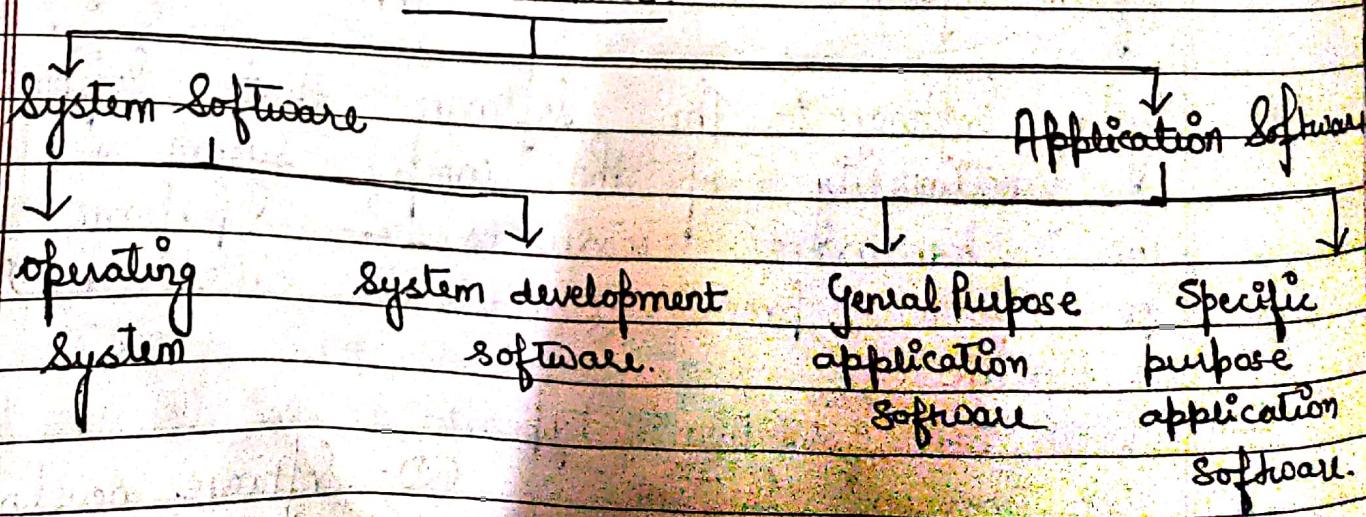
(i) **General Purpose Application Software** :- These software application are designed for General purpose. Notepad, MS-Excel, MS-Word

(ii) **Specific Purpose Application Software** :- These software application are designed for some specific purpose for ex:- Accounts Software & Software used for Material Managements



d

## SOFTWARE



\* Classification of the Computer :- The classification of comp. is done on the basis of the following two major categories.

(1) Size & Capacity & (2) Technology Use.

- \* → Based on its size & capacity :- There are four major types of comp. in this category
- 1. Super Comp → Most powerful, fastest comp system, it needs a large room.
- Used for complex scientific applications.
- High cost, able to handle large amount of data, high power consumption, large & fast memory (1<sup>o</sup> & 2<sup>o</sup> memory).
- Does multiprocessing & parallel processing & supports multiple programming.
- Ex:- Super Comp help in analysing weather pattern, light pollution, global warming & the depletion of the earth Ozone layers.
- 2. Main frame Comp - Able to process large amount of data at very high speed.
- Supports multi user facilities, no. of processor varies from 1 to 6.
- Helps in handling the process of various organizations such as Railways, Banks, hospitals & insurance companies.
- Supports many input & output & auxiliary devices & network of terminals.

Ex:- IBM-3000 Series, UNIBAC 1180 & DEC

- (3) Mini Computers :- Performs better than micro comp & medium size comp.
- Design to support more than one user at a time.
  - Posses large storage capacity & operators at higher speed.
  - for Ex:- Digital Equipment PDP 11/45 & VAX-11.

- (4) Micro Comp:- Uses a micro-processor as its CPU.
- Main difference b/w micro-comp & main frame or mini comp are that the micro comp have the smallest memory & less power, are physically smaller & permit fewer peripheral devices to be attached.
  - Ex:- Desktop Models, Note-book Comps or Laptop Comps, Network Comps are exp of micro comps.

- \* → Based On technology Use :- There are three major types of comp in this category.
- (1) Digital Comp :- The comp that performs the calculation & logical operation with quantities represented as 0 & 1 i.e. digits, a combination of 0 & 1 is a 'Digital Comp'.
- for Ex:- Personal Comp.

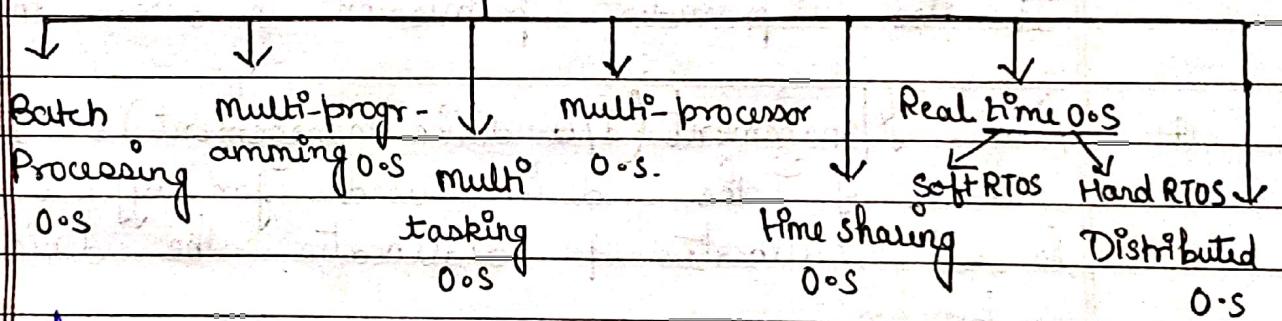
- (2) Analogue Comp:- The Comp that works on signal (by accepting temp, pressure, speed & so on by input) is an 'Analogue Comp'.

→ for expt- Speedometers, Thermometers & Voltmeters.

- (3). Hybrid Comps:- The Comp that accepts the both types of input i.e. signal & digits is a hybrid comp.  
 → for expt- ICUs

\* Operating System or operating system is a system software that provides an interface b/w the user & the hardware.

### OPERATING SYSTEM.



→ operating system works as a resource of a comp system & it also manage the responsible performing overall operation of the comp.

→ It provides services such as user Interface, file & data base access & interface to communication system such as internet protocol.

\* Function / Purpose / Goals Of Operating System  
 The O.S manages the various resources of a comp system such as the processor, memory, device & file.

- (1). Process Management :- The Os is responsible to allocate & de-allocate process to the processor (CPU). It keeps track of the process.
- (2). Memory Management :- The Os is responsible for allocation & deallocation of memory. It decides which data will store where. So, it keeps track record of memory as well.
- (3). File Management :- It keep track of files in use & the jobs using these file & maintenance of directory of their location. It also provides permission to access the files.
- (4). Device Management (I/O device management) :- It keeps track of the devices used for different purpose so, it allocates & deallocates the devices to a process.
- (5). Network & Communication :- It helps in networking & communication.
- (6). Security & Protection :- It provides security & protection to the system.
- \* → Types Of Operating System :- Operating System were classified into different bodies -

- (1). Batch Processing Os :- The Os system that automatically keeps executing to the one job to the

next job in a batch. for exp:- M-S-DOS

- (2). **Multi programming Os :-** The Os provides multiple jobs in main memory to be executed by the processor.

In Batch system, CPU sits idle during input & output operations, so the multiprogramming system allows the main processor to execute another process when the I/O processor is executing other process. for exp:- UNIX & Windows.

- (3). **Multitasking Os / Time sharing Os :-** Multitasking is an extension of multiprogramming Os. This job executes multiple jobs by the CPU switching b/w them.

It's provide a time slice to switch among diff. jobs. switching is done so frequently that each user thinks that the CPU is executing only his or her job for exp:- LINUX, UNIX & Windows

- (4). **Multiprocessor Os / Parallel Os :-** Multiprocessor Os refers to the use of two or more CPU's within a single comp system.

These multiple CPUs are in the close communication sharing the comp bus, memory & other peripheral devices.

These system are referred as tightly coupled system.

These type of system are used when very high speed is required to process a large amount of data.

These systems are generally used in environment like satellite control, weather forecasting & so on. exp: LINUX, UNIX

- (5). Real time Dos :- The real time Dos is used for a real time applicat<sup>n</sup> means for those applications where data processing should be done in the fixed & small quantum of time. RTOS is used as priority to execute the process when a high priority process enters the system. low priority process pribited to serve higher priority process.

RTOS synchronized the process, resource can be used efficiently with out wastage of time.

Types of RTOS.

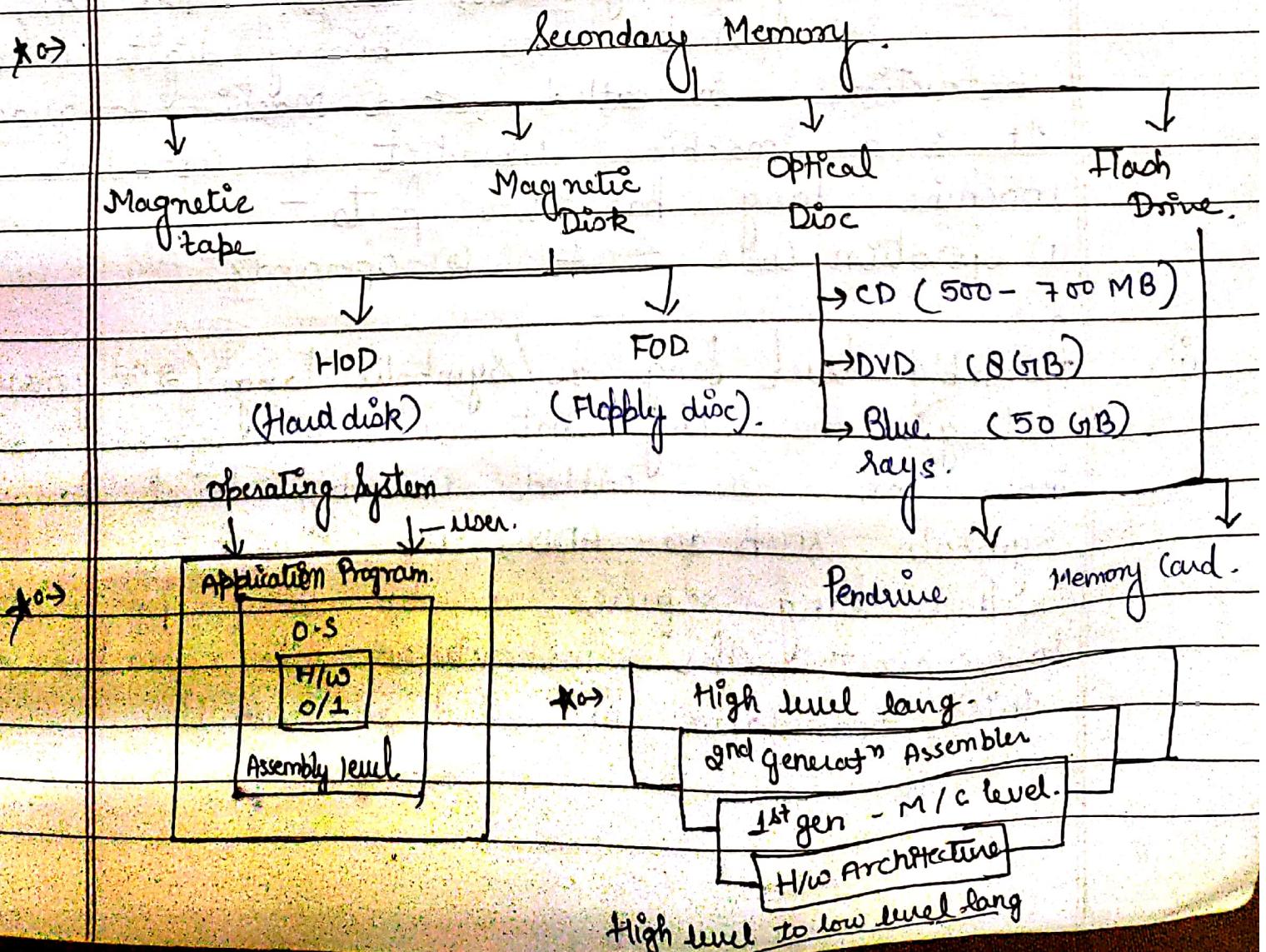
- (a). Soft RTOS :- A process might not be executed in even deadline. It can be crossed if they executed next, without harming the system. For exp:- Mobile phones, Digital camera & so on.

- (b). Hard RTOS :- A process should be executed in given deadline. The deadline shouldn't be crossed. preemption time. For hard RTOS is almost less than few micro seconds.

Expt- Air bag control in cars, Anti lock break, engine control system & so on.

(6) Distributed O.S / Usually Coupled O.S :- This system supports the multiple processors that have their own memory & clock. The processor communicates with one another through various communication line such as High speed buses or telephone lines.

(7) Multi-threading O.S :- A multithreading O.S allows the running of different parts of a program at the same time for exp:- UNIX, LINUX, windows etc.



## COMPUTER LANGUAGES

→ A programming languages consist of all characters, symbols & user rules that permit the user to communicate with the comp.

→ There are two major types of programming languages. (i) low level lang & (ii) High level lang.

[A]. Low level language:- This is the lang in which each statement or instructions is directly transmitted into machine code. It is of two types:-

(i) Machine lang (1<sup>st</sup> generation lang). The lang that uses binary digits in an instruction is called as a Machine language. It is a machine dependent & instruction in machine lang has two parts - (i) operation code & (ii) operands.

(ii) Assembly level language / Symbolic lang (2<sup>nd</sup> generation lang). The lang that uses symbols in an instruction is called Assembly level language. Symbols such as - ADD, SUB, MUL & DIV.

This lang requires a translator program to convert the instructions into machine code.

This translator is called as Assembler.

[B]. High level language :- In this type of lang, instructions are written in the form of 'English like statement'. It is of three types :-

i) Procedural Oriented lang :- (3<sup>rd</sup> generation)

This lang is design to facilitate the accurate description of procedure, algorithms & routines that belong to the set of instructions for ex:- C, ~~COBOL~~<sup>COBOL</sup> (Common Oriented business oriented lang), C++, Java.

ii) Problem Oriented lang. (4<sup>th</sup> generation lang). This

lang is used to solve specific problem these includes Query langs., report generators

& Application generator for ex. SQL

(Structure Query lang), MATLAB. (Matrix laboratory lang.) , LATEX

iii) Non-Procedural Languages. (5<sup>th</sup> Generation). These

langs are used in (Artificial Intelligence)

AI. AI means that the machine has the ability to think & work accordingly. for ex:-

LISP, ML & PROLOG.

\* Differences b/w High level lang, Assembly lang.  
4 Machine level lang

No.

CHL

HTGH level lang

Assembly level lang  
(and) JML

MACHINE level lang

Programs of those lang are easily understandable

Programs are less understandable  
than high level lang however it is more understandable than machine level lang.

Programs are portable & machine independent.

Programs are machine dependent.

In this lang debugging is easier

Debugging is more complex than HTL

It is most suitable for software development

It also not suitable.

Program is translated using compiler or interpreter

Program is translated using assembler into machine code.

No translation is required.

# TRANSLATOR Tools.

Translator is a software or programme that converts one lang into another.  
Some of the tools used for translation process are given as follows:-

1. Pre-Processor :- Pre-Processor is a translation programme that process the source code before compile time & sought out all the pre-processor directives.
2. Compiler :- A compiler is a translator program that reads the program written in ~~HLL~~ one lang (HLL) & translates it into target code i.e a (MLL)
3. Interpreter :- An Interpreter is also a translator program, it converts the source code into line by line into target code, so, it can't execute the whole code immediately.
4. Assembler :- Assembler is a translator program that translates assembly level lang or symbolic lang into machine lang.
5. Linker :- A linker is a system software that combine different object modules into a

common file, i.e. an executable file.  
The executable file is stored in a secondary memory.

6. Loader :- Loader is a program routine that copies a program executable file into main memory.

## \* Differences b/w a Compiler & a Interpreter



- (1) A compiler is a translator which transforms the source lang (HLL) into object lang (ML).
- (2) Compiler converts the whole program in one go.
- (3) It generates intermediate object code.
- (4) Compilation is done before execution.
- (5) Its speed is comparatively faster.
- (6) memory requirement is more due to the creation of object code.
- An interpreter is a program which imitates the execution of programs written in a source lang.
- Interpreter converts the program by taking a single line at a time.
- It doesn't produce any intermediate object code.
- Compilation & execution takes place simultaneously.
- Its speed is slower compared to compiler.
- It requires less memory as it doesn't require creating intermediate object code.

## Number System!:-

①

The computer are made up of digital electronic circuits. The major elements being transistors which are essential bistable or binary in nature. The term binary means that the transistor can operate only two states (0's and 1's). Fully conducting states and non-conducting states: thus the a, transistor acts like a switch i.e., it is either on or off.

Since the computer is made up of binary logic circuits, data is also represented in a computer by either the presence or absence of electronic "signals". Thus it's called binary or two state representation of data. The number system is used to represent data in a computer is therefore known as binary number system.

Number System: There are four types of number system.

- (1) Decimal Number System
- (2) Binary Number System
- (3) Octal Number System
- (4) Hexadecimal Number System.

### ① Decimal Number System:-

decimal system having digit value 0 through 9 with 0, 1, 2, 3, ..., 9 as 10 symbols. Decimal number system has base = 10. The absolute value of each digit is fixed but its positional value is determined by its position in the whole number.

Example:- 3958

Number	Symbol value	position from the right hand	Position value	Decimal Equivalent
<u>3958</u>				
3958	8	0	$10^0$	$8 \times 10^0 = 8$
	5	1	$10^1$	$5 \times 10^1 = 50$
	9	2	$10^2$	$9 \times 10^2 = 900$
	3	3	$10^3$	$3 \times 10^3 = 3000$
				<u>3958</u>

## 2) Binary Number System:-

Binary Number system has base-2. It consists of 2 digits as 0 and 1.

Example:-  $(110110110)_2$

Number	Symbol value	position from the right hand	Position value	Decimal equivalent
110110110	0	0	$2^0$	$0 \times 2^0 = 0$
	1	1	$2^1$	$1 \times 2^1 = 2$
	0	2	$2^2$	$1 \times 2^2 = 4$
	1	3	$2^3$	$0 \times 2^3 = 0$
	1	4	$2^4$	$1 \times 2^4 = 16$
	0	5	$2^5$	$1 \times 2^5 = 32$
	1	6	$2^6$	$0 \times 2^6 = 0$
	0	7	$2^7$	$1 \times 2^7 = 128$
	1	8	$2^8$	$1 \times 2^8 = 256$
				$(438)_{10}$

## 3) Octal Number System:-

Octal Number system has eight symbols as 0, 1, 2, 3, 4, 5, 6, 7. It has base-8. These digits have exactly same physical meaning as decimal system. Positional value of different digits is given by different power of 8.

Example:- 7651

Number	Symbol value	position value from the right hand	Position value	Decimal equivalent
7651	1	0	$8^0$	$1 \times 8^0 = 1$
	5	1	$8^1$	$5 \times 8^1 = 40$
	6	2	$8^2$	$6 \times 8^2 = 384$
	7	3	$8^3$	$7 \times 8^3 = 3584$
				$(4009)_{10}$

## 4) Hexadecimal Number System:-

Hexadecimal Number system has base-16 and it uses sixteen distinct digits but 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9. and A-10, B-11, C-12, D-13, E-14-F-15

Example:- 9AC2

Number	Symbol value	position from the right hand	Positional value	Decimal equivalent
9AC2	2	0	$16^0$	$2 \times 16^0 = 2$
	C	1	$16^1$	$12 \times 16^1 = 192$
	A	2	$16^2$	$10 \times 16^2 = 2560$
	9	3	$16^3$	$9 \times 16^3 = 36864$
				$(36864)_{10}$

(3)

## Conversion:-

- (1) Binary to Decimal
- (2) Decimal to binary
- (3) Octal to decimal
- (4) Decimal to octal
- (5) Hexadecimal to decimal
- (6) Decimal to Hexadecimal
- (7) Binary to Octal
- (8) Octal to binary
- (9) Binary to Hexadecimal
- (10) Hexadecimal to binary
- (11) Octal to Hexadecimal
- (12) Hexadecimal to octal

### (1) Binary to Decimal:-

Example:- Convert  $(1100010110)_2$  to its equivalent decimal number system.

Sol:-

$$(1100010110)_2 = (790)_{10}$$

1<sup>st</sup> Method:

$$\begin{array}{r}
 1100010110 \\
 | | | | | | \\
 0 \times 2^0 = 0 \\
 1 \times 2^1 = 2 \\
 1 \times 2^2 = 4 \\
 0 \times 2^3 = 0 \\
 1 \times 2^4 = 8 \\
 0 \times 2^5 = 0 \\
 0 \times 2^6 = 0 \\
 0 \times 2^7 = 0 \\
 1 \times 2^8 = 256 \\
 1 \times 2^9 = 512 \\
 \hline
 (790)_{10}
 \end{array}$$

2<sup>nd</sup> Method

$$\begin{array}{cccccccccc}
 2^9 & 2^8 & 2^7 & 2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\
 | & | & | & | & | & | & | & | & | & | \\
 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0
 \end{array}$$

$$\begin{aligned}
 &= 512 + 256 + 8 + 4 + 2 \\
 &= (790)_{10}
 \end{aligned}$$

(2) Find the decimal number of  $(11010.0110)_2$

④

$$\begin{array}{r} 2^5 \ 2^4 \ 2^3 \ 2^2 \ 2^1 \ 2^0 \\ \hline 1 \ 1 \ 0 \ 1 \ 0 \cdot 2^1 \ 2^2 \ 2^3 \ 2^4 \\ \end{array}$$

$11010$

$$\begin{array}{l} 0 \times 2^0 = 0 \\ 1 \times 2^1 = 2 \\ 0 \times 2^2 = 0 \\ 1 \times 2^3 = 8 \\ 1 \times 2^4 = 16 \end{array}$$

$$\overline{(26.375)}_{10}$$

$$16 + 8 + 2 = 26 \cdot 0 + 0.25 + 0.125 = 0$$

$0110$

$$\begin{array}{l} 0 = 0 \times 2^{-1} \\ 0.25 = 1 \times 2^{-2} \\ 0.125 = 1 \times 2^{-3} \\ 0 = 0 \times 2^{-4} \end{array}$$

Example:- ①  $(10110.11)_2 = (?)_{10}$

② Convert the binary number  $(110101110)_2$  to its decimal equivalent

$$\text{Ans: } (430)_{10}$$

③ Convert the fractional binary number  $(1101.01101)_2$  to its equivalent decimal number. Ans  $(13.40625)_{10}$

(2) Decimal to binary-

① Convert the decimal number  $(45)_{10}$  to its binary number

Method 1:-

$$\begin{array}{r} 2 | 45 \\ 2 | 22 \ 1 \\ 2 | 11 \ 0 \\ 2 | 5 \ 1 \\ 2 | 2 \ 1 \\ 2 | 1 \ 0 \\ \hline 0 \ 1 \end{array}$$

↑ LSB (Least significant bit or digit)

↑ MSB (Most significant bit or digits)

$$(45)_{10} = (101101)_2$$

2nd method

$$\begin{array}{r} 2^5 \ 2^4 \ 2^3 \ 2^2 \ 2^1 \ 2^0 \\ \hline 32 \ 16 \ 8 \ 4 \ 2 \ 1 \end{array}$$

$$(45)_{10} = (101101)_2$$

(2) Convert the decimal number  $(60.90)_{10}$  to binary number

(5)

1st Method:

2	60
2	30 0
2	15 0
2	7 1
2	3 1
2	1 1
1	0 1

$$\begin{aligned}
 & \cdot 40 \times 2 = 0.8 \quad \text{---} 0 \\
 & \cdot 8 \times 2 = 0.6 \quad \text{---} 1 \\
 & \cdot 6 \times 2 = 0.2 \quad \text{---} 1 \\
 & \cdot 2 \times 2 = 0.4 \quad \text{---} 0 \\
 & \cdot 4 \times 2 = 0.8 \quad \text{---} 0
 \end{aligned}$$

$(01100\cdots)_2$

$$60 \quad (111100)_2$$

$$(60.90)_{10} = (111100.01100\cdots)_2$$

(3)  $(24.625)_{10} = (?)_2$

2	24
2	12 0
2	6 0
2	3 0
2	1 1
1	0 1

$$\begin{array}{r}
 \begin{array}{cccccc}
 2^4 & 2^3 & 2^2 & 2^1 & 2^0 & \cdot 625 \\
 16 & 8 & 4 & 2 & 1 & \times 2 \\
 1 & 1 & 0 & 0 & 0 & \leftarrow 1.25 \\
 & & & & & \times 2 \\
 & & & & 0 & \leftarrow 0.5 \\
 & & & & \times 2 & \leftarrow 1.000 \\
 & & & & 1 & \leftarrow 1.000
 \end{array}
 \\[10pt]
 (24.625)_{10} = (11000.101)_2
 \end{array}$$

Example Convert decimal number  $(17)_{10}$  to its equivalent binary number

Ans  $(10001)_2$

(2)  $(0.8125)_{10} = (?)_2$  = Ans  $(0.1101)_2$

(3)  $(16.725)_{10} = (?)_2 = (10000.1011\cdots)_2$

(3) Octal to decimal conversion:-

Convert the octal number  $(176)_8$  to its decimal number

Sol:  $(176)_8$

$$\begin{aligned}
 &= 176 \\
 &\quad \begin{array}{l}
 6 \times 8^0 = 6 \\
 7 \times 8^1 = 56 \\
 1 \times 8^2 = 64
 \end{array} \\
 &\quad \hline
 & (126)_{10}
 \end{aligned}$$

(6)

Q) Convert the octal number  $(167.45)_8$  to its decimal number

$$\begin{array}{r}
 167 \\
 \left[ \begin{array}{l} 7 \times 8^0 = 7 \\ 6 \times 8^1 = 48 \\ 1 \times 8^2 = 64 \end{array} \right] \\
 \hline 119
 \end{array}$$

$$\begin{array}{r}
 45 \\
 \left[ \begin{array}{l} 4 \times 8^{-1} = 0.5 \\ 5 \times 8^{-2} = 0.078125 \end{array} \right] \\
 \hline 0.578125
 \end{array}$$

$$(167.45)_8 = (119.578125)_10$$

Ans

$$\textcircled{3} \quad (27.64)_8 = (\ )_{10} = (23.8125)_{10}$$

$$\textcircled{4} \quad (162)_8 = (\ )_{10} = (114)_{10}$$

$$\textcircled{5} \quad (177.670)_8 = (\ )_{10} = (127.85938)_{10}$$

(4) Decimal to octal :-① convert  $(961)_{10}$  to its equivalent octal numbers

Sol:

$$\begin{array}{r}
 961 \\
 \hline
 8 | 120 | 1 \\
 8 | 15 | 0 \\
 8 | 1 | 7 \\
 \hline
 0 | & 1
 \end{array} = (1701)_8$$

$$\textcircled{2} \quad (506.61)_{10} = (\ )_8$$

$$\begin{array}{r}
 506 \\
 \hline
 8 | 63 | 2 \\
 8 | 7 | 7 \\
 \hline
 0 | 7
 \end{array} = (772)_8$$

$$\begin{array}{r}
 61 \\
 \left[ \begin{array}{l} 6 \times 8^{-1} = 0.75 \\ 1 \times 8^{-2} = 0.015625 \end{array} \right] \\
 \hline 0.765625
 \end{array}$$

$$= (772.4702 \dots)_8$$

$$\begin{array}{r}
 61 \\
 \times 8 \\
 \hline
 488 \\
 -8 \\
 \hline
 04 \\
 -8 \\
 \hline
 32 \\
 -8 \\
 \hline
 56 \\
 -8 \\
 \hline
 48
 \end{array}$$

(7)

$$\textcircled{1} (19 \cdot 125)_{10} = (\quad)_8$$

$$\textcircled{2} (140)_{10} = (\quad)_8$$

\textcircled{5} Binary to octal

\textcircled{1}  $(10010110)_2$  convert it into octal number

$$\begin{array}{r} \stackrel{2}{\overline{010010}} \underline{110} \\ \underline{2} \qquad \underline{6} \end{array} = (226)_8$$

\textcircled{2}  $(110101011 \cdot 0110110)_2$  into  $(\quad)_8 = (\quad)_{10}$

$$\begin{array}{r} \stackrel{5}{\overline{110}} \underline{101011} \cdot \stackrel{3}{\overline{011}} \underline{011000} \\ \underline{6} \qquad \underline{3} \qquad \underline{0} \end{array}$$

$$(653 \cdot 330)_8$$

$$\textcircled{3} (10111100 \cdot 1000010)_2 = (574 \cdot 210)_8$$

$$\textcircled{4} (01100101 \cdot 11)_2 = (145 \cdot 6)_8$$

$$\textcircled{5} (110101110)_2 = (656)_8$$

\textcircled{6} Octal to binary:

\textcircled{1} convert the octal number  $(671 \cdot 54)_8$  into a binary number

$$671 \cdot 54$$

$$\begin{array}{r} \stackrel{6}{\overline{110}} \stackrel{7}{\overline{111}} \stackrel{1}{\overline{001}} \cdot \stackrel{5}{\overline{101}} \stackrel{4}{\overline{100}} \\ \underline{110} \qquad \underline{111} \qquad \underline{001} \qquad \underline{101} \qquad \underline{100} \end{array}$$

$$(110111001 \cdot 101100)_2 = (\quad)_{10}$$

$$\textcircled{2} (765 \cdot 727)_8 = (\quad)_2$$

$$\begin{array}{r} \stackrel{7}{\overline{111}} \stackrel{6}{\overline{110}} \stackrel{5}{\overline{101}} \cdot \stackrel{7}{\overline{111}} \stackrel{2}{\overline{010}} \stackrel{7}{\overline{111}} \\ \underline{111} \qquad \underline{110} \qquad \underline{101} \qquad \underline{111} \qquad \underline{010} \qquad \underline{111} \end{array}$$

$$(111110101 \cdot 111010111)_2$$

(8)

$$③ (572.11)_8 = (101111010.001001)_2$$

$$④ (1101.10)_8 = (001001000001.001000)_2$$

⑦ Hexadecimal to decimal:-

$$① (A9DB)_{16} = (?)_{10} = (43480)_{10}$$

A 9 D 8

$$\begin{array}{r} 8 \times 16^0 = 8 \\ 13 \times 16^1 = 208 \\ 9 \times 16^2 = 2304 \\ 10 \times 16^3 = 40960 \\ \hline (43480)_{10} \end{array}$$

$$② (9AC6.27A)_{16} = (?)_{10}$$

$$\begin{array}{r} 9 A C G \\ 6 \times 16^0 = 6 \\ 12 \times 16^1 = 192 \\ 10 \times 16^2 = 2560 \\ 9 \times 16^3 = 36864 \\ \hline 39622 \end{array}$$

• 27A

$$\begin{array}{r} 4 \\ 2 \times 16^{-1} = \\ 7 \times 16^{-2} \\ 10 \times 16^{-3} \end{array}$$

$$0.125 + 0.02734375 \\ + 0.0024414$$

$$(39622.155)_{10}$$

$$③ (8C.35)_{16} = (?)_{10} = \text{Ans} (140.20703125)_{10}$$

$$④ (\text{ACB.C9})_{16} = (?)_{10} = (?)_2 = (2760.7852)_{10}$$

$$⑤ (A9D)_{16} = (?)_{10} = (2717)_{10}$$

⑧ Decimal to Hexadecimal:-

① Convert  $(961)_{10}$  to its hexadecimal number.

16	961	
16	60	1
16	3	12-C
	0	3

$$= (3C1)_{16}$$

(9)

$$\textcircled{2} (30.875)_{10} = (\quad)_{16}$$

$$\begin{array}{r} \cdot 875 \\ \times 16 \\ \hline 14 \cdot 000 \\ \hline 1 \\ \hline E \end{array}$$

$$\begin{array}{r} 16 | 30 \\ \hline 16 | 1 \quad 14 - E \\ \hline 0 \quad 1 \end{array}$$

$$(1E.E)_{16}$$

$$\textcircled{3} (961.12)_{10} = (\quad)_{16} = (3C1.1EB)_{16}$$

$$\textcircled{4} (87)_{10} = (\quad)_{16} = (57)_{16}$$

$$\textcircled{5} (140)_{10} = (\quad)_{16} = (8C)_{16}$$

(9) Binary to Hexadecimal :- Hexadecimal to Binary conversion

(1) Convert the hexadecimal number  $(A2B)_{16}$  into the binary number

$$\begin{array}{r} A \quad 2 \quad B \\ \hline 1010 \quad 0010 \quad 1011 \\ \hline \end{array} (101000101011)_2$$

$$\textcircled{2} (2CB \cdot AD)_{16}$$

$$\begin{array}{r} 2 \quad C \quad B \cdot \quad A \quad D \\ \hline 0010 \quad 1100 \quad 1011 \quad 1010 \quad 1101 \\ \hline \end{array} (001011001011 \cdot 10101101)_2$$

$$\textcircled{3} (A5B \cdot 17)_{16} = (101001011011 \cdot 00010111)_2$$

$$\textcircled{4} (10 \cdot 11)_{16} = (00010000 \cdot 00010001)_2$$

(10) Binary to Hexadecimal:

$$\textcircled{1} (10111111 \cdot 111)_{2}$$

$$\begin{array}{r} 1011 \quad 1111 \cdot \quad 1110 \\ \hline B \quad E \quad 1 \\ \hline \end{array} = (BF \cdot E)_{16}$$

$$\textcircled{2} (010100 \cdot 101000)_2 = \underbrace{00010100}_{4} \cdot \underbrace{10100000}_A = (14 \cdot A0)_{16}$$

(10)

(1) Octal to Hexadecimal to Octal

1<sup>st</sup> method

$$(6E)_{10} = (156)_8 \quad \begin{array}{r} 5 \\ 001101110 \\ 1 \quad 6 \end{array}$$

1<sup>st</sup> method2<sup>nd</sup> Method

6 E

$$\begin{array}{l} 14 \times 16^0 = 14 \\ 6 \times 16^1 = 96 \\ \hline 110 \end{array}$$

2<sup>nd</sup> Method

$$\begin{array}{r} 8 | 110 \\ 8 | 23 \quad 6 \\ 8 | 1 \quad 5 \\ \hline 0 \quad 11 \end{array}$$

$$= (156)_8$$

(2) ~~(BFA · 3)~~  $(BFA \cdot 3)_{16} = (?)_8$

1<sup>st</sup> method

$$\begin{array}{r} B F A \\ \hline 101111111010 \\ 5772 \end{array} \quad \begin{array}{r} 3 \\ 001100 \\ 1 \quad 4 \end{array}$$

$$110 \overline{)101010}$$

$$(5772 \cdot 14)_8$$

2<sup>nd</sup> method

B F A

$$\begin{array}{l} 10 \times 16^0 = 10 \\ 15 \times 16^1 = 240 \\ 11 \times 16^2 = 2816 \\ \hline (3066 \cdot 188) \end{array}$$

$$\begin{array}{r} 3 \\ 3 \times 16^{-1} = 0.1875 \end{array}$$

$$\begin{array}{r} 8 | 3066 \\ 8 | 383 \quad 2 \\ 8 | 47 \quad 7 \\ 8 | 5 \quad 7 \\ \hline 0 \quad 5 \end{array}$$

$$\begin{array}{r} .188 \\ \times 8 \\ \hline .504 \\ \times 8 \\ \hline .032 \\ \times 8 \\ \hline .256 \\ \times 8 \\ \hline .48 \end{array}$$

$$(5772 \cdot 14)_2$$

(11)

(12) Octal to Hexadecimal:

$$① (457.24)_8 = (?)_{16}$$

$$\begin{array}{r} 4 \quad 5 \quad 7 \\ \hline 000 \overline{100} \quad \overline{101} \quad \overline{111} \\ \hline 1 \quad 2 \quad F \end{array} \quad \begin{array}{r} 2 \quad 4 \\ \hline 010 \quad \overline{10000} \\ \hline 5 \quad 0 \end{array}$$

$$(12F.50)_{16}$$

$$\begin{array}{r} 00 \quad \overline{101} \quad \overline{110} \\ \hline 2 \quad E \end{array}$$

$$② (56)_8$$

$$\begin{array}{r} 5 \quad 6 \\ \hline 00 \quad \overline{101} \quad \overline{110} \\ \hline 2 \quad E \end{array}$$

$$(2E)_{16}$$

2<sup>nd</sup> Method:

$$\begin{array}{r} 56 \\ \hline 6 \times 8^0 = 6 \\ 5 \times 8^1 = 40 \\ \hline (46)_{10} \end{array}$$

$$\begin{array}{r} 16 \bigg| 46 \\ \hline 16 \bigg| 2 \quad | \quad E \quad (14) \\ \hline 0 \quad | \quad 2 \end{array} \quad (2E)_{16}$$

II<sup>nd</sup> Method  $\rightarrow$  6 E

$$\begin{array}{r} 2 \mid 240 \\ \downarrow 14 \times 16 = 14 \text{ Ans} \\ \downarrow 6 \times 16^1 = 96 \end{array}$$

$$(110)_{10}$$

$$\begin{array}{r} 8 \mid 110 \\ 8 \mid 13 \quad 6 \\ 8 \mid 1 \quad 5 \\ \hline 0 \quad 1 \end{array}$$

$$(156)_8 \text{ Ans.}$$

Ques:- Convert  $(BFA \cdot 3)_{16} = (?)_8$ .

$$\begin{array}{ccccccccc} & 8+2 & 8+2 & 8+2 & 8+2 & 8+2 & 8+2 & 8+2 & 8+2 \\ & B & F & A & \cdot & 3 & & & \\ 1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 \\ \uparrow & \uparrow \\ 5 & 7 & 7 & 2 & \cdot & 1 & 4 & & \end{array}$$

$$(5772 \cdot 14)_8 \text{ Ans.}$$

## \* ARITHMETIC OPERATIONS —

→ Binary Arithmetic Operations :- Digital Computer is capable of performing arithmetic operation on binary no.

(1). Binary Addition :- The addition to binary numbers can be performed by following 1 rules-

Operation	Result
$0+0$	$0$
$0+1$	$1$
$1+0$	$1$
$1+1$	$0$ , with carry $1$

Ques:-

Add  $(1101)_2$  &  $(101)_2$ .

$$\begin{array}{r} 1101 \\ + 101 \\ \hline 10010 \end{array}$$

$$(13) \quad (5)$$

$$10010 \text{ Ans.}$$

Ques:-

Add  $(10101)_2$  &  $(11011)_2$ .

$$\begin{array}{r} 10101 \\ + 11011 \\ \hline 110000 \end{array}$$

$$110000$$

$$0 \text{ Ans.}$$

(2). Binary Subtraction :- The subtraction of binary no. can be performed by following the rule of binary subtraction.

Operation	Result
$0-0$	$0$
$0-1$	$1$ , with borrow 1
$1-0$	$1$ with borrow 1
$1-1$	$0$

Ques:-

Subtract  $(110)_2$  &  $(10)_2$ .

$$\begin{array}{r} 110 \\ - 10 \\ \hline 100 \end{array}$$

$$100$$

$$0 \text{ Ans.}$$

$$\begin{array}{r} 10101 \\ - 11011 \\ \hline 0 \end{array}$$

Date: \_\_\_\_\_ Page: \_\_\_\_\_

Ques:- Subtract  $(10101)_2$  &  $(11011)_2$ .

$$\begin{array}{r} 10101 \\ - 11011 \\ \hline 111010 \text{ Ans.} \end{array}$$

(3). Binary Multiplication :-

$$\begin{array}{r} 110 \\ * \underline{111} \\ 1110 \\ 110x \\ 110xx \\ \hline 100010 \text{ Ans.} \end{array}$$

(4). Binary Division.

$$\begin{array}{r} 1011 \\ 100) \underline{101101} \\ 100 \downarrow \\ 00110 \\ 100 \downarrow \\ 0101 \\ 100 \\ \hline 001. \end{array}$$

$$\begin{array}{r} 4 ) 45 ( 11 \\ 4 \downarrow \\ 05 \\ \underline{4} \\ 1. \end{array}$$

1011

101) 111011 (

101 ↓ ↓

01001

101 ↓

01001

101

100

\* Other operation are as follows:-

(1). One's Compliment :- It is a binary no., which is represented by changing zeros to one's & one's to zeros. exp-  
 $(10111010)_2 = (01000101)_2$ .

(2) Two's Compliment :- It is obtained by adding 1 in one's compliment of the binary no. exp-  $(10111010)_2 = (01000100)_2$

(3). Nine's Compliment :- Nine's Compliment of a decimal no. is obtained by subtracting it's each individual digit from 9. exp-  
 $(245)_{10} = (754)_{10}$

(4) Ten's Compliment :- Add 1 to 9's compliment to find 10's compliment. exp  $(245)_{10} = (755)_{10}$

\* Algorithms :- These are the sequence of logical steps required to perform a specific task of solving problems.

The algorithms help us to define & refine out programming problems or algorithms can be defined as a step by step procedure to solve a problem.

\* → Characteristics of Algorithms or properties of algorithms.

- (1) Input
- (2) Output
- (3) Definiteness
- (4) Effectiveness
- (5). Finiteness

o → The following are major characteristic properties of an algorithm.

- (1) Input :- every algorithm must accept some inputs or zero or more inputs
- (2) Output :- An algorithm must produce atleast one output
- (3) Definiteness :- task instructions must be declined well defined & precise there should not be any ambiguity.

(4) Effectiveness: An algorithm must be effective i.e. it must be simple & carry out all the operations with less complexity.

(5) Finiteness: An algorithm must contain finite sequence of instruction i.e. it must terminate after a fixed time.

## \* Types Or Approaches of Algorithms.

- (1) Divide & Conquer Algorithms.
- (2) Dynamic Programming.
- (3) Greedy Approach.
- (4) Branch & Bound Algorithm.
- (5) Approximation Algorithms.
- (6) Randomised Algorithms.

Ques:- Write an algorithm to make a tea.

- (1) Put the Tea-bag in a cup.
- (2) Fill the kettle with water.
- (3) Boil the water in the kettle.
- (4) Pour some of the boiled water into the cup.
- (5) Add milk to the cup.
- (6) Add sugar to the cup.
- (7) Stir the tea.
- (8) Drink the tea.

Ques:- Write an algorithm to find the addition of two no.

Step 1 :- START

Step 2 :- PRINT OR WRITE (Enter any two no.)

Step 3 :- READ OR INPUT a, b

Step 4 :- SUM = a + b.

Step 5 :- PRINT "Sum" = SUM.

Step 6 :- STOP.

OR.

① Start

② Print or write (first no. say a)

③ Print or write second no. say b

④  $c = a + b$

⑤ Print c

⑥ Stop.

Ques:- Write an algorithm To find addition product of two nos.

① Start

② Print or write (Enter any two no.)

③ Read or input (a, b).

④  $\text{Sum} = a+b + \text{Product} = a * b$ .

⑤ Print "Sum" = Sum

⑥ Print "Product" = Product

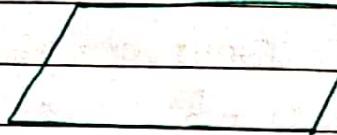
⑦ Stop.

## \* Flow chart.



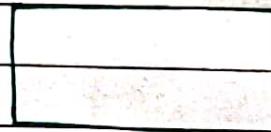
Oval shape.

(start/end).

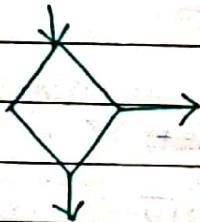


Parallelogram.

(Input/Output) symbol.



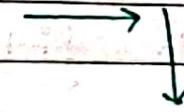
Rectangle (Process Box)



Diamond (Decision box)



Predefined Process Symbol.



Arrow symbol.

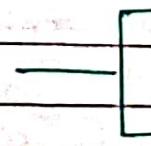
Flow lines / Arrow heads.

Page on



↑ Page off

Page off connector

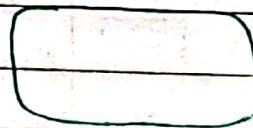
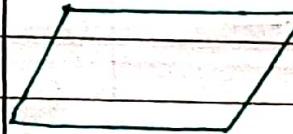
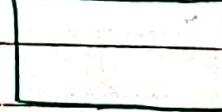
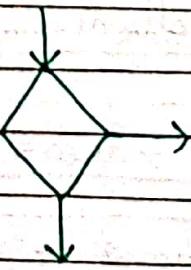


Annotation or Comment symbol.

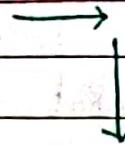
- o A flow chart is a pictorial / diagram representation of the sequence of operations necessary to solve the problem. The program flow chart shows as a sequential sequence of instructions in a program.

or a sub-routine.

The various symbol used to draw a flow chart. is given above below.

S.No.	Shape	Symbol	Name	Purpose & Meaning
1.	Oval		Terminal (start)/ Stop (end)	It is used to indicate the beginning/start & end of a program.
2.	Parallelogram		Input/ Output symbol	It show any statement that causes data to be read or print from a program.
3.	Rectangle		Process symbol	It is used for calculation or assignment of the values.
4.	Diamond		Decision box	It is used to take decisions in a program.
5.			Predifined process symbol	It is used to represent a group of statement that together accomplishes one task.

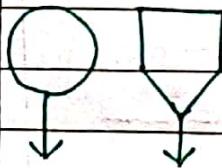
6.



flow lines  
arrow heads.

It is used to connect symbols & indicate the sequence of operations.

7.



Connector.

Page on Connector It's use to  
Page off Connector indicate that one symbol is connected to another.

8.



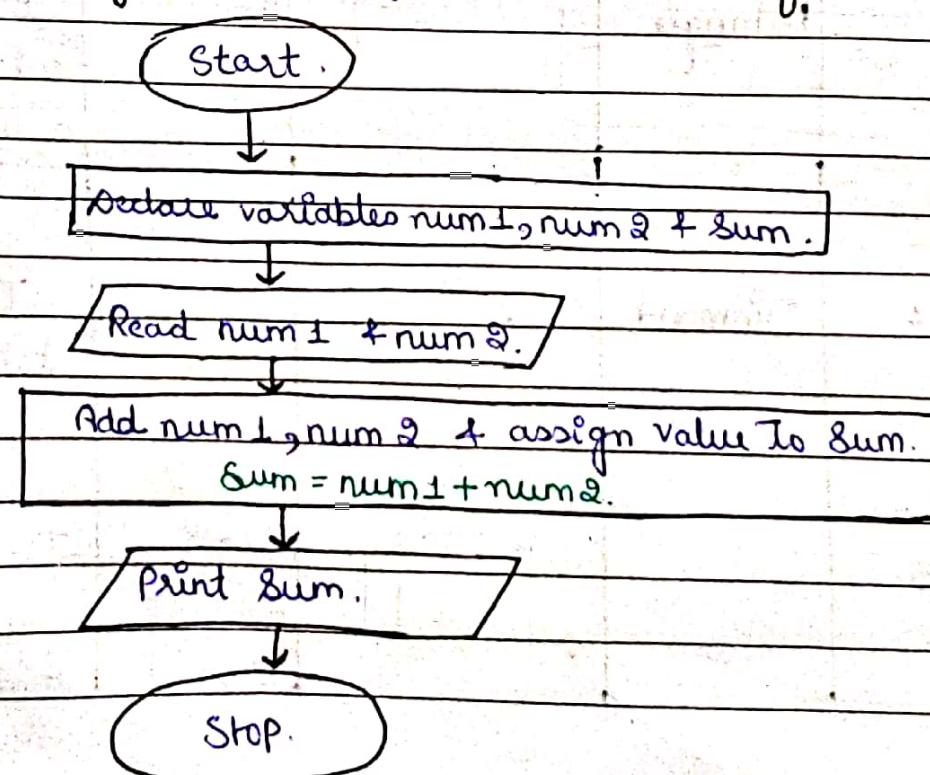
Annotation

or

Comment  
symbol.

It can be used to give comments.

Ques- with the help of flow chart diag write the algorithm of addition of two no.

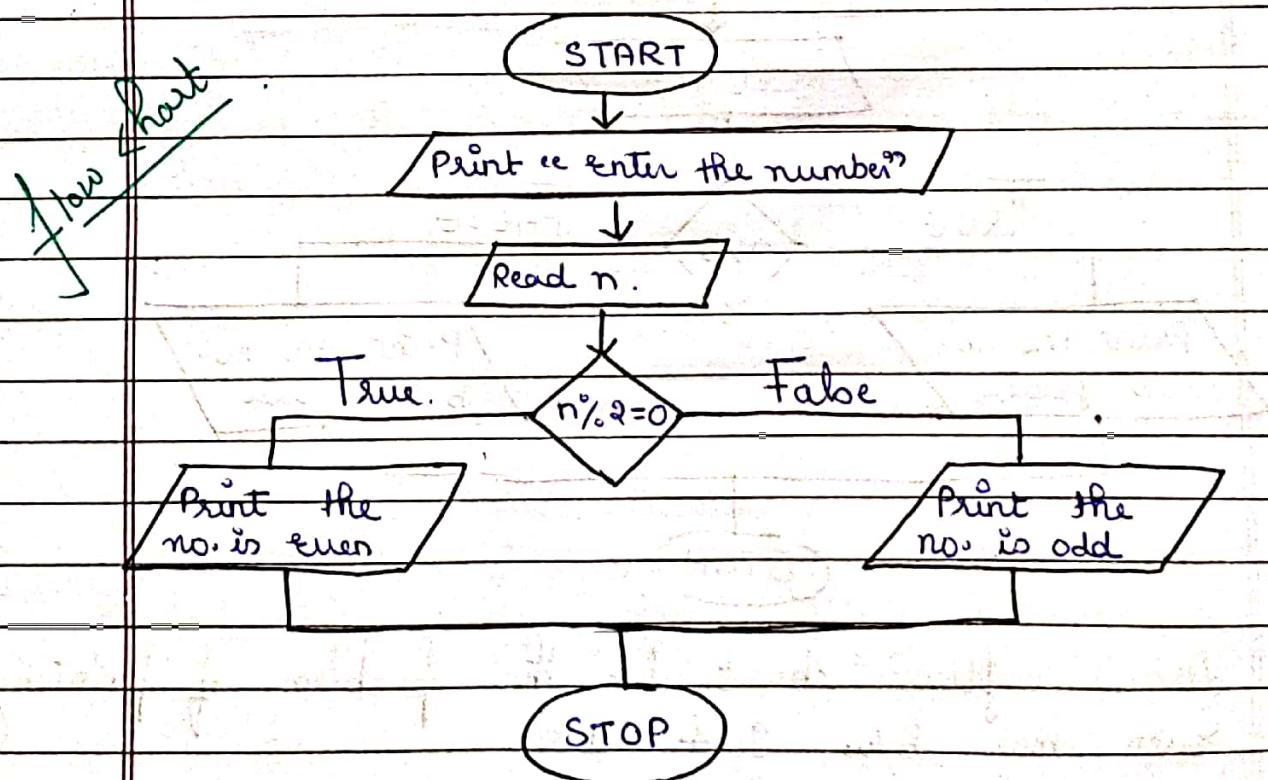


Quest-

Write an algorithm to check the no. is even or odd ~~depends on~~ + draw the flow chart.

Solt<sup>n</sup>:-

- ① Start.
- ② Print / Write "Enter the no." n.
- ③ Input or Read n
- ④ If  $(n \% 2) = 0$
- ⑤ Print "The no. is even".
- ⑥ Else
- ⑦ Print "The no. is odd".
- ⑧ End of if structure as step 4.
- ⑨ Stop.

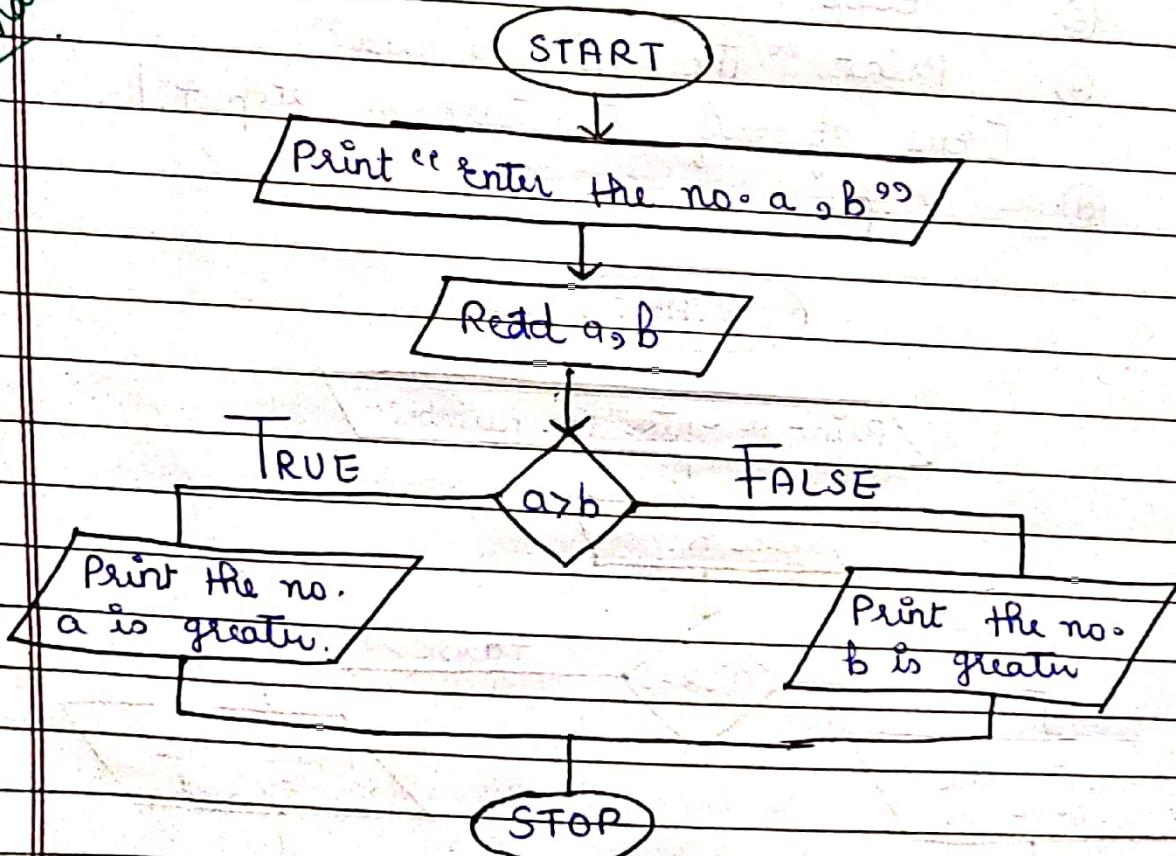


Quest

Write an algorithm to find the large no. among the two no. + draw the flow chart.

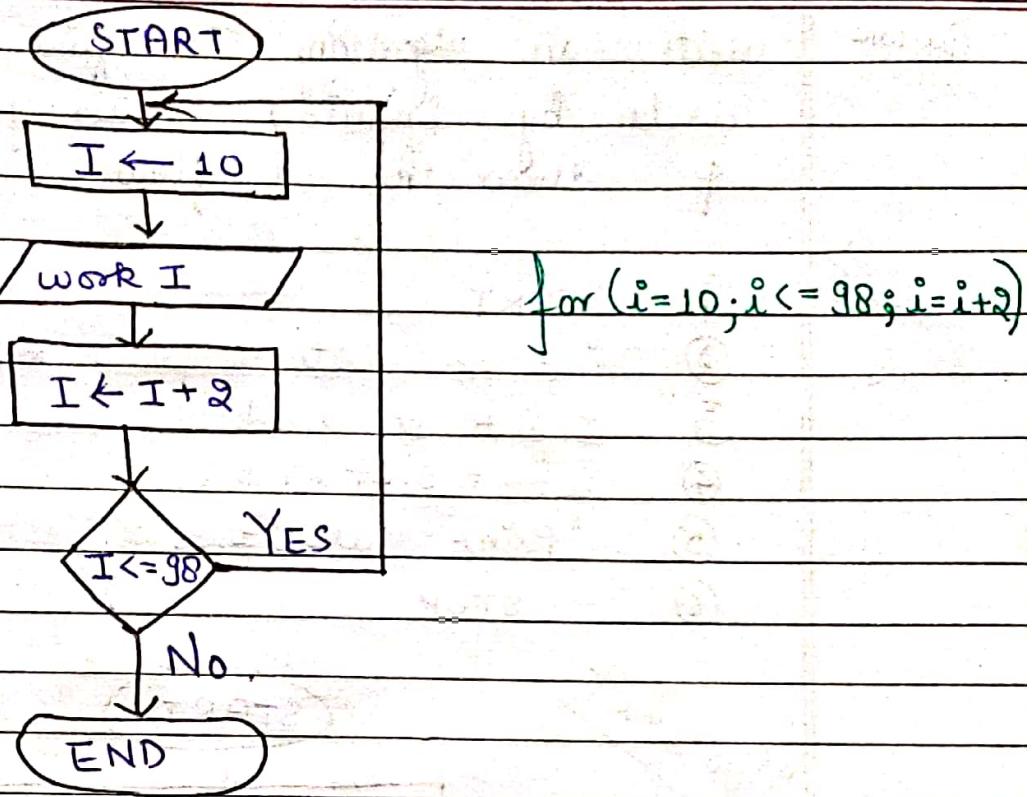
~~Algorithm~~

- ① START
- ② Print / write " enter the no. a, b "
- ③ Input / Read a, b.
- ④ If ( $a > b$ )
  - Print " a is greater than b "
  - else
  - Print " b is greater than a "
  - (End of if structure as step 4).
- ⑤ STOP.

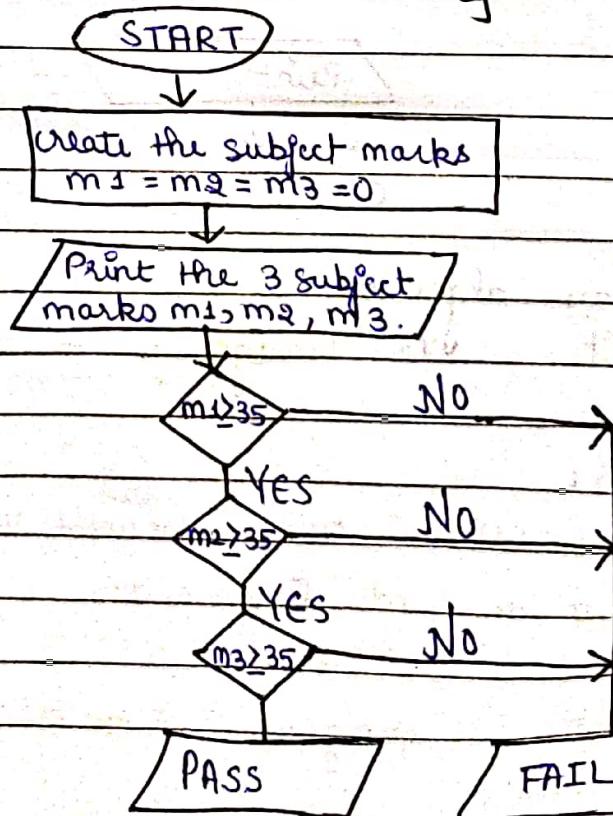
~~Flow Chart~~

Ques:-

Draw flow chart for the problem of printing even no. b/w 9 & 100.



Ques:- Flow chart to get marks for 3 subject to declare the marks, if the marks is greater than or equal to 35 in all subject, then the student passes otherwise fail.



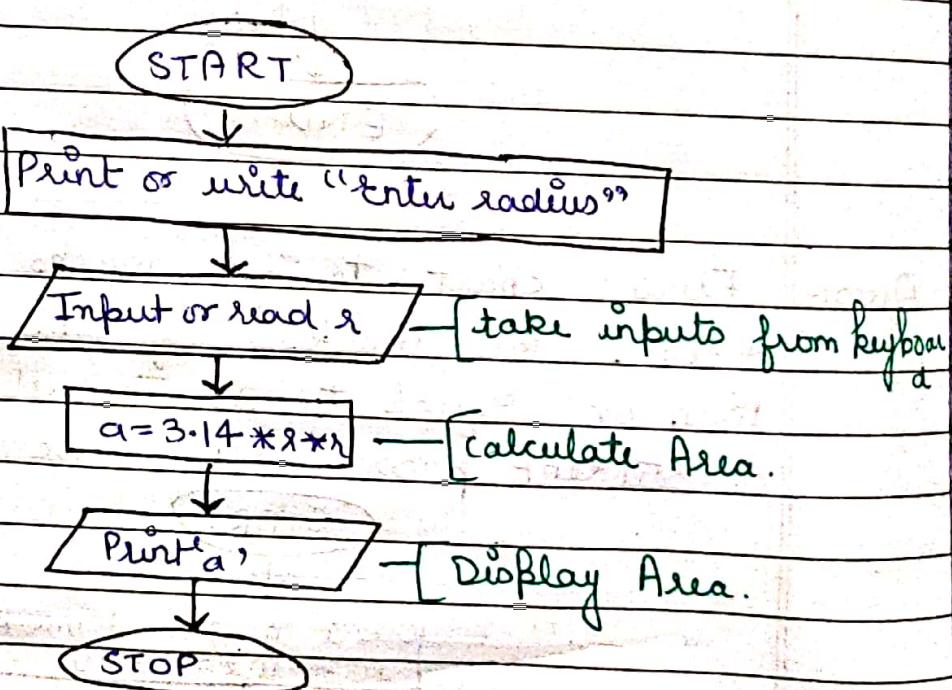
Ques-

write an algorithm To find the area of circle by inputting radius from the keyboard & draw the flow chart.

Algorithm

- ① START
- ② Print or write " enter Radius"
- ③ Input or read r.
- ④  $a = 3.14 * r * r$
- ⑤ Print "area" = a
- ⑥ STOP

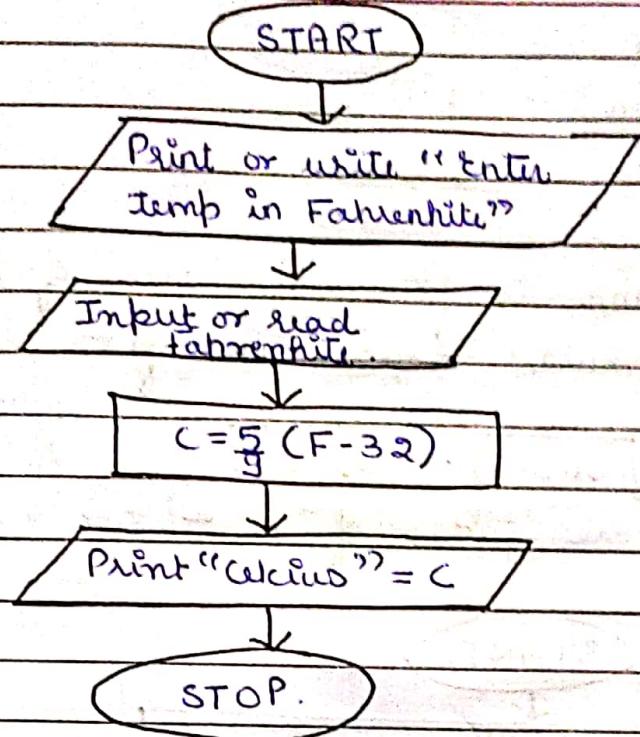
Flow chart



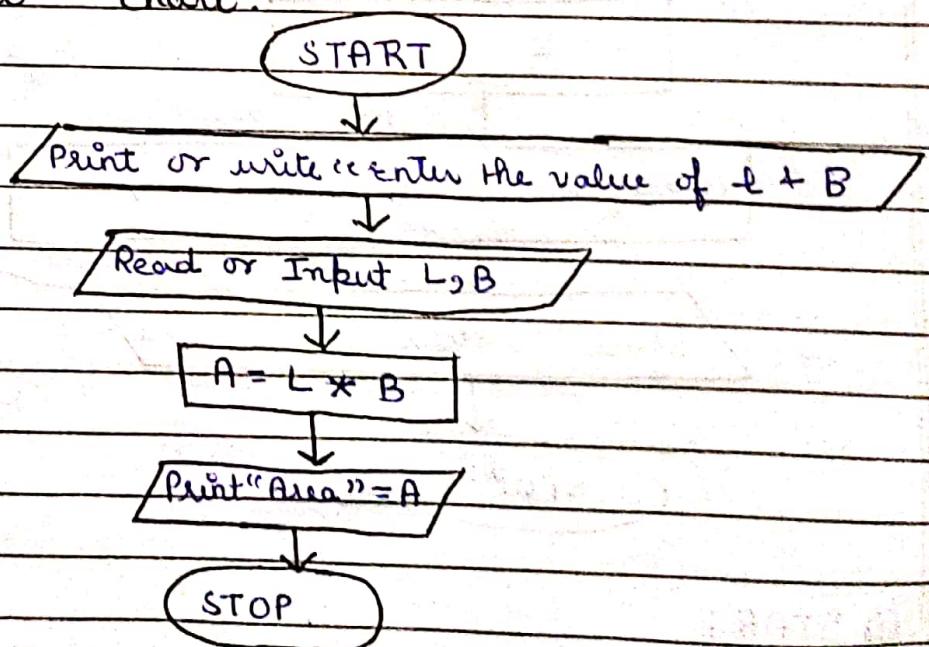
Ques-

Write an algorithm to convert the temp from fahrenheit to Centigrade & draw flow chart.

- ① START
- ② Print or write " enter Fahrenheit Value ". (temp Value).
- ③ Input or read Fahrenheit
- ④  $C = \frac{5}{9} (F - 32)$
- ⑤ Print "Centigrade" = C
- ⑥ STOP

*Flow chart*

Ques- Write an algorithm to find the area of rectangle by inputting sides from the keyboard & draw a flow chart.



① START

Print or write "enter the value of L & B".  
Read or Input L, B

②

③

④  $A = L * B$ .

⑤ Print "Area" = A

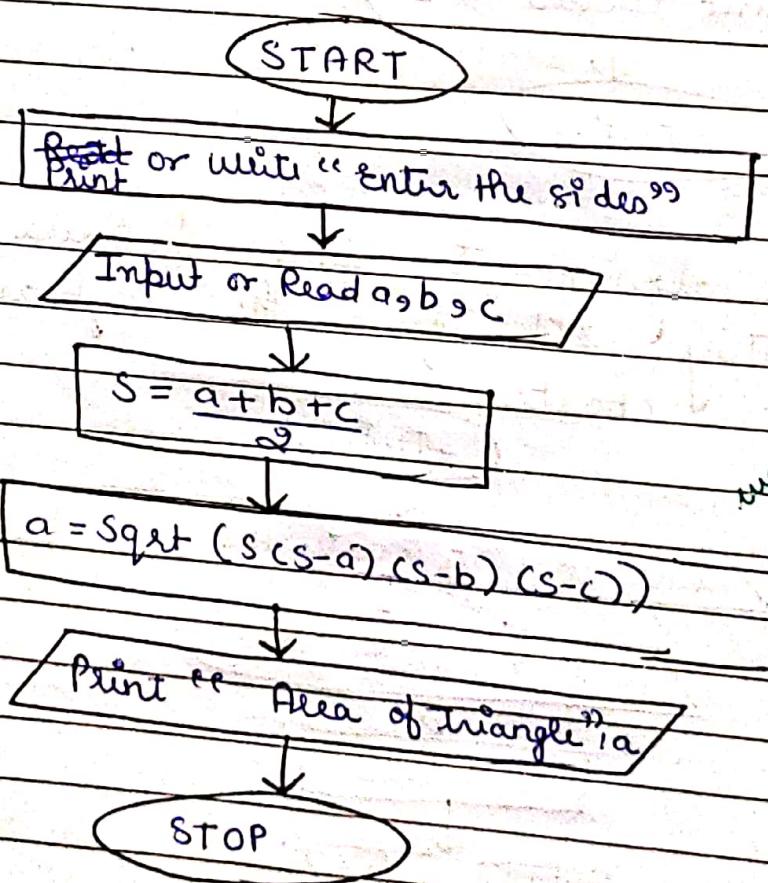
⑥ STOP.

Ques:-

Write an algorithm to find the area of  $\Delta$  using the given formula & draw the flow chart.

$$A = \sqrt{s(s-a)(s-b)(s-c)}$$

where  $a, b, c$  are sides of  $\Delta$  &  $s = \frac{a+b+c}{2}$ .



*we should use  
predefined bcoz  
multiple statement  
are done.*

① START

② Print or write "Enter the value of a, b, c"

③ Read or Input a, b, c

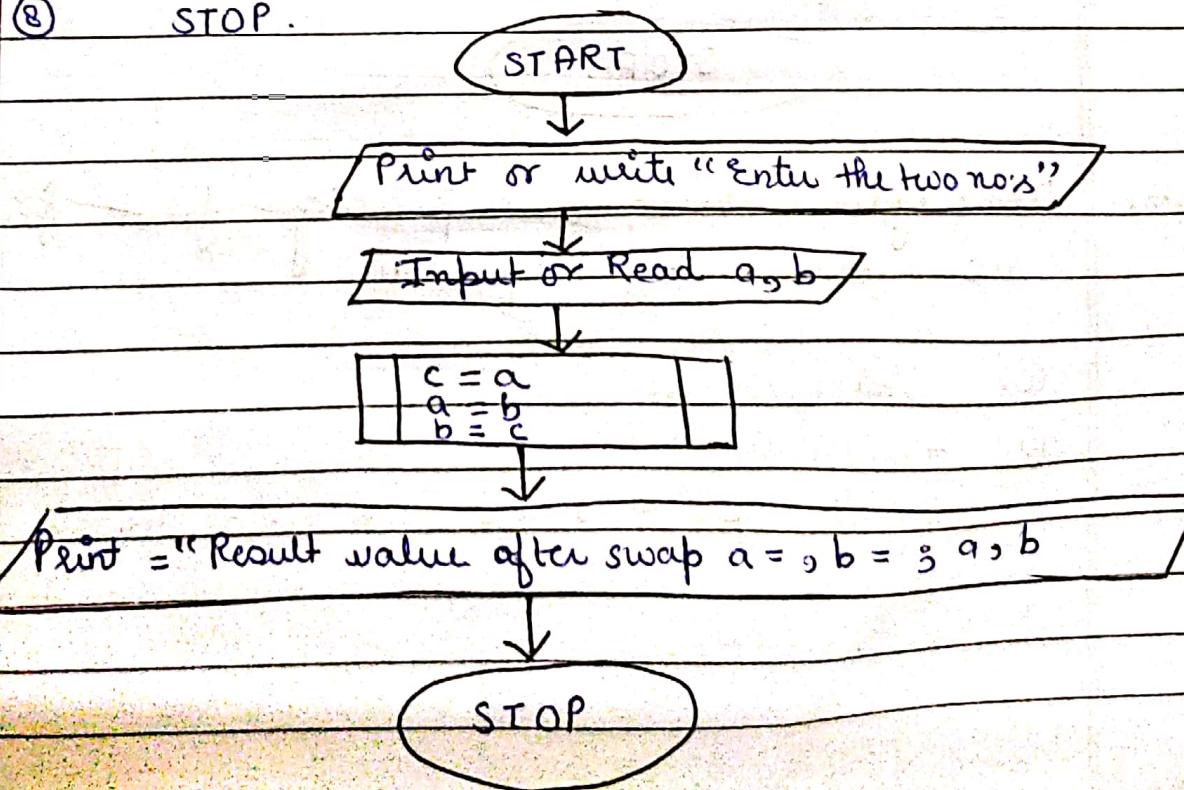
$$S = \frac{a+b+c}{2}$$

- (1)  $A = \text{Sqrt} [S(S-a)(S-b)(S-c)]$ .
- (2) Print "Area" = A
- (3) STOP.

Ques:- Write an algorithm to input Two values from the keyboard & Then swap (Interchange) the content of their.

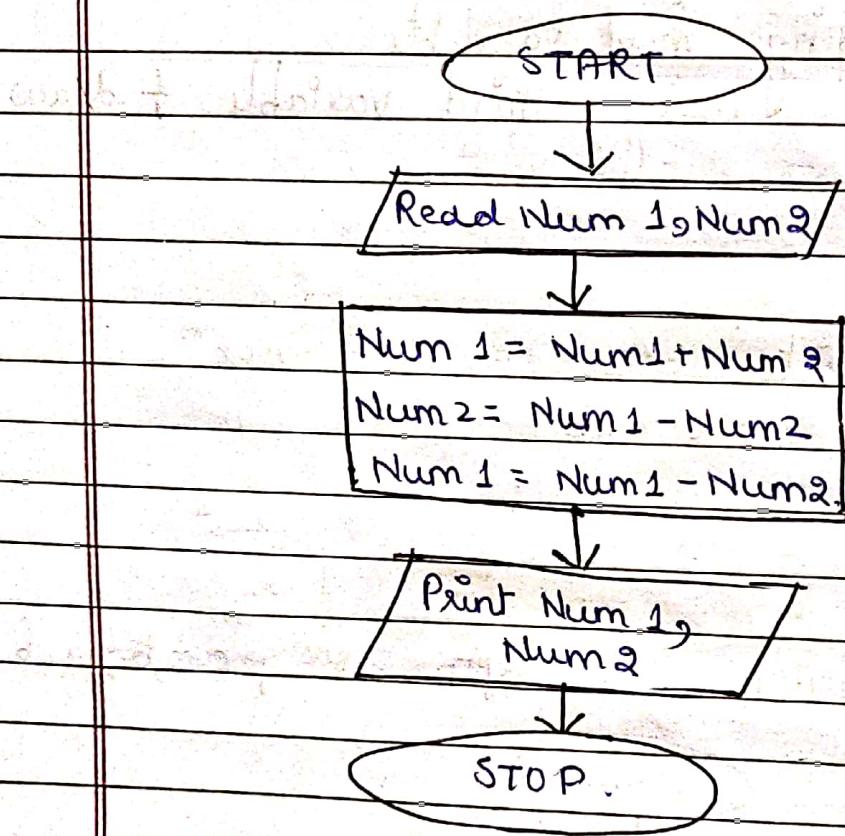
- (1) with using Third variable.
- (2) without using Third variable. & draw flow chart.

- (1).  
 ① START  
 ② Print or write "Enter the Two nos."  
 ③ Input or read a, b  
 ④  $c = a,$   
 ⑤  $a = b,$   
 ⑥  $b = c,$   
 ⑦ Print "Result value after swap"  $a = , b = , a, b.$   
 ⑧ STOP.



(ii) ① START

- ② Read the value of Num1 & Num2.
- ③  $\text{Num1} = \text{Num1} + \text{Num2}$
- ④  $\text{Num2} = \text{Num1} - \text{Num2}$
- ⑤  $\text{Num1} = \text{Num1} - \text{Num2}$
- ⑥ Print the value of Num1 & Num2
- ⑦ STOP-



Ques- Draw flow chart for printing first five odd no. & write algorithm.

① Start

② Assign  $i = 0$ Repeat steps 4, 5 & 6 until  $i = 5$  reaches

## Unit-2

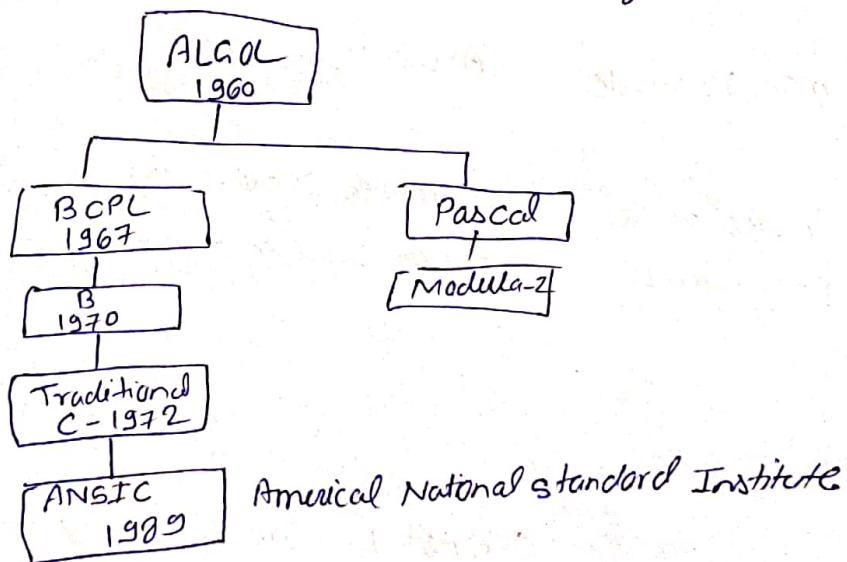
### What is C:-

C is a programming language developed at AT&T's Bell Laboratories of USA in 1972. It was designed and written by a man named Dennis Ritchie. C is often called a middle level language as it combines the elements of high level languages with the functionality of assembly language.

### Characteristics of C

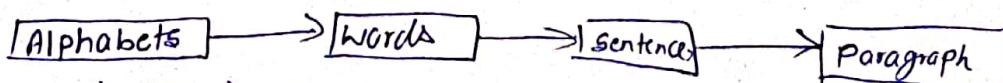
C has become a popular programming language because of its many features. The important characteristics of C are:

- \* C is a general purpose programming language.
- \* C is a Structured programming language.
- \* It has rich set of operators.
- \* It provides compact representation for Expression.
- \* It allows manipulation of internal processor registers.



### Fundamental data types :-

Steps in learning English language:-



steps in learning C



## C character set:-

A character denotes any alphabet, digit or special symbol used to represent information. Following tables shows the valid, alphabets, numbers and special symbol allowed in C.

Alphabets

A-Z, a-z

Digits

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

V-Bar

Special symbol

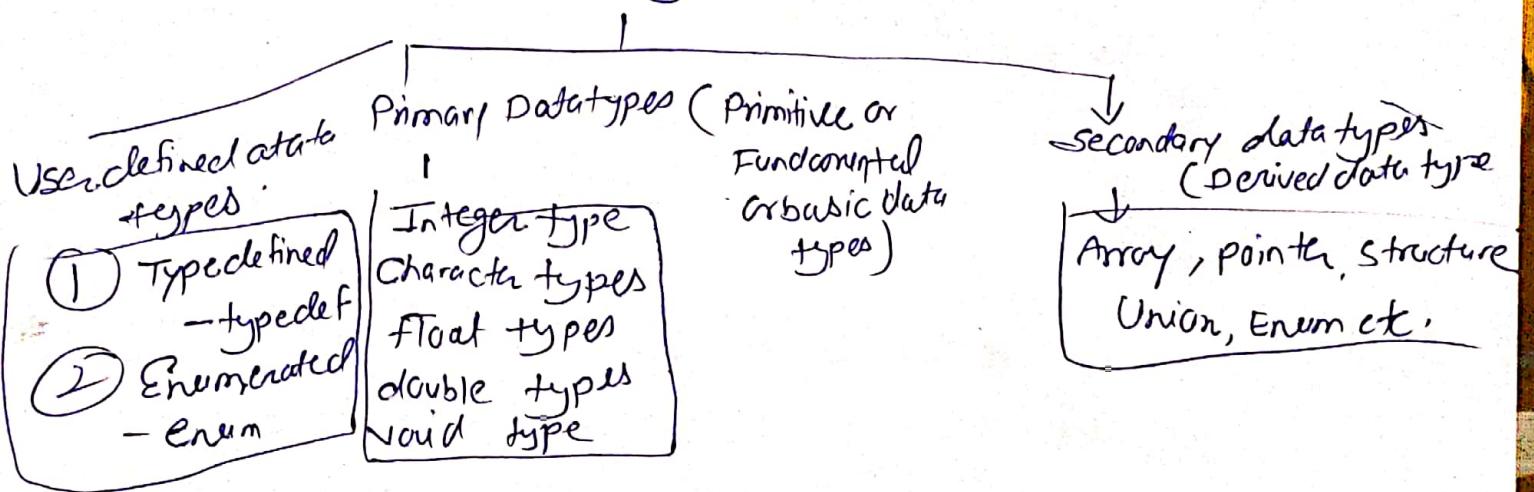
-, \*, !, @, #, %, ^, &, ;, (, ), -, +, =, \, | {, }, [ , : , ; , < , > , ? , /

Backslash

Colon separator

## Data types:-

### : Data types:-



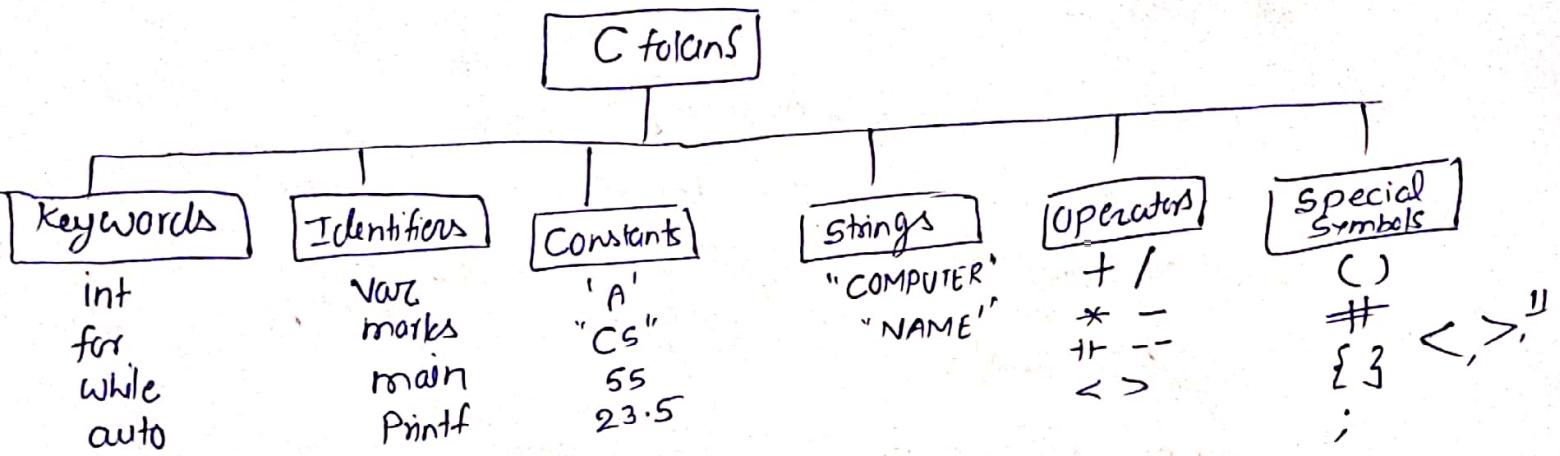
integer type: Integer represent in datatypes C  $\rightarrow$  int

## C Tokens:-

The smallest unit of a program is called the **Token**. These are the basic building blocks that can be put together to construct programs.

The six types of C tokens are as follows:-

Means. write a C program when we use words or symbols its known as **Tokens**.



### Keywords:-

Keywords are the words whose meaning has already been fixed in C language. Rules for writing Keywords are as follows:-

- (1) Keywords are also called reserved words. Means **नियत शब्द** के साथ।
- (2) keywords cannot be used as variables names. Use **नियत नाम**।
- (3) All keywords are predefined and have fixed meanings.
- (4) 32 keywords that are generally used in **traditional C**.
- (5) All keywords are in lower case.
- (6) ANSI C contains 48 keywords. (**Keywords always shown in white color as C compiler**)

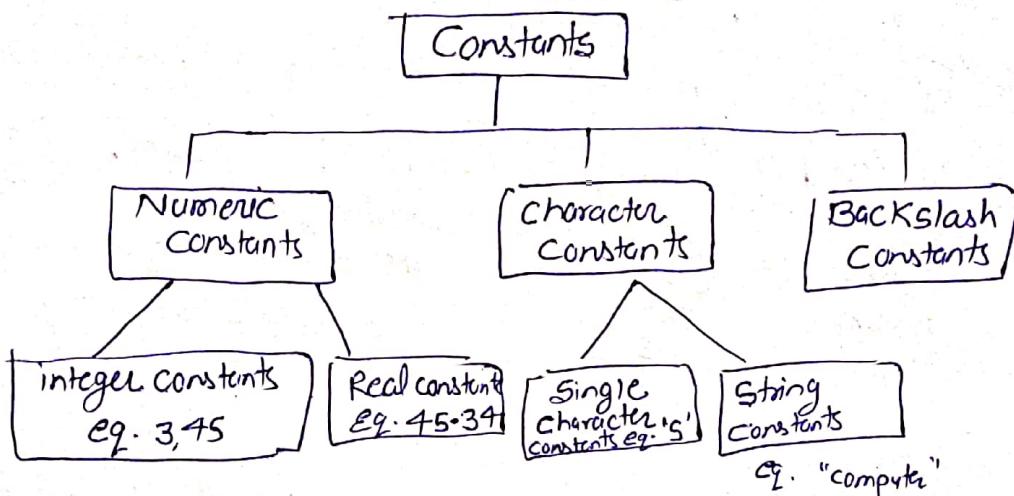
Identifiers:- An identifiers is a name for a variable, symbolic constants, functions, arrays, and so on.

Rules for writing Identifiers:-

- (1) An identifier can begin only with a letter or underscore (\_)
- (2) An identifier must consist only of letters, digits, and underscores.
- (3) Identifier cannot begin with a digit.
- (4) A C keywords cannot be used as an identifier
- (5) An identifier defined in a C library should not be redefined.

Example:- `printf`, `scanf`, `getch()`, `marks`, `computer_02`, `int i` (  
Salary\_amount, `A[20]`, `i, j, k`, `fun`, etc. valid identifiers  
`INumber`, "Y", `math_no`, `matrix first`, `ifun`, invalid identifiers.

Constants:- In C, a constant refers to a value that cannot change during the execution of a program. constants are basically of three types:-



① Numeric Constants:- Constants that contain numbers are called numeric constants. These of two types: Integer and real.

(a) Integer constants:- The collection of digit used as a constant is called an integer constant. The range of integers can be obtained using the formula  $-2^{n-1}$  to  $2^{n-1} - 1$ , where  $n$  is the size of the number of bits a computer system can process at a time.  
 These are further divided into three types:-

① Decimal constants: These are the collection of digits 0, 1, 2, ... 9. The

format specifier is `%d`. Examples are 9870, 356  
 ② Octal constants: These constants begin with 0 (zero) and are the collection of digits 0, 1, 2, ... 7. The format specifier `%o`.  
 For Example, 056, 071.

③ Hexadecimal Constants : These constants begin with 0X and ox and are the collection of digits 0, 1, 2, ... 9 and A, B, ... F. The format specifier for this is `%x` or `%X`. For Example, 0X134AB, 0x61fc.

The following are the rules for writing an integer constant:-

- (1) It must have at least one digit.
- (2) It must not contain a decimal point.
- (3) It may have either +ve or -ve sign.
- (4) When no sign is present, it is assumed to be positive.
- (5) Commas and blank spaces are not allowed in it.

### (b) Real or Floating Point Constants:-

A floating constant is a number with a decimal point.

It is defined as a sequence of digits preceded or followed by the decimal point. It has two forms:-

- (1) Fractional form:- The following rules are to be followed while writing a real constant in fractional form:

- (1) It must have at least one digit before or after the decimal point.
- (2) It may have either +ve or -ve sign.
- (3) Commas and blank spaces are not allowed to be ~~+ve~~ if.

- (2) Exponent Form: This constant consists of two parts, mantissa and exponent. The part before e is called mantissa and after e is called Exponent. The rules for writing real constants in exponent form are as follows:-

- Mantissa and exponent are separated by e.
- (2) The exponent must be an integer of at least one digit. It may be either +ve or -ve. However, by default it is +ve.
- (3) The mantissa must be either an integer or a real constant in fraction form.
- (4) Mantissa may be +ve or -ve.

valid constants

+15.5E4

.04e-12

12.5

29.4

Invalid constants

+65.3E (No digit after exponent)

23.6e2.5 (Exponent must be,

54.2 (Decimal point missing)

012

② Character Constants:-  
constants that contain characters are called character constants. These are further divided into two types:-

(a) Single character constants: The constant that is written under single quote marks (' ') is called a single character constant for example '2', 'a', 'A'

(b) String : The constant that is written under double quote marks (" ") and is a collection of characters is called string.  
For Example, "computer", "A".

(3) Backslash constants: The constant that is a combination of two characters in which the first character is always a backslash (\) and the second character can be any one of the characters n, a, b, f, r, t, v, ;, ", \, and \0. These are also known as escape sequences.

Example:-

① `printf ("Hello world");`      Angular bracket // C symbol  
|  
// identifier      || String

② `int x = 5;`      constants (Numeric)  
|  
Keywords      identifiers

## [ - Data type ]

Fundamental or  
I.O or Basic  
data-type

Derived  
data-type

User-defined  
data-type.

\* Declaration of variables:- After designing suitable variable name, we must declare them to the compiler declaration does two things -

- (a) It tells the compiler what the variable name is.
- (b) It specifies what type of data, the variable will hold / exist.

- A variable can be used to store a value of any data types
- The syntax for declaring a variable is as follows:

**data-type variable\_name;**

A declaration statement must ends with a semicolon.

- \* Data-types :- Data type indicates four things they are -
- Type of data can be stored in a variable or a constant.

- Amount of memory to be allocated for a specific type of data.
- Possible range of values can be stored in a specific type of variable or a constant.
- Type of operation can be performed on a specific type of data.

\* → C-lang supports three classes of data types

1. Primary data types  
(Fundamental or Basic data types)

- character data type
- integer data type
- double floating point data types
- float data type
- void data type

a. Character Integer data type :- The keyword int is used to indicate an integer no., any integer no. is a sequence of digits without a decimal point

→ Types      Keywords      Size of memory      Range of data size / value

Integer	int	2 bytes	$-2^n$ to $2^n - 1$
		1 byte = 8 bit	-32,768 to 32,767 (signed).
		2 bytes = 16 bit	0 to 65536 (unsigned).

b. character data types :- The keyword `char` is used to indicate the character data type. The data may be character constant or a string constant.

→ Types      keywords      size of memory      range of data size

character	<code>char</code>	1 byte = 8 bit	-128 to 127 (Signed) 0 to 255 (Unsigned)
-----------	-------------------	----------------	---

c. floating point data types :- The keyword `float` is used to indicate a floating point no. (floating point or real nos.) are stored in 32 bits (on all 16 bit + 32 bit machine, with 6 digits of precision). means taken value after decimal is only 6 digits.

→ Types      keyword      size of memory      range of data size.

Floating point	<code>float</code>	4 byte	$3.4 \times 10^{-38}$ to $4 \times 10^{38}$
		4 byte = 32 bit	

d. Double floating point data types :- The keyword `Double` is used to indicate the double precision floating point no. when the accuracy provided by the `float` no. is not sufficient, the type `double` can be used to define the no.

\* Double data type no. uses 64 bits given a precision of 14 digits.

The no. that double type represents the same data type that float represent but with a greater precision. To extend precision further we may use long double with which uses 80 bits.

→	Keywords	Data Types	Memory Size	Value Size or data storage
	double	Double floating point data types	8 byte 8 byte = 64 bits	1.7e - 308 to 1.7e + 308

(e) void data types:- The void type has no value. This is usually used to specify the type of function.

The type of a function is said to be void when it doesn't return to the calling function.

\* Modifiers:- Data type modifiers are used to modify the properties of primitive or basic data types. (except float & void data types) according to application requirement, so that we can able to utilise the comp memory.

- With the help of data type modifiers we can-
  - (1). Modify the size - i.e. is the amount of memory to be allocated.
  - (2). Modify the signed - i.e. they decide only positive or both true & -ve values can be stored.

→ There are four data types modifier-

(1) signed    (2) Unsigned    (3) short + (4) long

Signed :- This can be applied to integer variables.

The default declaration assumes a signed no. The signed modifier can also be applied with a char data types to be create the small integers.

Unsigned :- This can be used for the both integer + characters. It is used to create an unsigned integer + it can also be used in combination with long + short.

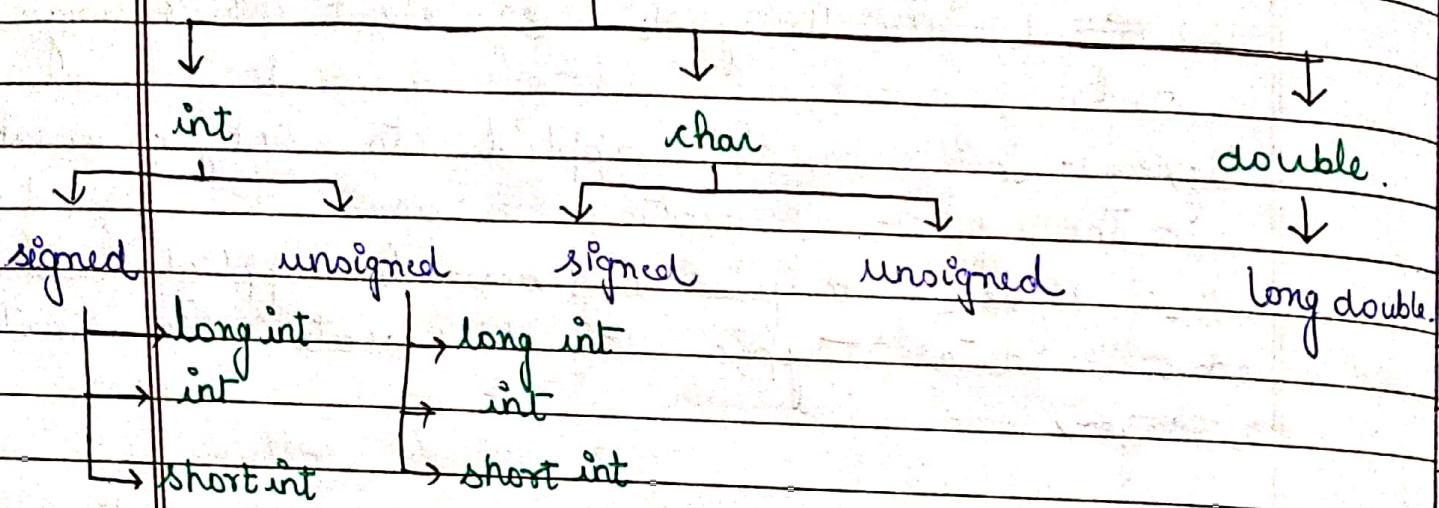
Long :- This is applied to integer data types, if an integer is of 16 bits then long int is of 32 bits.

Similarly, this can be used with double.

short :- This makes the size of an integer half, but most compilers have 16 bits integer. ANSI requirement is that the smallest acceptable size for integer is 16 bits.

Most of the systems provides the size of short int as that of integer.

## Modifiers



SNo.	Data Type	Keyword	format specific	size in bit	size in bytes	Range
①	Integer	int	%d	16	2	-32768 to +32769
	Signed int	int	%d	16	2	-32768 to +32769
	Unsigned int	int	%u	16	2	0 to 65535.
	Short int	int	%d	16	2	-32768 to +32767
	Signed short int	int	%d	16	2	-32767 to +32767
	Unsigned short int	int	%u	16	2	0 to 65535
	Long int	long	%ld	32	4	-9147,483648 to +9147,483647

	Signed long int	%ld	32	4	-2147, 483647 to +2147, 483647
	unsigned long int	%lu	32	4	0 to 4894967 295
(g)	Character .char	%c	8	1	-128 to 127
	Signed char Unsigned char	%c	8	1	-128 to 127 0 to 255
(3)	Floating point	float	%f	32	4 3.4e - 38 to 3.4e +38
		double	%lf	64	8 -1.7e - 308 to 1.7e +308
		long double	%lf	80	10 -3.4e - 4932 to 1.1e +4932

\* **Storage class of c -** the class that determines the storage location, initial value or default value, scope or visibility & life time of the variable is called storage class.

(1) Initial value or default value - The first value that is assigned automatic is called default value (as a garbage value).

(2) storage location - The location of a variable in the memory, that is main memory or CPU registers is called the storage location or memory location.

(3) scope - The area of the program in which the variable is accessible or visible is called scope.

(4) life time - The duration of execution of a program during which the variable retains its value into memory is called its life time.

→ There are four storage classes in C -

- ① Automatic storage class or Default storage class.
- ② Register storage class
- ③ Static storage class
- ④ External storage class

(1) Automatic Storage Class - Storage location of a memory member will be main memory (RAM)

- (b) Initial or default value of the variable will be garbage value.
- (c) scope will be local scope.
- (d) life time will be the execution of the block in which it is defined.

main ()

```

    {
        auto int x = 5;
        printf ("%d \n", x);
    {
        int x = 2;
        printf ("%d \n", x);
    }
    printf ("%d \n", x);
}

```

Result → 5  
2  
5.

- (a) Register Storage Class:- storage location of a member will be CPU register.
- (b) Initial or default value of the variable will be garbage.
- (c) Scope will be local scope.
- (d) life time will be the execution of block in which it is defined.

```

int main ()
{
    register int x=5;
    int y;
    y = x++;
    x--;
    y = x+1;
}

```

### \* (3) Static Storage Class of ~~etc~~ c

- (a) Storage location of a member will be main memory (RAM)
- (b) Initial or default value of the variable will be zero.
- (c) Scope will be local scope.
- (d) Life Time, it persist its value in different function calls.

void f1()  
main()

{  
    f1();  
    f1();  
}

void f1();  
{

    int i=0;  
    i++;  
    printf("i=%d\n", i);  
}

Result → i=1

i = 1

```

void f1();
main()
{
    f1();
    f1();
}
void f1();
{
    static int = i;
    i++;
    printf("i = %d\n", i);
}

```

Result  $\rightarrow i = 1 \quad i = 2$

#### \* (4) External Storage Class -

- (a) Storage location of a member will be main memory (RAM)
- (b) Initial or default value of the variable will be zero.
- (c) Scope will be the global scope.
- (d) Life time will be whole program execution.

```
int x; int x;
```

```
main()
```

```
{
    extern int x;
    printf("x = %d", x);
    f1();
    printf("x = %d", x);
}
```

```
int x;
```

```
void f1()
```

```
{
    int x = 5;
```

```
    x++;
    printf("x = %d", x);
```

Result  $\rightarrow x = 0$

$x = 6$

$x = 0$

\* o Differences b/w different storage classes

Sno.	Storage class	Where stored (location)	Initial Or default Value	Scope visibility	Life time
1.	Automatic	RAM	Garbage Value	local to the block in which it is defined.	The execution of block in which it is defined.
2.	Register	CPU registers	Garbage Value	local	The execution of block in which it is defined.
3.	Static	RAM	0	local	Persist its value in different function calls
4.	External	RAM	0	global	In-line program execution.

\* Operators :- An operator is a symbol that tells the comp. in which mathematical & logical operation are to perform. Operators are broadly classified as follows:-

- (1) Unary Operators - The operator that operates over a single operand is called unary operators for ex:- logical Not (!), bitwise Compliment (-) & Unary Plus/Minus (+/-)
- (2) Binary operators - The operator that operates upon Two operands is called Binary operators ex:- logical AND (&&), logical OR (||) & Relational operators.
- (3). Ternary Operators - The operator that operates over three operands is called as ternary operators for ex:- conditional operator (?/)
- \* Types of operators - Based on classification, operators are of eight Type
- Arithmetic operators
  - Relational "
  - Logical "
  - Assignment "
  - Increment & Decrement operators
  - conditional operators
  - Bitwise operators
  - Other operator

- (1). Arithmetic operators - (+, -, \*, /, %) operators that perform arithmetic operations are called arithmetic operators. for eg:-  $A = B + C$ . where, A, B, C are operands & +, = are operators.

→ There are three arithmetic operations that are based on category of operands.

(a) Integer Arithmetic expression:- Only containing integer operands. for eg:-  $A = 4 * 5 + 2$ .  
 $B = 2 + 3 - 2$ .

(b) Real Arithmetic expression:- Only real or floating point contains. eg:-  $A = 1.5 + 0.5 * 3.4$ .

(c) Mixed mode arithmetic expression:- when one operand is integer & the other is real the expression is called mixed mode arithmetic expression. The result of this expression is always real. eg:-  $A = 1.5 * 2 * 2$ .

(d) Relational operators:- We often compare two quantities & depending on their relation take certain decisions. for ex:- we may compare the age of two person or the price of two items & so on.

<sup>ee</sup> "The result of relational operation is either 1 (True) or 0 (False)."

→ C supports 8 relational operator in all that are given below -

Operator	Meaning & Purpose	Example	Result
(1) <	'is less than'	$5 < 10$	1 [True]
(2) >	'is greater than'	$5 > 10$	0 [False]
(3) <=	'is less than or equal to'	$5 <= 10$	1 [True]
(4) >=	'is greater than or equal to'	$5 >= 10$	0 [False]
(5) !=	'is not equal to'	$5 != 10$	1 [True]
(6) ==	'is equal to'	$5 == 10$	0 [False]

(3). logical operators :- In addition to the relational operators, C has the following 3 logical operators . -

(i) logical AND.

(ii) logical OR

(!) logical NOT.

for e.g:- ① If (age > 55 & salary < 1000)

② If (number < 0 || number > 100)

(4). Assignment operators :- Assignment operators are used to assign the result of an expression to a variable. C has a set of "short hand assignment operators" in the form of

$$V \cdot O P = \text{expression}$$

or,  $\text{Exp 1 OP} = \text{Exp 2}$

⇒ where  $V$  = variable,  $OP$  = operator &  $\text{Exp}$  = expression

→ The operator  $OP$  is k/a the short hand assignment operator.

$$\text{Exp 1. } (x + \textcircled{1}) = (x + 1)$$

→ This is same as the statement  $\{x = x + (x+1)\}$  result.

The short hand operator  $+ =$  means (add  $x+1$  to  $x$ ) or (encirclement by  $x+1$ ).

(a).  $x + = 3$ .

$$\therefore [x = x + 3]$$

(b)  $a * = a + 1$ .

$$\therefore [a = a * (a+1)]$$

(c).  $a / = 3$

$$\therefore [a = a / 3]$$

(d).  $a \% = b$

$$\therefore [a = a \% b]$$

Ques 1- # define N 100

# define A 9

main ()

{

int a;

a = A ;

while (a < N)

```

{ printf ("%d \n", a);
  a *= a;
}

```

Result  $\rightarrow$  2      } output contains only  
                 4      } 3 values i.e. 2, 4, 16  
                 16

- (v) Increment & Decrement operator - The operators that add one & subtract one to the operand are called increment (`++`) & decrement (`--`) operators respectively.  
 These operators are of two types -

- a. Preincrement or predecrement (prefix) - These operators add or subtract one of the operand & then assign the result to the variable respectively.

e.g. int  $x = 5, y$

$$y = ++x ; \quad | \quad x = 6, y = 6$$

$$y = --x ; \quad | \quad x = 4, y = 4$$

- b. Post increment or post decrement (Postfix) - These operators add or subtract one to the operand & then assign the result to the variable respectively. e.g. - int  $x = 5, y$

$$y = x++ ; \quad | \quad y = 5, x = 6$$

$$y = x-- ; \quad | \quad y = 5, x = 4$$

tg(1).

main ()

{ exp of  
increment }

```

int x = 3;
clrscr();
x++;
printf ("%d", x);
++x;
printf ("%d", x);
getch();
}

```

Result 4, 5.

tg(11).

main ()

{ exp of  
decrement }

```

int x = 3;
clrscr();
x--;
printf ("%d", x);
--x;
printf ("%d", x);
getch();
}

```

Result 3, 1.

(6)

Conditional operators - A ternary operator  $(?:)$  is available in C to construct conditional expression of the form -

Exp 1 ? Exp 2 ; Exp 3.

where Exp 1, 2, 3 are expressions.

- Exp 1 is evaluated 1st. If it is a non-zero (true), then the Exp 2 is evaluated & becomes the value of the expression.  
Note that only one of the expression (either Exp 2 or Exp 3 is evaluated).

e.g:-  $a = 10, b = 15.$

$$x = (a > b) ? a : b;$$

- In this example x will be assigned the value of b. This can be achieved using the if-else statement as follows:

*if (a > b)*

*x = a;*

*else*

*x = b;*

- (7). Bitwise operators :- C has supportive special operator &/a bitwise operators for manipulating of data at bit level. These operators are used for testing the bits or shifting them right or left. Bitwise operators may not be applied to float or double operators.

**+**

**/**

**^**

**<<**

**>>**

**Meaning**

**Bitwise AND**

**Bitwise OR**

**Bitwise exclusive OR**

**Shift Left**

**Shift Right**

- (a) Bitwise AND :- The result of this operator is 1 when both bits are one. otherwise the result is zero.

- b. Bitwise OR :- The result of this operator is zero, when both bits are zero, otherwise the result is 1.
- c. Bitwise Exclusive OR :- The result of this operator is 1 when both bits are one, otherwise it is zero.
- d. Bitwise left shift :- change the content by shifting bits to the left side.
- e. Bitwise right shift :- change the content by shifting bits to the right sides.

Result

X	Y	$X \& Y$	$X   Y$	$X ^ Y$
0	0	0 0	0	1
0	1	1 0	1	1
1	0	1 0	1	0
1	1	0 1	1	0

- (Q). Special operators :- C supports some special operators of interest such as comma operator, size of operator (), pointer operations (+, \*) & member selection operators.

- a. The comma operator, - The comma can be used to link the related expressions together. A comma linked list of operations are evaluated left to right and the value of eight most

expression is the value of combined expression  
ex:- the statement value ( $x=10, y=5, x+y$ );

First assign the value 10 to  $x$ , then assign 5 to  $y$   
& finally assign 15 to value  $x+y$ .

- b. The size of operators :- The size of is a compile time operator when used with an operand.
- It returns the no. of bytes, the operand occupies.
- The operand may be a variable, constant or a data type specifier, qualifier.
- The size of operator is normally used to determine the length of array & structures when their sizes are not known to the programmer.
- It is also used to allocate memory space dynamically to a variable during execution of a program

\* → size of (data type)

main()

{

int x;

clrscr();

$x = \text{size of (int)}$ ; or  $x = \text{size of (char)}$ ;

printf ("%d", x);

getch();

}

Result →  $x = 2$  ~~or 1~~

\* → size of variable  
main()

{

int x, y;

find K;

double d1;

char c;

x = size of (d1);

printf ("%d", x);

getch();

}

Result → x = 8.

\* → size of constants

main()

{

int x, y, z;

clrscr();

x = size of (32)

y = size of (32.6)

z = size of ('a')

printf ("%d %f %d", x, y, z);

getch();

}

Result → x = 8, y = 8, z = 2.

\* operator precedence & associativity - Determined which operator is evaluated first if there are more than one operators.

An arithmetic expression with out parenthesis will be evaluated from left to right using the rules of precedence of operators.

→ There are two priority levels of arithmetic operator in C lang-

(1) High priority ( $*$ ,  $/$ ,  $\%$ )

(2) Low priority ( $+$ ,  $-$ )

→ for expt-  $a = 100 - 2 * 30$

②: ① Precedence.

$$a = 100 - (2 * 30)$$

$$a = 100 - 60$$

$$\boxed{a = 40} \text{ Ans.}$$

Ques 1-  $x = 9 - 12/3 + 3 * 2 - 1$   
 ①                    ③.

$$x = 9 - 4 + (3 * 2) - 1 \quad ( / * \text{ has precedence})$$

$$x = 9 - 4 + 6 - 1 \quad \begin{matrix} \text{same then use left} \\ \text{to right operator} \end{matrix}$$

$$x = 5 + 6 - 1$$

$$x = 11 - 1$$

$$\boxed{x = 10} \text{ Ans.}$$

Ques 1-  $a = 100 + 200/10 - 3 * 10$   
 ①                    ③

$$a = 100 + 20 - (3 * 10) \quad ( / * \text{ has same})$$

$$a = 100 + 20 - 30 \quad \begin{matrix} \text{precedence then we use} \\ \text{left-to-right operator} \end{matrix}$$

$$a = 120 - 30$$

$$\boxed{a = 90} \text{ Ans.}$$

\* Associativity in C lang :- Associativity is used when there are two or more operators of same precedence is present in an expression. for ex:- multiplication & division operator has same precedence in an expression, then associativity is left to right for these operators is evaluated.

Ques:-  $(1 == 2) != 3$  ] .  $(1 == 2) \rightarrow$  first expression is evaluated.

i.e.  $(1 == 2)$  executes 1<sup>st</sup> resulting into true & false(0). Then  $(0 != 3)$ , executes resulting into one true (1).

where  $=$  sign doesn't equal to the same precedence, so using associativity.

Ques  $\rightarrow$   $(1 > 2 + 3 \& \& 4)$ . (precedency of  $&$  is greater among all the sign.)  
 $(1 > (2+3) \& \& 4)$ .  
 $(1 > 5 \& \& 4)$ . ( $>$  And  $\&$  has same precedence, so we use LTR)  
 $(0 \& \& 4)$   
 $(0) \rightarrow$  False.

Ques- The result of the given program is -

```
int main()
{
    int a = 1;
    int b = 1;
    int c = a++ - - b;
    int d = a-- + + -- b;
    printf("a=%d, b=%d, c=%d", a, b, c, d);
    return 0;
}
```

Result  $\rightarrow c = 1, a = 1, b = 0, d = 0 \quad c = 9 \text{ } 11 \text{ } B$   
 $c = 1 \text{ } 11 \text{ } 0$   
 $c = 1$

Ques:- int  $y = 10.$

$$\begin{aligned} & ++y + y++ + y-- + y++ ; \\ & \{(++y) + (y++) + (y--) + (y++)\}; \\ & \{11 + 12 + 11 + 12\} = 46 \text{ Ans.} \end{aligned}$$

Ques:-  $a = 10$

$$b = a++ + a-- - - - a.$$

$$b = (a++) + (a--) + - - - a.$$

$$b = 9 + 9 - 9 - - - 9$$

$b = 9$	Ans.
---------	------

Ques:-  $y = 10;$

$$y = (++y) + (y++) - (++y) + (y++) - (y-) ;$$

$y = 13$	Ans.
----------	------

\* Type Conversion in C :- The technique of converting the value of one type into another is called type conversion.

It is of two types -

- a. Implicit Type Conversion :- C language permits mixing of constant & variable of diff types in an expression. When C automatically converts any intermediate value to the proper type so, that the expression can be evaluated without loss of

any expression. It is called implicit type conversion  
 exp:- (exp of Implicit)

main ()

{

int a;

float b;

a = 3.14;

b = 30;

printf ("%d", a);

printf ("%d", b);

}

Result → a = 3

B = 30.000000

bcz, this is an integer type variable so, only store integer value & octal (point value) is truncated (discard)

b.

Explicit Type Conversion :- When one type is converted to another forcefully by a programmer, it is called explicit type conversion.  
 The syntax of explicit type conversion is data type (then as a value)

exp:-

main ()

{ int a;

int b;

float c;

c = ~~alpha~~ float (a/b);

printf ("c %.f", c);

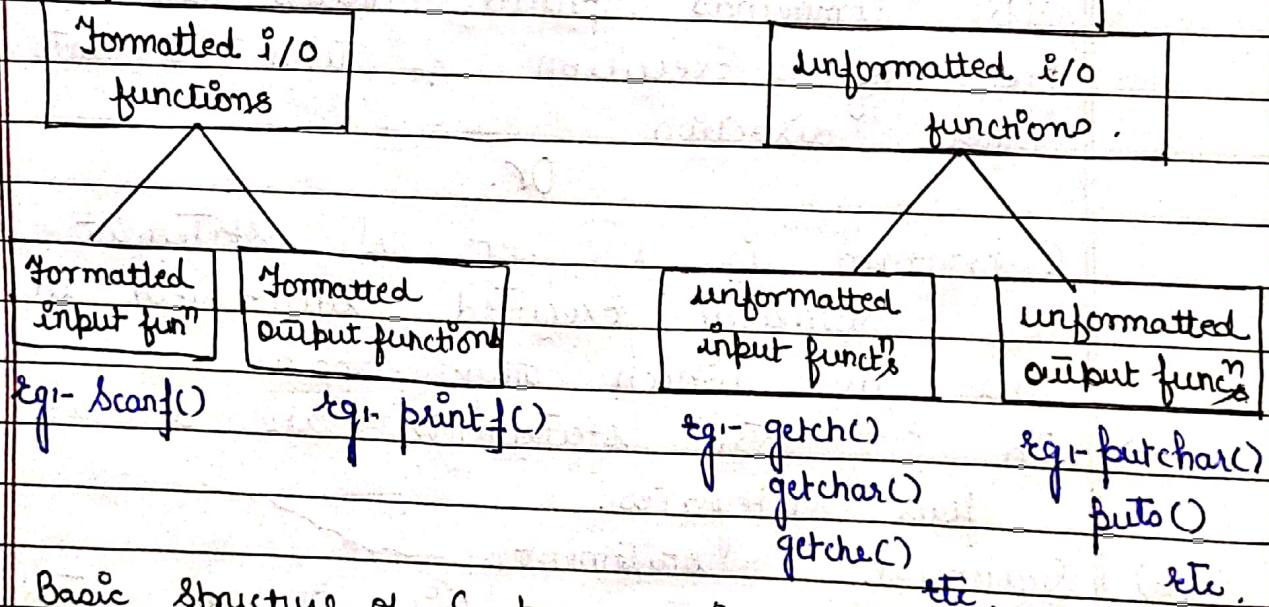
}

Result →  $c = 2.500000$

\* Structure of C program :-

→ Standard of I/O in C - The standard I/O functions are used to take the input from the console & provide output to the console.

stored input/  
output functions



\* Basic Structure of C programming.

① Documentation section

/\* comments section  
objective.

② Link section. (Pre-processor section).

#include<stdio.h>

#include<conio.h>

<math.h>

<graphics.h>.

④ Main()

① Declaration part

② Execution part

③ Sub function

function 1

function 2

⋮

n function

③ Definition section.

Void fun()

Tuesday

## UNIT - 3 Conditional Program Execution

### \* Conditional Program execution.

**Control statements:-** C lang, program execution start from the first line of the main() function & continuous after line after line, till the end of the function is reached.

Ctl. Commands provide means for the normal flow of execution & turn it into some other direction.

Or.

C program is a set of statement which are normally executed sequentially in the order in which they appear.

→ In C ctl. statements are divided into four categories.

- Sequential statement.
- Conditional statements (if, if else, nested if else)
- Itrative or looping statements (for, while, do while + switch)
- Unconditional or jumping statements (break, continue, return, exit, goto & so on)

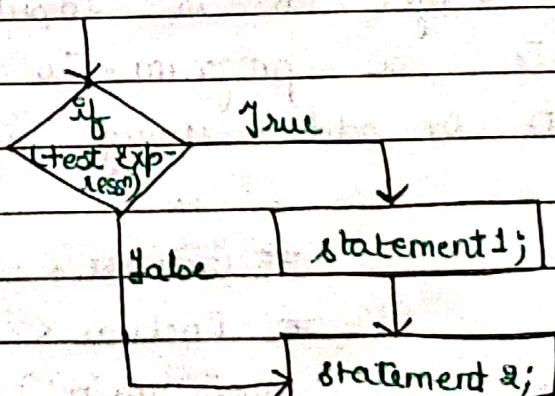
[1] Conditional Statement :- The execution of conditional statement involves both decision making & branching.

The following are the conditional statement -

(a) if statement :- The if statement is a conditional statement or decision making statement. If is a two way branching statement & uses a logical statement that result in either true or false. If the result is true, the body of the if statement is executed otherwise, the control will be transferred to outside the if statement.

→ The syntax & flow chart is as follows -  
 syntax → if (logic expression)  
 {  
 statement 1 ;  
 }  
 statement 2 ;

flow chart →



Ques:- Two integer no.'s are input through the keyboard with the program to find out the greater no. using if statement.

Sol:-

```

#include <stdio.h>
#include <conio.h>
void main()
{
    clrscr();
    int x, y;
    printf (" enter the two no. , x & y ");
    scanf ("%d %d", &x, &y);
    if (x > y)
    {
        printf (" x is greater ")
    }
    if (x < y)
    {
        printf (" y is greater ")
    }
    getch();
}.

```

Ques :- One integer no. is input through the keyboard. write a program to find out the no. is even or odd using if statement.

Sol:-

```

#include <stdio.h>
#include <conio.h>
void main()
{
    clrscr();
    int x;
    printf (" enter the no x ");

```

```

scanf ("%d", &x);
if (x % 2 == 0)
{
    printf ("x is even no");
}
else
{
    if (x % 2 != 0)
    {
        printf ("x is odd");
    }
    getch();
}

```

(b). if - else - statement :- This is also a two way branching statement in this the body of if is executed. If the result of logical expression is true ; otherwise the else block is executed automatically & then the control is transferred to the next statement.

→ The syntax & flow chart are as follows -

Syntax →      if (logical expression)

{

    Statement 1 ;

}

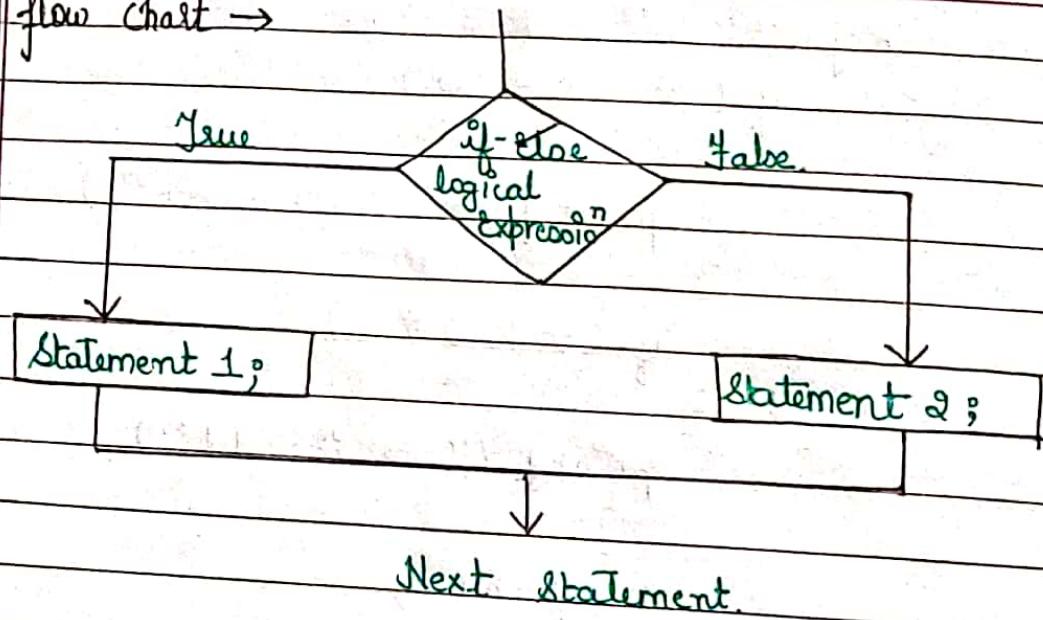
else

{

    Statement 2 ;

}

flow chart →



→ where if & else are the keywords.

Ques:- Two integer no. are input through the keyboard.  
WAP To find out the greatest no. using  
if - else statement.

```

#include <stdio.h>
#include <conio.h>
void main()
{
    clrscr();
    int x, y;
    printf("Enter the two no. x & y");
    scanf("%d %d", &x, &y);
    if (x > y)
        printf("x is greater");
    else
  
```

→ `void main` indicates that the main funct' will not return any value.

Date: \_\_\_\_\_ Page: \_\_\_\_\_

```
{  
    printf("y is greater")  
}  
getch();  
}.
```

Ques:- One integer no. is input through the keyboard. WAP to find out the no. is even or odd using if - else statement.

```
#include < stdio.h>  
#include < conio.h>  
void main()  
{  
    clrscr();  
    int x;  
    printf("enter the no. x");  
    scanf("%d", &x);  
    if (x % 2 == 0)  
    {  
        printf("x is even no.");  
    }  
    else  
    {  
        printf("x is odd no.");  
    }  
    getch();  
}.
```

(c) nested if else statement :- When an if - statement is written inside another if - statement , it is called nested if else statement . When an if - else statement is written inside another if - else block it is called nested if else statement .

→ Such statements are used when multiple conditions are specified to perform a given task so , an if else statement may have multiple if else inside its body .

→ The syntax & flow chart are as follows -

Syntax → if ( test - exp<sub>1</sub> )

{  
if ( test - exp<sub>2</sub> ) .

Statement 1 ;

}

else

{ Statement 2 ;

}

} else

{ if ( test + exp<sub>3</sub> )

{

Statement 3 ;

}

else

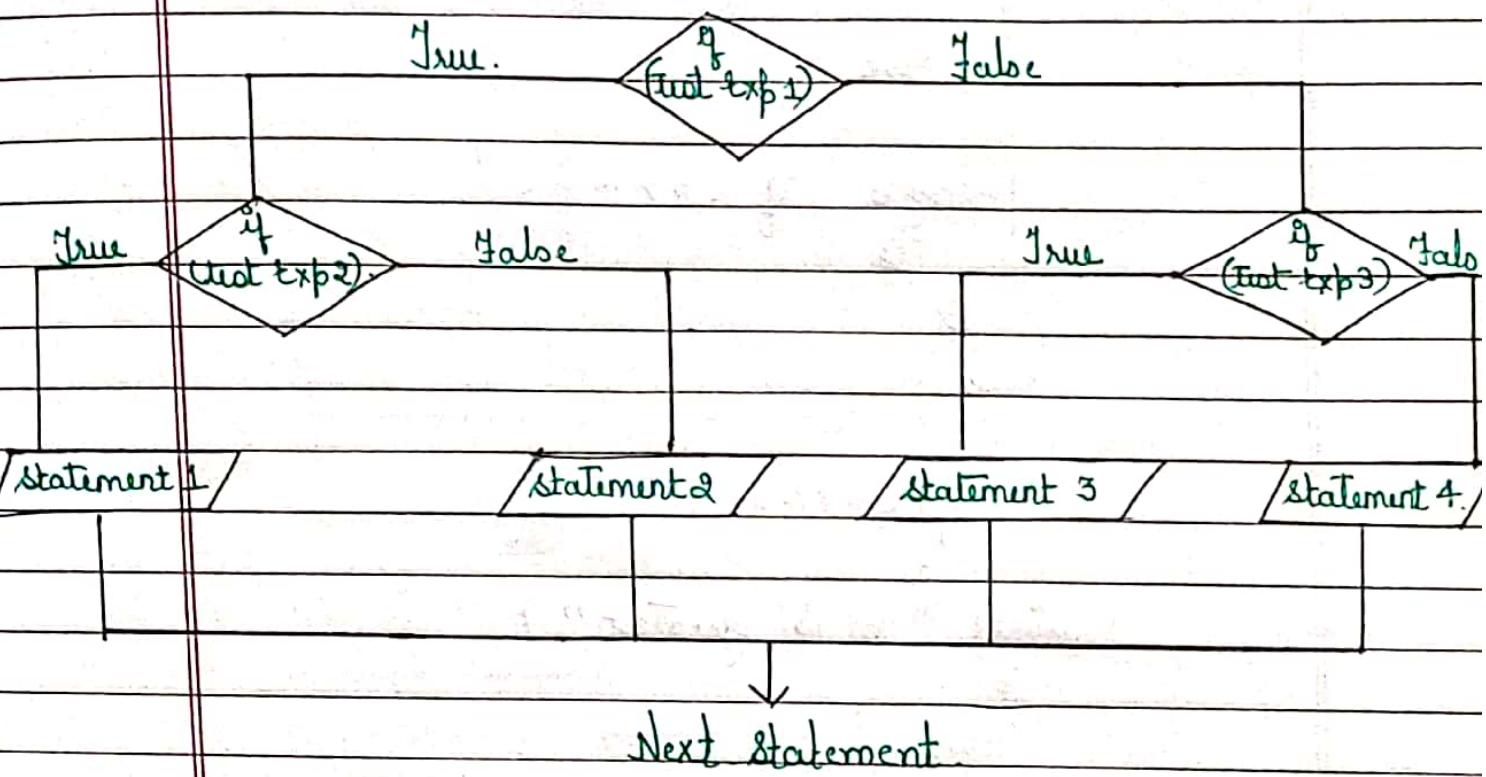
{

Statement 4 ;

}

2

flowchart →



Ques- WAP to find the largest no. among three nos. using nested if-else statement or without using logical operators.

```

#include < stdio.h >
#include < conio.h >
void main()
{
    clrscr();
    int a, b, c;
    printf("Enter the three nos. a, b & c");
    scanf("%d %d %d", &a, &b, &c);
    if (a > b)
    {
        if (a > c)
        {
    
```

```
printf("a is greatest")
```

{

} else

{

```
printf if (b > a).
```

{

else

{

```
if (b > c)
```

{

```
printf("b is greatest")
```

{

else

{

```
printf("c is greatest")
```

{

}

(d) The else if ladder: There is another way of putting if statements together when multipath decisions are involved.

A multipath decision is a chain of ifs in which the statement associated with each if is an if. It takes the following general form. The if ladder is multi-way branching statement. It contains a chain of multiple if statements. It starts by checking the first logical expression. If the exp1 is true,

then the body of the first is executed otherwise  
the next logical expression is evaluated & so on.

Syntax →  $\text{if} (\text{test exp}_1)$   
          {  
            statement 1 ;  
          }

          else if ( $\text{test exp}_2$ )  
          {  
            statement 2 ;  
          }

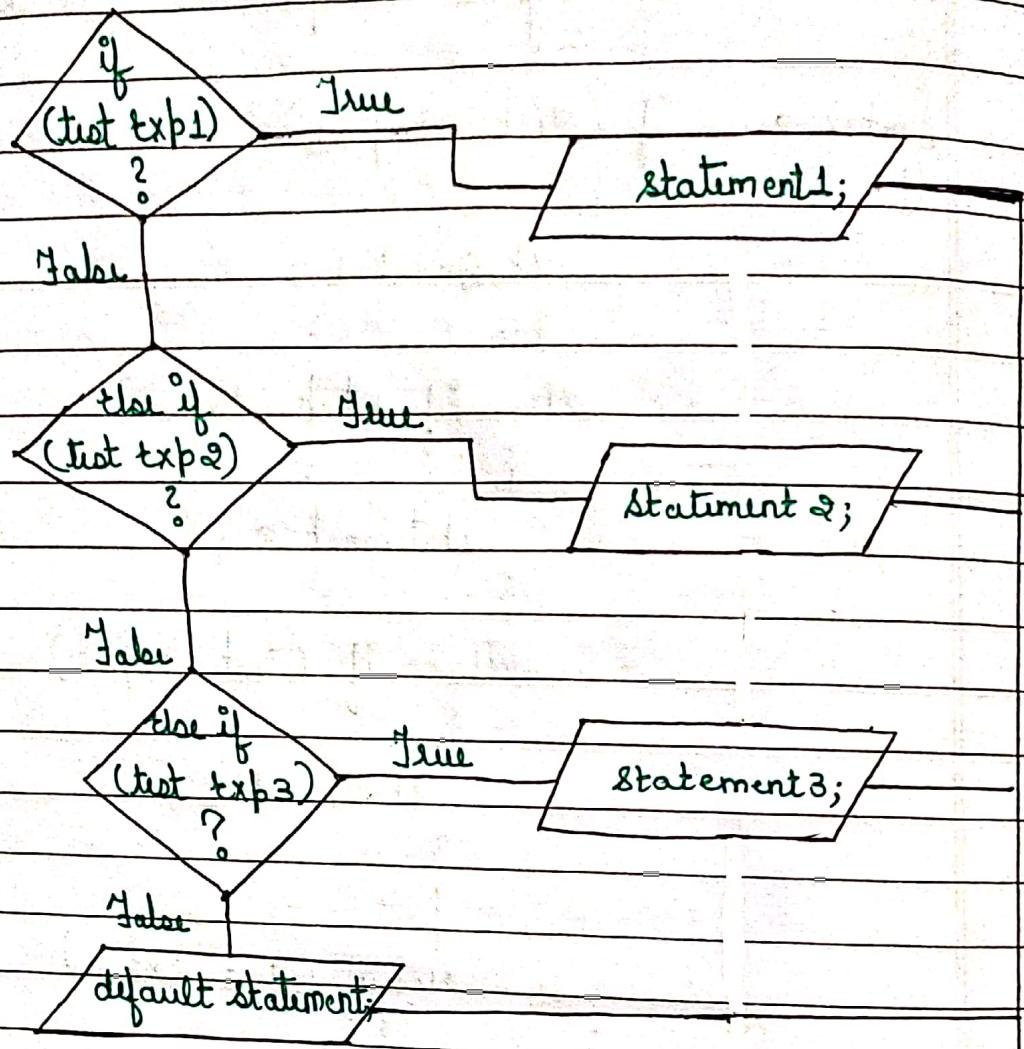
          else if ( $\text{test exp}_3$ )  
          {  
            statement 3 ;  
          }

          else if ( $\text{test exp}_n$ )  
          {  
            statement n ;  
          }

        else

        {  
          default statement ;  
        }

flow chart →



Let us consider an exp of grading to the student in an academic institution. The grading is done according to following rules.

Average Marks	Grade
80 - 100	Honors
60 - 79	I <sup>st</sup> Division
50 - 59	II <sup>nd</sup> Division
40 - 49	III <sup>rd</sup> Division
0 - 39	Fail

```
#include < stdio.h >
#include < conio.h >
void main()
{
    clrscr();
    int marks, grade;
    if (marks > 80)
    {
        printf ("honor");
    }
    else if (marks > 59 && marks < 79)
    {
        printf ("grade = first");
    }
    else if (marks > 49 && marks < 59)
    {
        printf ("grade = second");
    }
    else if (marks > 39 && marks < 49)
    {
        printf ("grade = third");
    }
    else ("marks > 0 && marks < 39")
    {
        printf ("Fail");
    }
    getch();
}
```

Thursday

Date: 24/01/2023 Page: \_\_\_\_\_

(e) - Switch Case Statement :- Switch statement provides means for checking a given expression & selecting the one course of action among various action available.

→ Switch statement consists of two sections

i) This section is identified by switch keywords & contain the expression i.e. to be evaluated.

ii) This section include of multiple case statement which clearly associates all possible outcomes of the expression & the action to be performed in each case.

→ The general syntax & flow chart of the switch statement is -

Syntax →  $\text{switch} (\text{expression}) \{$

Case 1 :

    Statement ;  
    break ;

Case 2 :

    Statement ;  
    break ;

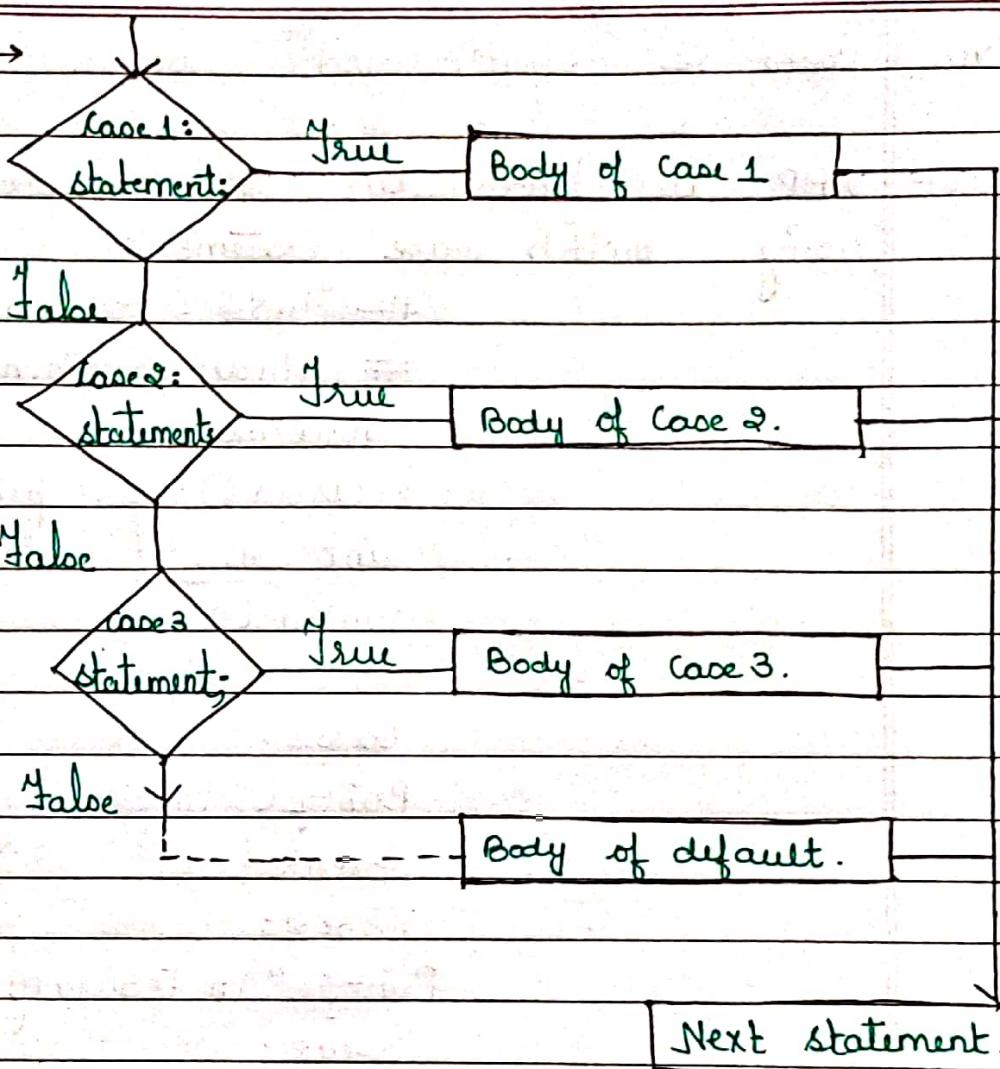
Case 3 :

    Statement ;  
    break ;

default n :

    Statement n ;  
    }

flowchart →



- The following rules should be used for switch statement are :-
- (i) The expression value must be an integer, hence the type can be int or character.
- (ii) Case should also be followed by an int constant, char constant or constant expression.
- (iii) All the cases should be different.
- (iv). The block of statement under default is executed when none of the cases match the value of the expression.
- (v) Default can optionally be present.

(vi) Case & default can occur in any order

Ques- WAP to find out the no. of days in a month using switch case statement.

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

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

```
void main()
```

```
{ clrscr(); } printf("Enter the no.");
```

```
int a; { scanf("%d", &a); }
```

```
switch (a)
```

```
{}
```

Case 1:

```
printf("In January = 31 days");
```

```
break;
```

Case 2:

```
printf("In February = 28 or 29 days");
```

```
break;
```

Case 3:

```
printf("In March = 31 days");
```

```
break;
```

Case 4:

```
printf("In April = 30 days").
```

default:

```
printf("In No choice")
```

```
break;
```

```
}
```

```
getch();
```

```
}
```

Ques:- WAP to find out the week days using switch statement

```

#include < stdio.h >
#include < conio.h >
void main()
{
    clrscr();
    int x;
    printf("Enter the no");
    scanf("%d", &x);
    switch(x)
    {
        case 1:
            printf("In Monday");
            break;
        case 2:
            printf("In Tuesday");
            break;
        case 3:
            printf("In Wednesday");
            break;
        case 4:
            printf("In Thursday");
            break;
        case 5:
            printf("In Friday");
            break;
        default:
            printf("In No choice");
            break;
    } getch();
}

```

Ques: WAP to find the value 'y' for a particular value of 'k'  
 if  $\rightarrow k=1$  then  $y = ax + b$   
 if  $\rightarrow k=2$  then  $y = ax^2 + b^2$ .  
 if  $\rightarrow k=3$  then  $y = ax$ .  
 if  $\rightarrow k=4$  then  $y = a+x$ .

#include < stdio.h >

#include < conio.h >

void main ()

{

clrscr();

int k;

float a, b, x, y;

printf ("Enter the value of a, b, x, k");

scanf ("%f %f %f %d", &a, &b, &x, &k);

switch (k)

case 1:

$y = ax + b;$

break;

case 2:

$y = a * (x * x) + (b * b)$

break;

case 3:

$y = a * x;$

break;

case 4:

$y = (a+x)$

break;

default:

printf ("\\n value is not valid");

```
break;
```

```
}
```

```
getch();
```

```
};
```

Ques-

WAP to calculate the addition, subtraction, multiplication & division of two int no. using switch statement

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

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

```
void main()
```

```
{
```

```
clrscr();
```

```
float a, b, x;
```

```
int k;
```

```
printf ("Enter the value of a & b");
```

```
scanf ("%f %f", &a, &b);
```

```
switch (k)
```

Case 1:

```
x = a + b;
```

```
break;
```

Case 2:

```
x = a - b;
```

```
break;
```

Case 3:

```
x = a * b;
```

```
break;
```

Case 4:

```
x = a / b;
```

```
break;
```

Wednesday

Date: 30/ Oct/ 19  
Page: \_\_\_\_\_

default :

```
printf("In value is not valid")
```

```
break;
```

```
}
```

```
getch();
```

```
}
```

[2]

Repetitive statement or Iterative statements or loops statement that may repeat other statement more than one until a logical statement evaluates to True is called a loop, Repetitive or iterative statement.

C lang provides the following three loops-

while statements

do-while statements

for-loops statements

a. while statements - The simplest of all the looping structure in C is the while statement.

The basic format of while statement is -

Syntax →

```
void main()
```

```
{
```

```
int
```

```
while (test condition)
```

```
{
```

Body of the loop;

```
}
```

→ The while is an entry controlled loop statement

- The test condition is evaluated & if the condition is true then, the body of loop is executed.
- After execution of body, the test condition is once again evaluated & if it is true the body is executed once again. This process of repeated execution of the body continues until the test condition finally becomes false & the control is transferred out of the loop & exist the program continuously with the statement immediately after the body of the loop.

void main()

{

clrscr();

i=1; // initialization or counter variable  
while ( i <= 5 ) // condition

{

printf("In Allmin");

i++; // increment

}

getch();

}

Ques:-

WAP to find out the sum of first 'n' integers (using while loop).

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

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

```
Void main ()
```

```
{
```

```
    clrscr();
```

```
    int i, n, sum = 0;
```

```
    printf ("enter the range");
```

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

$i = 1;$

```
    while (i <= n)
```

```
{
```

```
        sum = sum + i.
```

$i++;$

```
}
```

```
    printf ("%d, sum);
```

```
getch();
```

```
}
```

P	$i \leq n$	$sum = sum + i$	$i++$
1	$1 \leq 9 (T)$	$sum = 0 + 1 = 1$	2
2	$2 \leq 9 (T)$	$sum = 1 + 2 = 3$	3
3	$3 \leq 9 (T)$	$sum = 3 + 3 = 6$	4
4	$4 \leq 9 (T)$	$sum = 6 + 4 = 10$	5
5	$5 \leq 9 (T)$	$sum = 10 + 5 = 15$	6
6	$6 \leq 9 (T)$	$sum = 15 + 6 = 21$	7
7	$7 \leq 9 (T)$	$sum = 21 + 7 = 28$	8
8	$8 \leq 9 (T)$	$sum = 28 + 8 = 36$	9
9	$9 \leq 9 (T)$	$sum = 36 + 9 = 45$	10
10	$10 \leq 9 (F)$	<del>sum =</del>	

Output  $\rightarrow$  enter the range = 9.  
 $sum = 45.$

Ques- WAP To evaluate the factorial value using  
imp- while loop.

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

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

```
void main()
```

```
{
```

```
clrscr();
```

```
int i, n;
```

```
long int Fact = 1;
```

```
Printf (" enter the range ");
```

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

```
i = 1;
```

```
if (n > 1)
```

```
while (i <= n)
```

```
{
```

```
Fact = Fact * i;
```

```
i++;
```

```
}
```

```
printf ("\n Factorial value = %d", fact);
```

```
getch();
```

```
}
```

Ques- WAP To display the result when an integer no. is input through the keyboard. Find out the table of its integer no. using while loop.

Thursday

#include &lt;stdio.h&gt;

#include &lt;conio.h&gt;

void main()

{

clrscr();

int n, i, m;

printf ("Enter int no.");

scanf ("%d", &amp;n);

i = 1;

while (i &lt;= 10)

{

m = n \* i;

printf ("In %d \* %d = %d", n, i, m);

i++

{

getch();

{

Result -

$$m = n * i = 10 * 1 = 10$$

$$m = n * i = 10 * 2 = 20$$

$$m = n * i = 10 * 3 = 30$$

$$m = n * i = 10 * 4 = 40$$

$$m = n * i = 10 * 5 = 50$$

$$m = n * i = 10 * 6 = 60$$

$$m = n * i = 10 * 7 = 70$$

$$m = n * i = 10 * 8 = 80$$

$$m = n * i = 10 * 9 = 90$$

$$m = n * i = 10 * 10 = 100$$

(b). Do-while loop - This is an exit control or a post test loop

Syntax → do

{

body of the loop

while (test-condition);

On reaching the do statement, the program proceeds to evaluate the body of the loop

first. At the end of the loop, the test condition in the while statement is evaluated. If the condition is true, the program continues to evaluate the body of the loop once again.

This process continues as long as the condition is true. When the condition becomes false, the loop will be terminated & the control goes to the statement that appears immediately after the while statement.

Ex:- main ()

{

choice();

i = 1;

do

{

printf ("In Allmin");

i++;

}

while ( i <= 5 );

getch();

}

Ques:- WAP to print sum of n natural no., using do while loop.

```

#include < stdio.h >
#include < conio.h >
void main()
{
    clrscr();
    int i, n, sum = 0;
    printf(" Enter the range ");
    scanf("%d , %n");
    i = 1;
    do
    {
        sum = sum + i;
        i++;
    }
    while (i <= n);
    printf("%d", sum);
    getch();
}

```

Ques- WAP To evaluate the factorial of a integer no  
 imp. using do while loop.

```

#include < stdio.h >
#include < conio.h >
void main()
{
    clrscr();
    int i, n;
    long int Fact = 1;
    printf(" Enter the range ");

```

```

scanf ("%d", &n);
i = 1;
if (n > 1)
do
    {
        fact = fact * i;
        i++;
    }
while (i <= n);
printf ("The factorial value = %d", fact);
getch();
}

```

- (c) For statement - The for loop is another entry controlled loop that provides a more concise loop control structure.
- The general form of the for loop is -
- ```

for( initialization ; test condition ; increment/decrement )
{
    body of the loop
}

```

Ex:-

```

for (i = 1; i <= 5; i++)
{
    printf ("In Allmün");
}
getch();

```

*Today*  
Quest WAP To find out the sum of first n integers using for statement.

void main ()

```
{  
    clrscr ();  
    int i, n, sum = 0;  
    for (i = 1; i <= n; i++)  
    {  
        sum = sum + i;  
    }  
    printf ("\n sum");  
    getch();  
}
```

Quest WAP To evaluate the factorial of an integer n using for loop.

```
#include <stdio.h>  
#include <conio.h>  
void main()  
{  
    clrscr();  
    int fact = 1, a;  
    printf ("Enter a no. for finding factorial");  
    scanf ("%d", &a);  
    if (n > 0)  
        for (int i = 1; i <= a; i++)  
        {  
            fact = fact * i;  
        }  
    printf ("\n factorial = %d", fact);  
    getch();  
}
```

Ques- WAP to display the result when an int no. is input through the keyboard. Find out the table of its int no. using do-while loop.

Date: 4 Nov Page: \_\_\_\_\_

```
#include <stdio.h>
#include <conio.h>
void main()
{
    clrscr();
    int n, i, m;
    printf("Enter int no:");
    scanf("%d", &n);
    i = 1;
    do
    {
        m = n * i;
        printf("\n %d * %d = %d", n, i, m);
        getch();
    } while (i <= 10);
}
```

Ques: WAP to find the sum of series  
 $1 + 2 + 3 + 4 + 5 + \dots + n$ .

```
#include <stdio.h>
#include <conio.h>
void main()
{
    clrscr();
    int i, n, sum = 0;
    for (i = 1; i <= n; i++)
    {
        sum = sum + i;
    }
}
```

Ques5. WAP To calculate fibonacci series 0, 1, 2, 3, 5, 8, 13, ...

```

#include < stdio.h>
#include <conio.h>
void main()
{
    clrscr();
    int a, b, c, i, n;
    printf("Enter the number range");
    scanf("%d", &n);
    printf("%d", a);
    printf("%d", b);
    for (i=0; i<=n; i++)
    {
        c = a+b;
        printf("%d", c);
        a = b;
        b = c;
    }
    getch();
}

```

output →

$c = a + b$   
 $= 0 + 1 = 1$

$c = a + b$   
 $= 1 + 1 = 2$

Ques6. WAP To calculate sum of series  $2^0, 2^1, 2^2, 2^3, \dots, 2^n$ .

```

#include <math.h>
#include <stdio.h>
#include <conio.h>
void main()
{
    clrscr();
    int i, n, sum=0, pow;
    for (i=0; i<=n; i++)
    {
        pow = 2^i;
        sum += pow;
    }
}

```

sum = sum + pow (2, i)

}

getch();

}.

Ques 7:- WAP to print sum of series  $1^4 + 2^4 + 3^4 + 4^4 \dots + 100^4$

#include <Math.h>

#include <stdio.h>

#include <conio.h>

void main()

{

clrscr();

int i, n, sum=0, pow;

for (i=1 ; i<=100; i++)

{

sum = sum + pow (i, 4);

}

getch();

}

Ques 8:- WAP to print sum of series  $1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \dots + \frac{1}{n}$

#include <stdio.h>

#include <conio.h>

#include <Math.h>

void main()

{

clrscr();

int i, n;

float sum=0.0;

Tuesday

Date: 5 Nov Page: \_\_\_\_\_

```
printf ("Enter the no. ");
scanf ("%d", & n);
for (i = 1; i <= 100; i++)
{
    sum = sum + pow (1, 4);
}
printf ("%d", ln sum);
getch();
```

\* Nested loops loops that contain one or more loops inside their body are called nested loop. Each loop must be controlled by a different variable or counter or index.

Syntax →

```
for (exp1; condition 1; exp2)
```

```
{  
    for ( +exp3; condition 2; exp4)  
    {  
        statement 1;  
        statement 2;  
    }  
}
```

Ques 1:- WAP to find the sum of digits

```
#include <math.h>
```

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

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

```
void main()
```

```
{
```

```

class();
int i, j, sum;
for (i=1; i<=5; i++)
{
    for (j=1; j<=5; j++)
        sum = i+j;
    printf (" i = %d |n -> j = %d |n -> %d
            = %d");
```

getch();  
};

Ques:- WAP To print the following

|   |   |   |   |   |
|---|---|---|---|---|
| 1 |   |   |   |   |
| 2 | 2 |   |   |   |
| 3 | 3 | 3 |   |   |
| 4 | 4 | 4 | 4 |   |
| 5 | 5 | 5 | 5 | 5 |
| ! | ! | ! | ! | ! |
| n | - | - | - | - |

# include <math.h>

# include <conio.h>

# include <stdio.h>

void main ()

{

class();

int i, j, n;

printf (" Enter Two no. of rows");

scanf ("%d", &n);

for (i=1; i<=n; i++)
{

for (j=1; j<=i; j++)
};

```

    {
        printf ("%d", i);
    }
}

getch();
}

```

Ques 3:- WAP in c to generate the following -

```

    *
   * *
  * * *
 * * * *
* * * * *

```

```

#include <stdio.h>
#include <conio.h>

void main()
{
    clrscr();
    int i, j, n;
    printf ("Enter the no. of rows");
    scanf ("%d", &n);
    for (i = 1; i <= n; i++)
    {
        for (j = 1; j <= n; j++)
        {
            printf ("%d", * );
        }
    }
    getch();
}

```

Ques: Write a loop in C to generate the following pattern

```

5
5 4
5 4 3
5 4 3 2
5 4 3 2 1.

```

```

#include <stdio.h>
#include <conio.h>
void main()
{
    clrscr();
    int i, j, n;
    printf("Enter the no. of columns");
    scanf("%d", &n);
    for (i = n; j = 1; i--)
    {
        for (j = 1; j <= i; j++)
        {
            printf("%d", i);
        }
        getch();
    }
}

```

# Modular Programming Using FUNCTIONS

---

---

A function groups a number of statements into a single unit and gives it a name. Every C program is a collection of functions. Every C program has atleast one function 'main()', which gets executed first and calls other functions within it.

The functions defined by user according to his/her need are called user defined functions.

"A Function is a portion of code within a larger program, which performs a specific task and is relatively independent of remaining program"

working →

```
void main()
```

```
{
```

```
    Statement-1;
```

```
    Statement-2;
```

```
    Statement-3;
```

```
    function1(); ————— Now execution of function1( )
```

```
    Statement-4;
```

```
    |
```

```
    Statement-n;
```

```
}
```

```
function1 definition
```

```
—————
```

```
control back to main()
```

(Liman)

C functions can be classified into two categories :-

- \* Library functions
- \* User defined functions.

library functions are not required to be coded by users eg:-  
`printf()`, `scanf()`, `sqrt()`, `pow()` etc.

User defined functions are to be coded by users as per the requirement eg:- `main()`, `factorial()`, `add()` etc.

### Need for functions

Every C program must have `main()`, as to indicate where the program has to begin its execution. If we write entire code inside `main`, it leads to a number of problems.

- ✓ program becomes too large and complex
- ✓ Difficult to understand
- ✓ Debugging, testing of program becomes difficult.

So if a program is divided into modules (functions), then each part may be independently coded and later combined into a single unit. These independently coded programs are called subprograms or functions.

Also, many times we want certain operations to be repeated at many places throughout a program. (eg. factorial calculation)

In such cases, either repeat the statements at those places or call respective function where required. This saves time and memory space.

(3)

## Advantages of Using function

(2)

- ① It facilitates Modular programming.
- ② length of source program can be reduced using functions.
- ③ It is easy to locate and isolate faulty function for further investigation
- ④ A complex program can be divided into small subprograms ∴ becomes easy to write and understand.
- ⑤ It enhances reusability ∴ saves time and memory.
- ⑥ functions can be coded independently and then separately compiled into the program.

eg Illustration of function \*/

```
#include <stdio.h>
#include<conio.h>
void display(); /*function declaration or prototype*/
void main()
{
    clrscr();
    printf("Welcome to world of C Programming\n");
    display(); /*function call*/
    printf("Programming is LOGIC");
    getch();
}
void display() /*function definition*/
{
    printf (" @@@@ @@@@\n");
}
```

outputs

|                                   |
|-----------------------------------|
| Welcome to world of C Programming |
| @@@@ @@@@                         |
| Programming is LOGIC              |

Note: main() -

(4)

There is no limit on number of functions that can be present in a program. A function can be called any number of times. The function is called in the sequence specified by function call in main(). When a function have been executed, control returns back to calling function (generally main()). Also, execution begins and ends at main().

### Elements of User defined function

In order to use user defined function, three things are needed →

1. function declaration (Prototype)
2. function call
3. function definition

### Function Declaration / Prototype

Like variables, all functions in a C program must be declared before they are invoked. It informs the compiler that the function as mentioned in declaration would be referenced later in the program.

Syntax → returntype functionname ( parameter list );

#### Note :

- \* parameter list is separated by commas and use of parameter names is optional.
- \* returntype void means function doesn't return any value.
- \* Empty parenthesis means function takes no parameters.
- \* Default return type in C is int.
- \* When there is a mismatch in function declaration and definition, compiler produces error.
- \* It is written above all functions (including main())

eg    int add (int , int);

(5)

void display();

add (int, int)

void display (int);

## Function definition

The function contains code for the function.

Syntax →   
 function Header { returntype functionname (parameter list) }

function Body

} local variable declaration (if any) ;

statement 1;

statement 2;

-----

-----

return statement;

}

Notes ✓ The parameter list is a comma separated list of variables that will receive values sent by calling program. They are known as formal parameter / dummy arguments.

- ✓ function body contains declaration and statements necessary for performing required task.
- ✓ When function reaches return statement, control transfers back to calling program.
- ✓ In absence of return statement, closing brace acts as return.
- ✓ syntax of return :→  
    return;      or      return (variable);

Eg. (6)

(a) void sum(int a, int b)

{

    printf (" sum= %d", a+b);

    return;

        /\* returns nothing, optional \*/

}

(b) void display()

{

    printf (" Welcome");

}

    /\* closing brace acts as return \*/

(c) int mul (int a, int b).

{

    int x;

    x = a \* b;

    return(x); /\* returns value of x to calling program \*/

}

### function Call.

A function can be called by specifying its name, followed by list of arguments enclosed in parenthesis. If no arguments are there, empty parenthesis are given.

The arguments appearing in function call are known as actual parameter, in contrast to formal parameter that appear in function definition.

Also actual argument must match in number, type and order with their corresponding formal arguments.

eg main()  
{  
    int y;  
    y = mul(10, 5); //function call  
    printf("%d", y);  
}

eg display();

sum(10, 20);

Note when a function returns a value, the value is stored  
in a variable of same type.

~~Ex:~~ WAP to find sum of digits of a given number using function

```
int sod(int);
void main()
{
    int num,sum;
    clrscr();
    printf("Enter any number ");
    scanf("%d", &num);
    sum = sod(num); /* function call */
    printf("sum of digits = %d", sum);
    getch();
}
```

```
int sod(int n)
{
    int r,s=0; /* local variables */
    while(n)
    {
        r = n%10;
        n = n/10;
        s = s + r;
    }
    return(s);
}
```

eg WAP to find factorial of a number using functions.

(9)

```
int factorial(int t);  
void main()  
{  
    int num, f;  
    clrscr();  
    printf("enter number");  
    scanf("%d", &num);  
    f = factorial(num);  
    printf("factorial of %d is %d", num, f);  
    getch();  
}
```

```
int factorial (int n)  
{  
    int i, p=1;  
    for(i=n; i>=1; i--)  
    {  
        p=p*i;  
    }  
    return p;  
}
```

~~Q8~~ = WAP to find sum of series  $2^0 + 2^1 + 2^2 + \dots + 2^n$ .  
using functions.

```
int series(int)
{
    void main()
    {
        int n, sum
        printf("Enter value of n");
        scanf("%d", &n);
        sum = series(n);
        printf("sum of series = %d", sum);
        getch();
    }
}
```

```
int series(int n)
{
    int i, s=0;
    for(i=0; i<=n; i++)
    {
        s = s + pow(2, i);
    }
    return s;
}
```

~~Q7~~ WAP to find largest of three numbers using functions. (11)

```
int big (int, int, int);  
void main ()  
{  
    int a, b, c, result;  
    clrscr();  
    printf (" Enter three numbers ");  
    scanf ("%d %d %d", &a, &b, &c);  
    result = max (a, b, c);  
    printf (" Largest = %d ", result);  
    getch();  
}
```

```
int big (int x, int y, int z)  
{  
    if (x > y && x > z)  
        return (x);  
    else if (y > x && y > z)  
        return (y);  
    else  
        return (z);  
}
```

In C, function can be called in one of the two ways.

- \* call by value.

- \* call by reference.

call by Value.

Here actual parameters are evaluated and are copied into formal parameters.  $\therefore$  only a copy of actual parameter is passed. Actual parameter and formal parameter have two different memory location. Any changes made to value of formal parameter is not reflected back in actual parameter value.

eg ~~void~~ swap (int, int);    fun

void main()

{

int a=10, b=20;

printf("Value before call a=%d, b=%d", a, b);

swap(a, b);

printf("Value after call a=%d, b=%d", a, b);

getch();

}

void swap(int x, int y)

{

int temp;

~~temp=x;~~     $x = a$

~~x=y;~~     $a = b$

~~y=temp;~~     $b = c$

printf("Value inside function a=%d, b=%d", x, y);

[ In function call  $\rightarrow$  Actual parameter ]  
 [ In function definition  $\rightarrow$  Formal parameter ]

Output.

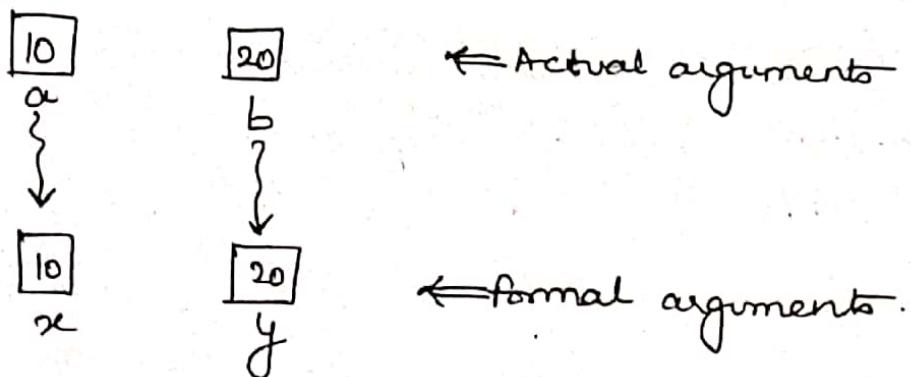
(13)

Value before call  $a=10, b=20$ .

Value inside function  $a=20, b=10$ .

Value after call  $a=10, b=20$ .

Any changes made inside function, affects only copy passed.



after function call  $x=20, y=10$ , but this change is not reflected in actual parameters  $a, b$ .

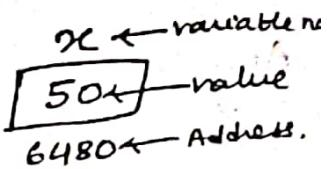
This is CALL BY VALUE.

## Pointer.

when we declare `int x = 50;`

it tells C compiler to =

- i) reserve memory to store integer value
- ii) Give memory location name `x`.
- iii) store value 50 at this location



Compiler will automatically assign a memory cell for `x`. To get the address of `x`, we use operator ampersand (`&`) for getting location of `x`. `&` is known as "address of" operator.

`&x`

Now this address can be stored in some other variable.

`ptr = &x ;`

This `ptr` is not an ordinary variable, but a pointer since it points to location of `x`.

∴ Pointer is a variable that holds address of another variable.

### Advantage of pointer :

- ✓ You can access memory location without using their name.
- ✓ Speed of program execution increases, because accessing with address is faster than name.
- ✓ You can allocate/deallocate memory at runtime as per your need (Dynamic memory allocation)
- ✓ You can return multiple values via function which was not possible earlier
- ✓ You can write more compact & efficient code.

## Declaration of Pointer

Like other variables, pointer variable is also declared. When a pointer variable is declared, variable name must be preceded by \*. The datatype of pointer variable is of the variable it points refers to.

- i.e. if ptr points to integer variable `int *ptr ;`
- if ptr points to float variable `float *ptr ;`
- if ptr points to character variable `char *ptr ;`

Syntax, `[datatype *pointername ;]`

When declared, it is a good programming practice to initialize it to a value.

∴ `ptr = &x ;` (address of x is stored in ptr)

Note :   
`&` is "Address of" operator  
`*` is "Value at" operator

eg.

`int i, *p ;`

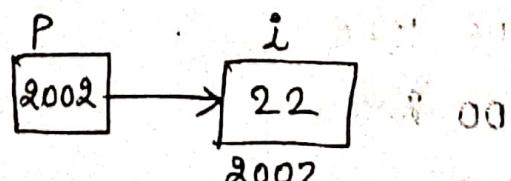
`i = 22 ;`

`p = &i ;`

`printf("%d", i); // 22`

`printf("%d", *p); // 22`

`printf("%d", p); // 2002`



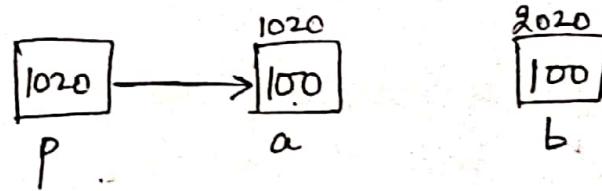
$$*p \equiv *(2002) \equiv 22$$

Eg /\* pointer demo \*/

```

void main()
{
    int a, b, *p;
    clrscr();
    a=100;
    p=&a;
    b=*p;
    printf ("%d", a);
    printf ("%d %d", a, p);
    printf ("%d %d", b, &b);
    printf ("%d", &a);
    printf ("%d", &(*p));
    getch();
}

```

Output:

100  
100 1020  
100 2020  
1020  
1020

$$\ast(\&a) = \ast(1020) \\ = 100$$

## call by Reference

(17)

- Here address of actual arguments is copied into the formal arguments so formal arguments now refers to the actual parameters (alias created). Both called and calling function refer to same memory location so any changes made to variables in called function are reflected inside calling function.

eg

```
void swap(int * , int *)
void main()
{
    int a=10, b=20;
    printf("Value before call a=%d, b=%d", a, b);
    swap(&a, &b);
    printf("Value after call a=%d, b=%d", a, b);
    getch();
}
```

```
void swap(int *x, int *y)
{
```

```
    int temp;
    temp=*x;
    *x=*y;
    *y=temp;
```

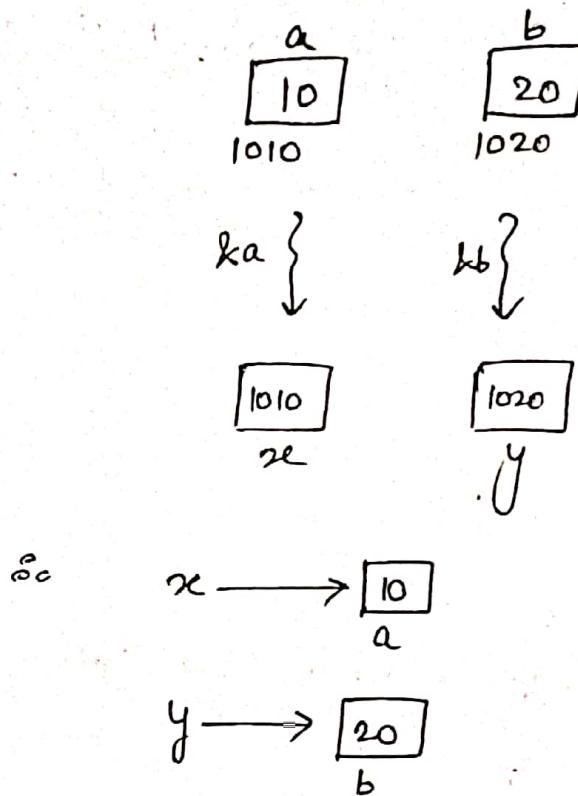
```
    return;
```

```
}
```

Output.

Value before call:  $a=10, b=20$

Value after call:  $a=20, b=10$ .



So  $x$  now points to  $a$ ,  $y$  points to  $b$ .

Any changes made to  $x, y$  are actually manipulating value of  $a, b$ .

Note: If the variable of pointer is incremented then the calculation of next value take place w.r.t. data type of pointer.

int \*p

$p = p + 1$  means  $p = 1020 + 2 = 1022$  because datatype used is int & int takes 2 bytes.

float \*p  $\rightarrow p = p + 1 = 1020 + 4 = 1024$  as float takes 4 bytes.

## Call by Value

- ① Value of variable is passed as argument.
- ② Any alteration in value of argument is not reflected in calling function.
- ③ Memory occupied by actual and formal parameter is different.
- ④ Since new location is created, this method is slow.
- ⑤ Since here we deal with variables so not much programming skill required.

## Call by Reference

- ① Address of variable is passed as argument.
- ② Any alteration in value of argument gets reflected in calling function.
- ③ Memory occupied by actual and formal parameter is same so saves memory.
- ④ Since existing location is used, this method is fast.
- ⑤ Since here we are dealing with addresses so good skill of programming is required.

### ⑥ Example:

```
void swap(int x, int y)
{
    int temp;
    temp = x;
    x = y;
    y = temp;
}
```

### Example:

```
void swap(int *x, int *y)
{
    int temp;
    temp = *x;
    *x = *y;
    *y = temp;
}
```

# Arrays

So far we have used only fundamental datatypes like int, float, char etc. Although these types are very useful, but they are constrained by the fact that these types can store only one value at any given time. However in many applications, we need to handle large amount of data for reading, processing etc. This can be achieved using arrays. "An array is a C derived type that can store several values of one-type".

An array is a collection of homogeneous (same type) elements that are stored in contiguous memory location and are referred by a common name.

e.g.: collection of 10 integers

|           | 1000 | 1002 | 1004 | 1006 | 1008 | 1010 | 1012 | 1014 | 1016 | 1018 | ← Memory address |
|-----------|------|------|------|------|------|------|------|------|------|------|------------------|
| array A ⇒ | 45   | 62   | 12   | 17   | 28   | 56   | 19   | 50   | 77   | 90   |                  |
|           | a[0] | a[1] | a[2] | a[3] | a[4] | a[5] | a[6] | a[7] | a[8] | a[9] |                  |

where 'a' is the name of array containing integers.

for accessing 1<sup>st</sup> element → a[0]

for accessing 5<sup>th</sup> element → a[4]

Number written in square brackets is called Subscript.

Array is also called Subscripted variable as array elements can be accessed using array name and subscript. Subscript starts with 0.

2

The  
Date

Arrays are of following types: →

- \* One dimensional array.
- \* Two dimensional array.
- \* Multidimensional array.

### One-Dimensional Array

A list of items stored under a single name and accessed using one subscript is called 1D array.

#### Declaration of 1D array

Like other variables, arrays must be explicitly declared so that the compiler may allocate space for them in memory.

syntax

datatype arrayname [size];

size defines the maximum number of elements that can be stored inside array.

e.g: int arr[5];

declared an integer array containing maximum 5 elements.

In C, array index begins with 0.

∴  $a[0] \rightarrow 1^{\text{st}}$  element

$a[1] \rightarrow 2^{\text{nd}}$  element

$a[2] \rightarrow 3^{\text{rd}}$  element

$a[3] \rightarrow 4^{\text{th}}$  element

$a[4] \rightarrow 5^{\text{th}}$  element.

Size is  $5 \times 2 = 10$  bytes

subscript is from 0 to  $(n-1)$ , where n is total number of

(3)

The amount of storage required to store an array is directly related to it's datatype and size.

$$\therefore \text{Total bytes} = \text{sizeof(datatype)} \times \text{size of array}$$

eg `int arr[5];`

$$\text{Total bytes} = 2 \times 5 = 10 \text{ bytes.}$$

### Initialization of 1D array

After an array is declared, it's element must be initialized. Otherwise, they will contain "garbage" value. An array can be initialized at either of following stages :-

- \* At compile time
- \* At run time.

### Compile time initialization

Initialization at the time of declaration is known as compile time initialization.

eg `int arr[5] = { 55, 90, 17, 88, 36 };`

Here while initialization, size of array is optional.

eg `int arr[] = { 55, 90, 17, 88, 36 }.`

Here compiler will count the number of elements and reserves space accordingly.

eg `int number[5] = { 10, 20, 30 };`

Here remaining extra elements are initialized to '0'.

.. 0 .. 0 .. 0 .. 0 .. 0 ..

(9)

## Run time Initialization.

Initialization of array while execution of program is called run time initialization.

```
eg int a[5], i;
for (i=0; i<5; i++)
{
    scanf ("%d", &a[i]);
}
```

Note: C doesn't perform bound checking so it is the duty of programmer to check that the numbers entered must not exceed size of array.

Advantage of using arrays: In array we can easily use a loop and repeat same action on each element of array, which is not possible with separate variables so array simplifies our code. It is used for concise and efficient programming.

```
int a[10]; i; sum = 0, float avg;
means Enter 10 value by user
if (printf ("Enter the 10 number"),
for (i=0; i<10; i++)
    scanf ("%d", &a[i]),
for (i=0; i<10; i++)
    sum = sum + a[i]           sum = 0 + a[0]
                                = 0
    avg = sum / 10.0;
    printf ("Average is %.2f", avg);
    getch();
```

(5)

e.g. WAP to read array elements and print them.

```

void main()
{
    int i, a[10];
    printf(" Enter array elements\n");
    for (i=0; i<10; i++)
    {
        scanf("%d", &a[i]);
    }

    printf(" Given array is:\n");
    for (i=0; i<10; i++)
    {
        printf("%d", a[i]);
    }

    getch();
}

```

Output.

Enter array elements

4 2 7 10 8 5 6 15 20 17

Given array is:

4 2 7 10 8 5 6 15 20 17

(6)

~~Ques~~ WAP to find sum and average of n numbers

```
void main()
{
```

```
    int a[50], i, n;
    float avg, sum = 0;
    printf ("Enter value of n");
    scanf ("%d", &n)
    printf ("Enter elements");
    for (i=0; i<n; i++)
    {
        scanf ("%d", &a[i]);
    }
```

*/\* Finding sum and average \*/*

```
    for (i=0; i<n; i++)
    {
        sum = sum + a[i];
    }
```

avg = sum/n;

printf ("Sum = %f", sum);

printf ("Average = %f", avg);

```
    getch();
}
```

Output

Enter value of n

4

Enter elements 4 6 5 9

Sum = 24.000000

~~Q8~~ WAP to print total positive numbers, negative numbers, zero in an array.

void main()

{

int i, a[10], p=0, n=0, z=0;

for (i=0; i<10; i++)

{

printf("Enter any number");

scanf("%d", &a[i]);

}

for (i=0; i<10; i++)

{

if (a[i]>0)

p++;

else if (a[i]<0)

n++;

else

z++;

}

printf(" Positive=%d, Negative=%d, Zeros=%d", p, n, z);

getch();

Output:

Enter numbers.

10 -20 4 0 17 -6 -5 0 -7 1

Positive=4, Negative=4, Zeros=2

(9)

WAP to perform linear search in an array.

void main()

{

int i, n, a[100], key, flag=0;

printf("Enter number of elements");

scanf("%d", &n);

printf("Enter array elements");

for(i=0; i<n; i++)

{

scanf("%d", &a[i]);

}

printf("Enter number to be searched");

scanf("%d", &key);

for(i=0; i<n; i++)

{

If (a[i]==key)

{ printf("Search successful");

printf("number found at location %d",  
(i+1));

flag=1;

}

}

If (flag==0)

printf("Search unsuccessful");

}

## Two Dimensional Array

(17)

C supports multidimensional arrays as well. The simplest form of multidimensional array is the two dimensional array. The two dimensional array is also called a matrix.

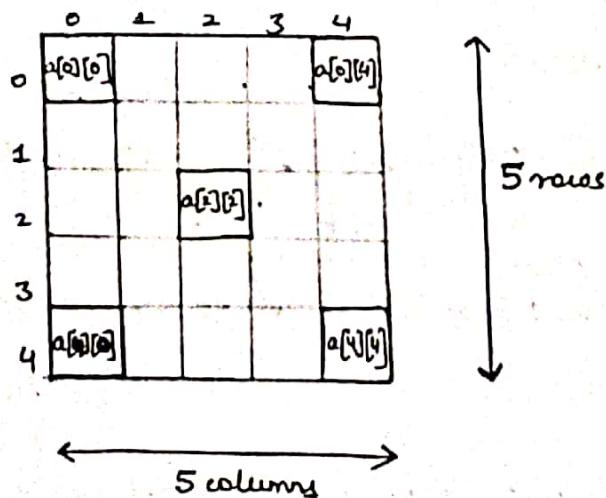
A two dimensional array is a grid having rows and columns in which each element is specified using two subscripts.  
The first subscript tells the row number, whereas second subscript tells the column number. Remember, the counting of rows and columns begins with zero.

### Declaration of 2D Array

Syntax  $\Rightarrow$  datatype arrayname [no. of rows] [no. of columns];

e.g. int a[5][5];

Here 'a' is name of array of type int of size 5 by 5. It will have 5 rows and 5 columns.



The array elements are  $a[0][0]$ ,  $a[0][1]$ ,  $a[0][2]$  .....  $a[4][4]$ .

Memory allocated  $\Rightarrow 5 \times 5 \times 2 \Rightarrow 50$  bytes.

In General

If array is  $a[m][n]$

- ✓ It will have  $m$  rows.
- ✓ It will have  $n$  columns.
- ✓ Element will be of form  $a[i][j]$ , where

$$0 \leq i \leq (m-1)$$

$$0 \leq j \leq (n-1)$$

Total bytes needed to store 2D array is  $\Rightarrow$

$$m \times n \times \text{sizeof(datatype)}$$

Initialization of 2D array

Two dimensional array is array of arrays. (collection of arrays)

Initialization can be done in two ways :

- \* at compile time
- \* at run time

compile time initialization :

```
int stud [4][2] = { {1234, 56}, {1212, 33}, {1434, 80},
                    {1312, 78} };
```

Here each subarray is enclosed in braces and separated by commas.

Alternative way :

```
int stud [4][2] = { 1234, 56, 1212, 33, 1434, 80, 1312, 78 };
```

But here it is mandatory to mention the second dimension, whereas first dimension is optional.

`int a[2][3] = { { 12, 34, 23 }, { 45, 56, 60 } }` ✓ correct

`int a[2][3] = { 12, 34, 23, 45, 56, 60 }` ✓ correct "

`int a[ ][3] = { 12, 34, 23, 45, 56, 60 }` ✓ correct

`int a[2][ ] = { 12, 34, 23, 45, 56, 60 }` ✗ incorrect

`int a[ ][ ] = { 12, 34, 23, 45, 56, 60 }` ✗ incorrect.

Above initialization would result in following Memory map

| <code>a[0][0]</code> | <code>a[0][1]</code> | <code>a[0][2]</code> | <code>a[1][0]</code> | <code>a[1][1]</code> | <code>a[1][2]</code> |
|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|
| 12                   | 34                   | 23                   | 45                   | 56                   | 60                   |

1000      1002      1004      1006      1008      1010

Run time initialization.

for initializing arrays at run time, we have to take value of array elements from user. The array elements can be inputted using nested loop (one outer loop take care of row number, inner loop take care of column number).

```
void main()
{
    int i, j, m, n, a[10][10];
    printf("Enter the dimension of 2D array");
    scanf("%d %d", &m, &n);
    printf("Enter array elements");
    for(i=0; i<m; i++) // for rows
    {
        for(j=0; j<n; j++) // for columns
        {
            scanf("%d", &a[i][j]);
        }
    }
}
```

```

/* printing elements of array */
printf("Entered array is: ");
for(r=0; r<m; i++)
{
    for(j=0; j<n; j++)
        printf("%d", a[r][j]);
    printf("\n");
}
}
//end of main.

```

Output.

Enter dimensions of 2D array

3 2

(m=3, n=2)

Enter array elements

10 20 30 40 50 60.

(a[0][0], a[0][1], a[1][0], a[1][1], a[2][0], a[2][1])

Entered array is:

|    |    |
|----|----|
| 10 | 20 |
| 30 | 40 |
| 50 | 60 |

(21)

WAP to print diagonal elements of a matrix - (square)

```

void main()
{
    int a[10][10], i, j, n;
    printf("Enter order of square matrix ");
    scanf("%d", &n);
    printf("Enter array elements ");
    for (r=0; r<n; r++)
    {
        for (j=0; j<n; j++)
        {
            scanf("%d", &a[r][j]);
        }
    }
    printf("Entered Matrix is : ");
    for (r=0; r<n; r++)
    {
        for (j=0; j<n; j++)
        {
            printf("%d", a[r][j]);
        }
        printf("\n");
    }
    printf("Diagonal Elements are :- ");
    for (r=0; r<n; r++)
    {
        for (j=0; j<n; j++)
        {
            if (r==j)
                printf("%d", a[r][j]);
        }
    }
    getch();
}

```

Output.

Enter order of square matrix

4

Enter array elements

10 5 20 15 30 25 40 35 50 45 60 55 70 65 80 75

Entered Matrix is :

|    |    |    |    |
|----|----|----|----|
| 10 | 5  | 20 | 15 |
| 30 | 25 | 40 | 35 |
| 50 | 45 | 60 | 55 |
| 70 | 65 | 80 | 75 |

Diagonal elements are :-

10 25 60 75

Q: WAP to add elements of two matrices of same order.  
(3x3)

```
void main()
```

```
{
```

```
int a[3][3], b[3][3], c[3][3], i, j;
```

```
printf ("Enter elements of first Matrix ");
```

```
for (i=0; i<3; i++)
```

```
{
```

```
for (j=0; j<3; j++)
```

```
{
```

```
scanf ("%d", &a[i][j]);
```

```
}
```

```
}
```

```
printf ("Enter elements of second Matrix ");
```

```
for (i=0; i<3; i++)
```

```
{
```

```
for (j=0; j<3; j++)
```

```
{
```

```
scanf ("%d", &b[i][j]);
```

```
}
```

```
}
```

```
printf ("Addition of above two matrices : ");
```

```
for (i=0; i<3; i++)
```

```
{
```

```
for (j=0; j<3; j++)
```

```
{
```

```
c[i][j] = a[i][j] + b[i][j];
```

```
printf ("%d", c[i][j]);
```

```
{
```

```
printf ("\n");
```

```
getch();
```

Scanned with CamScanner

Output.

Enter elements of first Matrix

1 2 3 4 5 6 7 8 9 .

Enter elements of second Matrix

10 20 30 40 50 60 70 80 90 .

Addition of above two matrices:

|    |    |    |
|----|----|----|
| 11 | 22 | 33 |
| 44 | 55 | 66 |
| 77 | 88 | 99 |

e.g. WAP to print **TRANSPOSE** of a matrix.(3x3)

```
void main()
{
```

```
    int a[3][3], i, j;
    printf("Enter elements of 3x3 Matrix");
    for (i=0; i<3; i++)
    {
        for (j=0; j<3; j++)
        {
            scanf("%d", &a[i][j]);
        }
    }
```

```
/* printing Matrix */
printf("Entered Matrix is : ");
for (i=0; i<3; i++)
{
    for (j=0; j<3; j++)
    {
        printf ("%d", a[i][j]);
    }
    printf("\n");
}
```

```
printf(" Transposed Matrix is : ");
```

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

```
getch();
```

Output.

Enter elements of  $3 \times 3$  Matrix

10 4 6 5 7 11 15 20 13

Entered Matrix is :

|    |    |    |
|----|----|----|
| 10 | 4  | 6  |
| 5  | 7  | 11 |
| 15 | 20 | 13 |

Transposed Matrix is :

|    |   |    |
|----|---|----|
| 10 | 5 | 15 |
| 4  | 7 | 20 |
| 6  | 7 | 13 |

WAP to check whether matrix is identity matrix or not.  
(3x3)

Void main()

{

```
int a[3][3], i, j, flag=0;
printf ("Enter Matrix elements");
for (i=0; i<3; i++)
{
    for (j=0; j<3; j++)
    {
        scanf ("%d", &a[i][j]);
    }
}
```

printf ("Entered matrix is : ");

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

//check for Identity Matrix..

```
for (i=0; i<3; i++)
{
    for (j=0; j<3; j++)
    {
        if (i==j)
        {
            if (a[i][j]==1)
                continue;
            else
                flag=1;
                break;
        }
    }
}
```

```

else
{
    if ( $a[i][j] \neq 0$ )
        continue;
    else
    {
        flag = 1;
        break;
    }
}

if (flag == 0)
    printf(" Identity Matrix");
else
    printf(" Not an Identity Matrix");
getch();

```

Output:

Enter Matrix elements :-  
 1 0 0 0 1 0 0 0 1  
 Entered Matrix is :-  
 1 0 0  
 0 1 0  
 0 0 1

Identity Matrix.

Enter Matrix elements :-  
 10 0 0 0 20 30 0 0 4  
 Entered Matrix is :-  
 10 0 0  
 0 20 30  
 0 0 4  
 Not an Identity Matrix

Q. WAP to multiply Two matrices of  $n \times n$  dimension.

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

void main()
{
    int i, j, k, n, a[5][5], b[5][5], c[5][5];
    printf(" Enter dimension of square matrices ");
    scanf("%d", &n);
    printf(" Enter Matrix A of order %d * %d ", n, n);
    for (i=0; i<n; i++)
    {
        for (j=0; j<n; j++)
        {
            scanf("%d", &a[i][j]);
        }
    }
    printf(" Enter Matrix B of order %d * %d ", n, n);
    for (i=0; i<n; i++)
    {
        for (j=0; j<n; j++)
        {
            scanf("%d", &b[i][j]);
        }
    }
    /* matrix multiplication */
    printf(" AxB is = ");
    for (i=0; i<n; i++)
    {
        for (j=0; j<n; j++)
        {
            c[i][j] = 0;
            for (k=0; k<n; k++)
            {
                c[i][j] = c[i][j] + a[i][k] * b[k][j];
            }
        }
    }
    printf("%d", c[i][j]);
}
```

(30)

Output

Enter dimension of square matrix

3

Enter Matrix A of order 3x3

1 2 3 0 0 1 1 2 0

Enter Matrix B of order 3x3

2 0 1 1 2 3 2 3 1

$$A \times B \text{ is } \begin{bmatrix} 10 & 13 & 10 \\ 2 & 3 & 1 \\ 4 & 4 & 7 \end{bmatrix}$$

concept

$$\begin{bmatrix} 1 & 2 & 3 \\ 0 & 0 & 1 \\ 0 & 2 & 0 \end{bmatrix} \times \begin{bmatrix} 2 & 0 & 1 \\ 1 & 2 & 3 \\ 2 & 3 & 1 \end{bmatrix} = \begin{bmatrix} 1 \times 2 + 2 \times 1 + 3 \times 2 & 1 \times 0 + 2 \times 2 + 3 \times 1 & 1 \times 1 + 2 \times 3 + 3 \times 0 \\ 0 \times 2 + 0 \times 1 + 1 \times 2 & 0 \times 0 + 0 \times 2 + 1 \times 3 & 0 \times 1 + 0 \times 3 + 1 \times 0 \\ 0 \times 2 + 2 \times 1 + 0 \times 2 & 1 \times 0 + 2 \times 2 + 0 \times 3 & 1 \times 1 + 2 \times 3 + 0 \times 1 \end{bmatrix}$$

## Structures

(1)

### Properties

- \* It is a collection of one or more variables of different datatypes, grouped together under a single name.
- \* It is a user defined datatype.
- \* It groups related information together and treats them as a unit.  
e.g all information related to an employee can be stored in a structure (employee name, employee number, designation, department)
- \* Structure members are stored in contiguous memory location.

### Declaration of a structure

#### Syntax,

```
struct structurename
```

```
{
```

```
datatype member1;
```

```
datatype member2;
```

```
-----
```

```
-----
```

```
datatype membern;
```

```
} structureVariable ;
```

- Here members can be ordinary variables, array or other structures.
- Structure Variable is a variable of type structure.
- It can also be declared as

```
struct structurename
```

```
{
```

```
datatype member1;
```

```
datatype member2;
```

```
-----
```

```
-----
```

```
-----
```

```
}
```

```
struct structurename structureVariable ;
```

(2)

eg

struct employee

{

int empno;

char name[30];

char designation[25];

char dept[20];

{ e1 ;

Here e1 is a structure variable.

Memory allocated to e1 will be  $2 + 30 + 25 + 20 = 77 \text{ bytes}$ .

e1 can also be declared as →

struct employee

{

int empno;

char name[30];

char designation[25];

char dept[20];

};

struct employee e1;

Note: Memory gets allocated only when variable is declared not when structure is defined.Note: Structure is usually defined globally, before main.eg: struct book

{

char title[25];

char author[20];

int pages;

float price;

... etc.

or struct book b1, b2;

| Array                                                                                                                                 | Structure                                                                                                                      |
|---------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------|
| It is collection of homogeneous elements.                                                                                             | It is collection of heterogeneous elements.                                                                                    |
| It is a derived datatype.                                                                                                             | It is user defined datatype.                                                                                                   |
| Array cannot contain structure in it.                                                                                                 | Structure can contain array in it.                                                                                             |
| It is used for storing same type elements.<br>e.g. <code>int arr[10];</code><br><code>float a[20];</code><br><code>char x[30];</code> | It is used for storing database<br>e.g. <u>struct student</u><br>{<br>int rollno;<br>char name[30];<br>};<br>struct student s; |

### Structure Initialization

Like variables, arrays, structure variable can also be initialized where they are declared.

```
eg. struct employee
{
    int empno;
    char name[30];
    char designation[25];
    char dept[20];
};
```

```
struct employee e1 = {5001, "Scott Daniel", "Manager", "IT"};
```

5001 is empno, Scott Daniel is name, Manager is designation, IT is dept of employee e1.

(4)

## Referencing Structure Members / Elements

Individual structure element can be referenced by using dot operator (.) and name of structure variable.

Syntax      structureVariable . membername.

Structure member themselves are not variables. They must be linked to a structure variable using dot operator (.)

eg:

e1.empno;

e1.name

e1.designation

e1.dept

REFERENCE OF STRUCTURE

## Structure Assignment

The information contained in one structure variable may be assigned to another structure variable of same type by using assignment statement.

eg. struct employee

{

int empno;

char sname[20];

char designation [25];

char dept [20];

} e1, e2;

..... and members we can use

107-011

Q.1. WAP to read an employee record using structure & print it.

```

#include <stdio.h>
#include <conio.h>

struct employee
{
    int empno;
    char name[30];
    char designation[25];
    char dept[20];
} emp;

void main()
{
    clrscr();
    printf(" Enter employee record \n ");
    printf(" Employee number: ");
    scanf("%d", &emp.empno);
    printf(" Employee name: ");
    gets(emp.name);
    printf(" Designation: ");
    gets(emp.designation);
    printf(" Department: ");
    gets(emp.dept);
    printf(" Entered record is as follows : \n ");
    printf(" Employee number = %d ", emp.empno);
    printf(" Name = %s ", emp.name);
    printf(" Designation = %s ", emp.designation);
    printf(" Department = %s ", emp.dept);
    getch();
}

```

~~Q. If you want to copy a structure to another variable, how will you do it?~~

```

#ifndef <stdio.h>
#include <conio.h>

struct employee
{
    int empno;
    char name[30];
    char designation[25];
    char dept[20];
} e1, e2;

void main()
{
    clrscr();
    printf("Enter employee 1 records \n");
    printf("Employee number: ");
    scanf("%d", &e1.empno);
    printf(" Name: ");
    gets(e1.name);
    printf("Designation: ");
    gets(e1.designation);
    printf("Dept: ");
    gets(e1.dept);

    e2 = e1 /* Assign e1 to e2 */
    printf("Records of employee 2 are: ");
    printf("Employee number: %d", e2.empno);
    printf(" Name: %s", e2.name);
    printf("Designation: %s", e2.designation);
    printf("Deprtment: %s", e2.dept);
    getch();
}

```

## Array of Structures

Suppose we want to maintain record of 100 employees working in a company. One way is to create 100 variables of structure employee. Another way would be to create an array of structure variable.

eg.

```
struct employee
{
    int empno;
    char name[30];
    char designation[25];
    char dept[20];
} emp[100]; /* array of structure */
```

Subscript will start from 0 i.e.  $\text{emp}[0] \rightarrow$  first employee  
 $\text{emp}[1] \rightarrow$  second employee  
 $\vdots$   
 $\text{emp}[99] \rightarrow$  Hundredth employee

- \* We can read and print the value of structure members using a loop.

Q. Declare a structure which contains given below members and write a program to list all students who scored more than 75 marks.  
 members → rollno, father's name, name, age, city, marks.

```
#include <stdio.h>
#include <conio.h>
struct student
{
    int rollno;
    char name[30];
    char fname[30];
    int age;
    char city[20];
    int marks;
} stud[20]; /* Assuming 20 students are there */
void main()
{
    int i, count = 0;
    for(i=0; i<20; i++)
    {
        printf(" Enter record of student %d ", (i+1));
        printf(" Roll no: ");
        scanf("%d", &stud[i].rollno);
        printf(" Name: ");
        gets(stud[i].name);
        printf(" Father's Name: ");
        gets(stud[i].fname);
        printf(" Age: ");
        scanf("%d", &stud[i].age);
        printf(" City: ");
        gets(stud[i].city);
        printf(" Marks: ");
        scanf("%d", &stud[i].marks);
    }
}
```

/\* searching students who scored more than 75 \*/

for (i=0; i<20; i++)

{

if [stud[i].marks > 75]

{

```
printf(" Roll no: %d", stud[i].rollno);
printf(" Name = %s", stud[i].name);
printf(" Father's Name = %s", stud[i].fname);
printf(" Age: %d", stud[i].age);
printf(" City = %s", stud[i].city);
printf(" Marks: %d", stud[i].marks);
count++;
```

}

}

if (count == 0)

printf(" No student scored more than 75 marks");

getch();

}

(10)

Q. Declare a structure train-info having members, Train no., train name, source station, end station, type.

```
struct train-info
{
    int trainno;
    char trainname[30];
    char source [20];
    char destination [20];
    char type [30];
} t1 = { 4526, "Shatabdi", "Kanpur", "New Delhi",
        "superfast" };
```

$$\begin{array}{r}
 6721 \\
 1010 \\
 0101 \\
 \hline
 6421 \\
 -5
\end{array}$$