Unit - I

Date: 26/08/19 Page: 1.

INTRODUCTION OF COMPUTER.

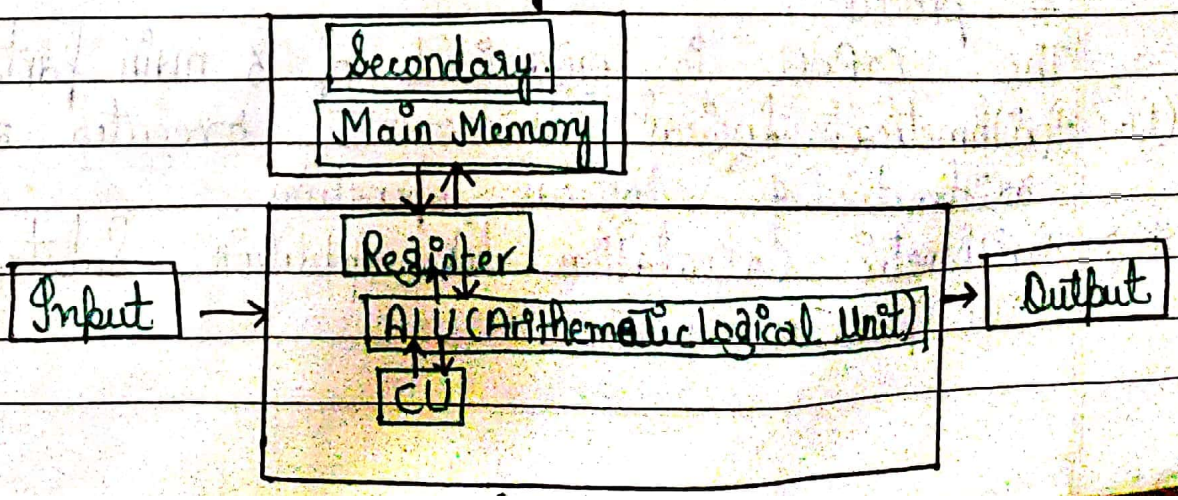
* **Computer :-** A Comp is an electric device, operating under the control of instruction stored in it's own memory that can accept (input), process the data according to specified rules, produce information (output) & store the information for future use.

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

* **Functionality of a Computer :-** Any digital Comp carries out 5 functions in broad 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 information
4. Generates the Output
5. Controls all the above 4 steps.

* **Components of a Comp System :-**



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

★ HARD WARE :- Comp hardware is the collection of physical Element that compute a Computer System

→ Comp hardware refers to the physical parts or components of a computer such as the monitor, mouse, keyboard, 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 an 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 (C·P·U) :- The CPU is the Brain of a Computer. It is responsible for all function & process.

The C·P·U is organised of 3 main parts -

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

*→ Arithmetic Calculation like Addition, Substraction, Multiplication & Division

*→ Logical operation like Comparing nos, letters or special charaters.

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

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

o→* 1° Memory / Main Main memory / Internal Memory :- The memory that is directly connected to the Processor & is called 1° Memory. It is used to store the data i.e cussently in use. It is the fastest memory. There are ~~three~~ two major types of 1° memory -

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

* Types of RAM :- Types of RAM are as followed.
(1) Dynamic RAM (DRAM) & (2) Static RAM (SRAM).

(2). ROM (Read Only Memory) :- This memory reads only the programe stored ~~it~~ 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 store data in the memory structure. It comes in b/w CPU register & RAM. The size of Cache memory is very small incomparision with RAM & it is very high cost.

→ * 2° Memory Or Auxiliary Memory / External Memory :- The memory i.e is not directly connected with the processor is called 2° memory.
2° Memory is used to store the data i.e is currently not in use. 2° 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 processing carried by an information processing System Eg:- Speaker, Projectors
Types of Output Device :-
1. CRD (Cathode Ray Tube)
2. LCD (Liquid Crystal Display)
3. LED (Light Emmiting Diode)

* Printers :- These are of following types :-
(1) Laser Printer :- (2) Ink jet Printer (3) Dot Matrix printer

MEMORY

- 1° memory
  - Cache Memory
  - Main Memory
    - (Random Access Memory) RAM
      - (Flip Flop) → SRAM (Systamatic RAM)
      - (Capacitor & Transistor) → DRAM (Dynamic RAM)
    - (Read Only Memory) ROM
      - → PROM (Programmable ROM)
      - → EPROM (UV Rays). Erasable Programable ROM
      - → EEPROM (Electricity). Electricity Erasable Programable ROM.
- 2° memory.

* 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 support the functionality of the comp system, manage the hardware resources & perform required. for inform processing task is called a System Software.

→ It has two types :-
(1) Operating System (2). Software development is System based.

Date: _____ Page: _____

(i) Operating System :- The software that provides the interface b/w user & hardware is called an 'operating System'. Exp:- 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 exp:- Compiler & Interpreter.

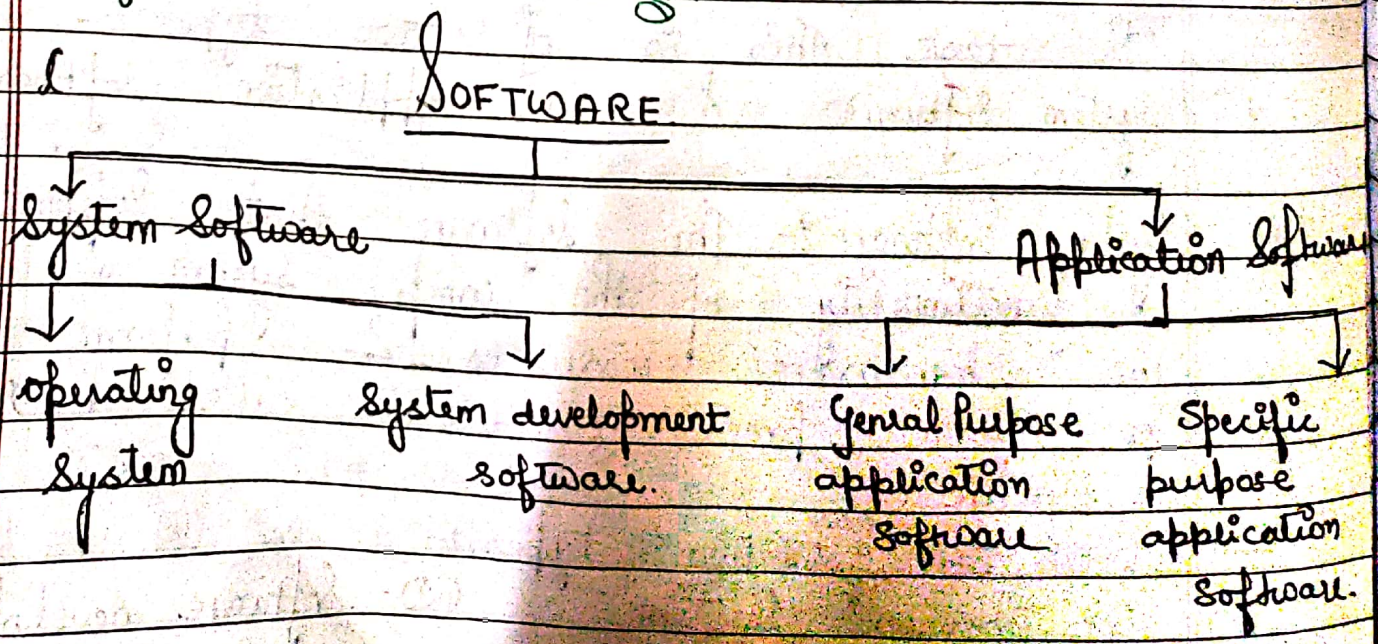
[B] Application Software :- The Software that facilities the user are called 'Application Software'

→ It is of two types. :-

(i) Genral Purpose Application Software :- These Software application are designed for Genral purpose Note pad, MS-Excel, Ms-word

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

★



Date: _____ Page: _____

* Classification of the Computer :- The classification of comps 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, needs a large room.
→ Used for complex Scientific ~~publications~~ applications.
→ High cost, able to handle large amount of data, high power consumption, large & fast memory (1° & 2° memory).
→ Uses multiprocessing & parallel processing & supports multiple programming.
→ Exp:- Super Comp help in analysing weather pattern, light pollution, global warming & the depletion of the earth Ozone layer.

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 organisation such as Railways, Banks, hospitals & insurance Companies.
→ Supports many input & output & storage devices & network of terminals.

Date: _____ Page: _____

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

③. Mini Computers:- Peforms better than micro comps & medium size comps.
→ Design to support more than one user at a time
→ Passes large storage capacity & operators at higher speed.
→ for Exp:- 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.
→ Exp:- Desktop Models, Notebook 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 digits, i.e a combination of 0 & 1 is a 'Digital Comp'
→ for Exp:- Personal comps.

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

Date: _____ Page: _____

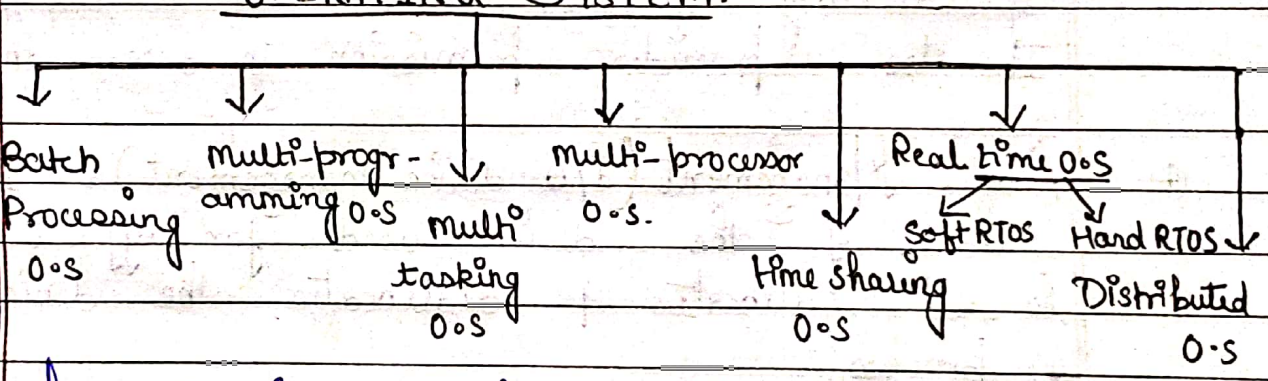
→ for expl- Speedometer, Thermometer & Voltmeter.

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

→ for expl- ICUs

* Operating System :- 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 & so also manage the responsible performing overall operation of the comp. system.

→ It provides services such as user interface, file & data base access & interface to communicatn System such as internet protocal.

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

Date: _____ Page: _____

(1). Process Management :- The OoS is responsible to allocate & de-allocate process to the processor (CPU). It keeps track of the process.

(2). Memory Management :- The OoS 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 & maintainence 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 bodis -

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

Date: _____ Page: _____

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

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

In Batch System, CPU sits ideal 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 DOS / Time Sharing DOS :- Multitasking is an extension of multiprogramming DOS. 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 DOS / Parallel DOS :- Multiprocessor DOS refers to the use of Two or more CPU with in a single comp System.

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

These System are refered as tightly coupled System.

Date: _____ Page: _____

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 forcasting & so on. Exp:- LINUX, UNIX.

(5). Real time DOS:- The real time DOS is used for a real time applicatn 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 pirnited 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 given deadline. It can be crossed if they executed next, without harming the system. Exp:- Mobile phones, Digital Camera & so on.

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

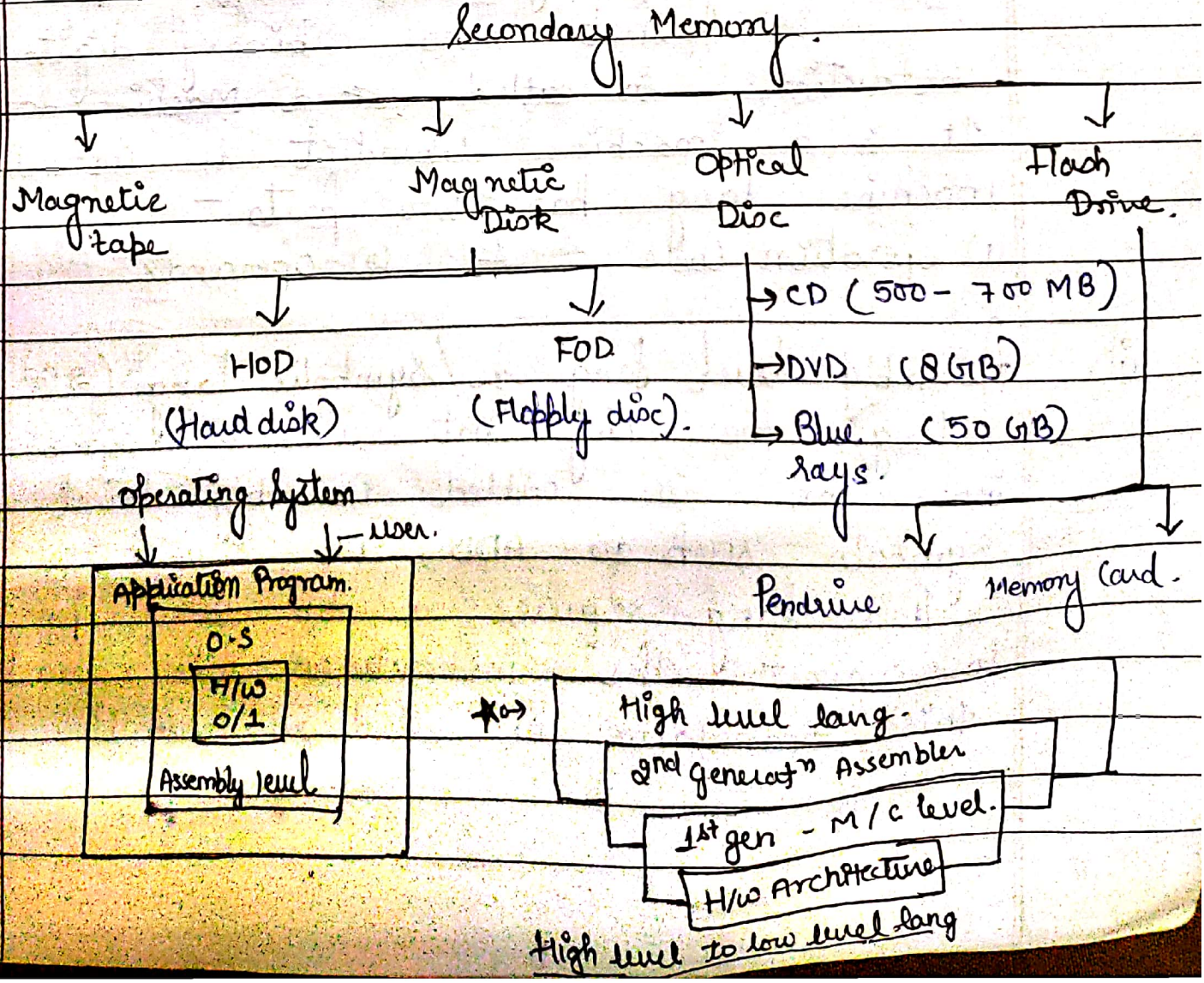
Exp:- Air bag control in cars, Anti lock break, engine control System & so on.

(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 telephones 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.

*o→



*o→

*o→

High level to low level lang

Date: 6/sept Page:

COMPUTER LANGUAGES

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

→ There are two major types of programming languages (1) low level lang & (2) 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 (1st generation lang). The lang that uses binary digits in an instructions is called a Machine language. It is a machine dependent & instruction in machine lang has two parts -
(1) operation code & (2). operands.

(ii) Assembly level language/Symbolic lang (2nd 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.

Date: _____ Page: _____

[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 :- (3rd generation)
This lang is design to facilitate the accurate description of procedure, algorithms & routines that belong to the set of instructions. for exp:- C, ~~Cobol~~ COBOL (Common Oriented business Oriented lang), C++, Java.

(ii) Problem Oriented lang. (4th generation lang) This lang is used to solve specific problem. These includes Query langs., Report generators & Application generator. for exp. SQL. (Structure Query lang), MATLAB. (Matrix Laboratory lab), LATEX.

(iii) Non-Procedural languages. (5th Generation). These langs are used in (Artificial Intelligence) AI. AI means that the machine has the ability to think & work accordingly. for exp:- LISP, ML & PROLOG.

* Differences b/w High level langs, Assembly lang. & Machine level langs.

Sr No.	(HLL) High level Lang	(ALL) Assembly level Lang	(MLL) Machine level Lang.
1.	Programs of these lang are easily understable nda	These are less understandable than high level lang however it is a more understandable than machine level lang.	Programs are less understandable.
2.	Programs are portable & machine independent.	Programs are machine dependent.	Programs are machine dependent.
3.	In this lang, debugging is easier	Debugging is more complex than HLLs	Debugging is not as a good as in HLLs & ALLs.
4.	It is the most suitable for Software Development	It is ~~x~~ not-suitable	It also not-suitable.
	Program is translated using Compiler or Interpreter	Program is translated using assembler into machine code.	No translation is required.

Date: _____ Page: _____

Date: 9/sep/19 Page: ____

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 ~~the~~ 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 n°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 Exicutable file is store 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 Interpreator

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

:- Number System :-

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 ie., 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 is called binary or two stat 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) Hexodecimal 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
3958				
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	$3 \times 10^3 = 3000$
				3958

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	$0\times2^0=0$
	1	1	2^1	$1\times2^1=2$
	1	2	2^2	$1\times2^2=4$
	0	3	2^3	$0\times2^3=0$
	1	4	2^4	$1\times2^4=16$
	1	5	2^5	$1\times2^5=32$
	0	6	2^6	$0\times2^6=0$
	1	7	2^7	$1\times2^7=128$
	1	8	2^8	$1\times2^8=256$

$(438)_{10}$

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	$1\times8^0=1$
	5	1	8^1	$5\times8^1=40$
	6	2	8^2	$6\times8^2=384$
	7	3	8^3	$7\times8^3=3584$

$(4009)_{10}$

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	$2\times16^0=2$
	C	1	16^1	$12\times16^1=192$
	A	2	16^2	$10\times16^2=2560$
	9	3	16^3	$9\times16^3=36864$

$(39618)_{10}$

Conversion:-

1. Binary to Decimal
2. Demimal to binary
3. octal to decimal
4. Decimal to octal
5. Hexadecimal to decimal
6. Decimal to Hexidecimal
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 1) Convert $(1100010110)_2$ to its equivalent decimal number system.

Sol^n:- $(1100010110)_2 = (\quad 790 \quad)_{10}$

1st Method:-

1100010110

$0 \times 2^0 = 0$
$1 \times 2^1 = 2$
$1 \times 2^2 = 4$
$0 \times 2^3 = 0$
$1 \times 2^4 = 8$
$0 \times 2^5 = 0$
$0 \times 2^6 = 0$
$0 \times 2^7 = 0$
$1 \times 2^8 = 256$
$1 \times 2^9 = 512$

$(790)_{10}$

2nd Method

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

$= 512 + 256 + 8 + 4 + 2$

$= (790)_{10}$

(2) Find the decimal number of $(11010.0110)_2$ (4)'

$2^4\ 2^3\ 2^2\ 2^1\ 2^0 \cdot 0\ 1\ 1\ 0$
$1\ 1\ 0\ 1\ 0 \cdot 2^{-1}\ 2^{-2}\ 2^{-3}\ 2^{-4}$

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

11010
- $0 \times 2^0 = 0$
- $1 \times 2^1 = 2$
- $0 \times 2^2 = 0$
- $1 \times 2^3 = 8$
- $1 \times 2^4 = 16$

$(26.375)_{10}$

0110
- $0 = 0 \times 2^{-1}$
- $0.25 = 1 \times 2^{-2}$
- $0.125 = 1 \times 2^{-3}$
- $0 = 0 \times 2^{-4}$

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

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

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

Sol' Method I:-

2	45
2	22 1
2	11 0
2	5 1
2	2 1
2	1 0
	0 1

LSB (Least significant bit or digit)

MSB (Most significant bit or digits)

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

IInd method

$2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$
$32\ 16\ 8\ 4\ 2\ 1$
$(45)_{10} = (1\ 0\ 1\ 1\ 0\ 1)_2$

(2). Convert the decimal number $(60.40)_{10}$ to binary number

1st Method:

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

$.40 \times 2 = 0.8$ — 0
$.8 \times 2 = 0.6$ — 1
$.6 \times 2 = 0.2$ — 1
$.2 \times 2 = 0.4$ — 0
$.4 \times 2 = 0.8$ — 0

$(.01100\cdots)$

$(60)_{10} = (111100)_2$

$(60.40)_{10} = (111100.01100\cdots)_2$

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

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

$2^4\ 2^3\ 2^2\ 2^1\ 2^0$
16 8 4 2 1
1 1 0 0 0

.625 × 2 = 1.25 ← 1
.25 × 2 = 0.5 ← 0
.5 × 2 = 1.0 ← 1

$(24.625)_{10} = (11000.101)_2$

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": $(176)_8$

$= 176$
$6 \times 8^0 = 6$
$7 \times 8^1 = 56$
$1 \times 8^2 = 64$
$(126)_{10}$

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

167
- $7 \times 8^0 = 7$
- $6 \times 8^1 = 48$
- $1 \times 8^2 = 64$

119

.45
- $4 \times 8^{-1} = 0.5$
- $5 \times 8^{-2} = 0.078125$

0.578125

$(167.45)_8 = (119.578125)_{10}$

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

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

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

(4) Decimal to octal :-

(1) Convert $(961)_{10}$ to its equivalent octal numbers

Soln :

8	961	
8	120	1
8	15	0
8	1	7
	0	1

$= (1701)_8$

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

8	506	
8	63	2
8	7	7
	0	7

$(772)_8$

.61
- $6 \times 8^{-1} = 0.75$
- $1 \times 8^{-2} = 0.015625$

0.765625

.61 × 8

(4) .88 × 8

(7) .04 × 8

(0) .32 × 8

(2) .56 × 8

(4) .48

$= (772.4702\ldots)_8$

(1) $(19.125)_{10} = (\quad)_8$

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

(5) Binary to octal

(1) $(10010110)_2$ convert it into octal number

Sol^n = $\underbrace{010}_{2}\overset{2}{\overbrace{010}}\underbrace{110}_{6}$ $= (226)_8$

(2) $(110101011.01101110)_2$ into $(\quad)_8 = (\quad)_{10}$

$\underset{6}{110}\ \overset{5}{\overbrace{101}}\underbrace{011}_{3} \cdot \underbrace{011}_{3}\ \overset{3}{\overbrace{011}}\underbrace{000}_{0}$

$(653.330)_8$

(3) $(101111100.1000010)_2 = (574.210)_8$

(4) $(01100101.11)_2 = (145.6)_8$

(5) $(11010111 0)_2 = (656)_8$

(6) Octal to binary:

(1) convert the octal number $(671.54)_8$ into a binary number

671.54

$\underset{110}{6}\ \underset{111}{7}\ \underset{001}{1} \cdot \underset{101}{5}\ \underset{100}{4}$

$(110111001.101100)_2 = (\quad)_{10}$

(2) $(765.727)_8 = (\quad)_2$

$\underset{111}{7}\ \underset{110}{6}\ \underset{101}{5} \cdot \underset{111}{7}\ \underset{010}{2}\ \underset{111}{7}$

$(111110101.111010111)_2$

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

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

(7) Hexadecimal to decimal:-

(1) $(A9D8)_{16} = (\quad)_{10} = (43480)_{10}$

A 9 D 8

$8 \times 16^0 = 8$

$13 \times 16^1 = 208$

$9 \times 16^2 = 2304$

$10 \times 16^3 = 40960$

$(43480)_{10}$

(2) $(9AC6.27A)_{16} = (\quad)_{10}$

9 A C 6

$6 \times 16^0 = 6$

$12 \times 16^1 = 192$

$10 \times 16^2 = 2560$

$9 \times 16^3 = 36864$

39622

·27A

$2 \times 16^{-1} =$

7×16^{-2}

10×16^{-3}

$0.125 + 0.0273437$

$+ 0.0024414$

$(39622.155)_{10}$

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

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

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

(8) Decimal to hexadecimal:-

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

16	961	
16	60	1
16	3	12 – C
	0	3

$= (3C1)_{16}$

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

16	30	
16	1	14 – E
	0	1

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

.875
× 16
14.000
1
E

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

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

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

(9) ~~Binary to hexadecimal:-~~ Hexadecimal to Binary convasion

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

$\frac{A}{1010}\ \frac{2}{0010}\ \frac{B}{1011}$ $(101000101011)_2$

(2) $(2CB.AD)_{16}$

$\frac{2}{0010}\ \frac{C}{1100}\ \frac{B.}{1011}\ \frac{A}{1010}\ \frac{D}{1101}$

$(001011001011.10101101)_2$

(3) $(A5B.17)_{16} = (1010\ 0101\ 1011.0001\ 0111)_2$

(4) $(10.11)_{16} = (0001\ 0000.0001\ 0001)_2$

(10) Binary to Hexadecimal:

(1) $(1011\ 1111.111)_2$

$\frac{1011}{B}\ \frac{1111}{F}\ .\ \frac{1110}{E}$ $= (BF.E)_{16}$

(2) $(010100.101000)_2 = 00010100.10100000 = (14.A0)_{16}$

$\frac{0001}{1}\ \frac{0100}{4}\ \frac{1010}{A}\ \frac{0000}{0}$

(1) Octal to Hexaclecimal to octal

1st method

$(6E)_{16} = (156)_8$

5

0 0 1 1 0 1 1 1 0

1 6

1st Method

2nd Method

6 E

└ $14 \times 16^0 = 14$

$6 \times 16^1 = 96$

110

2nd Method

8	110	
8	13	6
8	1	5
	0	1

$= (156)_8$

(2) ~~(157)~~ $(BFA \cdot 3)_{16} = (\quad)_8$

1st Method

B F A · 3

1011 1111 1010 · 0011 00

5 7 7 2 · 1 4

$(5772 \cdot 14)_8$

110 | 101010

2nd Method

BFA

└ $10 \times 16^0 = 10$

$15 \times 16^1 = 240$

$11 \times 16^2 = 2816$

$(3066 \cdot 188)$

·3

└ $3 \times 16^{-1} = 0.1875$

8	3066	
8	383	2
8	47	7
8	5	7
	0	5

·188

× 8

1 · 504

× 8

4 · 032

× 8

2 · 56

× 8

4 · 48

$(5772.14)_2$

(12) Octal to Hexadecimal:

(1) $(457.24)_8 = (\quad)_{16}$

4	5	7	.	2	4
000 100	101	111		010	10000
1	2	F	.	5	0

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

00 101	110
2	E

(2) $(56)_8$

5	6
00101	110
2	E

$(2E)_{16}$

2nd Method:

56
- $6 \times 8^0 = 6$
- $5 \times 8^1 = 40$

$(46)_{10}$

16	46	
16	2	E (14)
	0	2

$(2E)_{16}$

Date: _____ Page: _____

IInd Method → 6 E

↳ $14 \times 16^0 = 14$

↳ $6 \times 16^1 = 96$

$(110)_{10}$

8	1 1 0	
8	13	6 ↑
8	1	5
	0	1

$(156)_8$ Ans.

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

8 4 2 1 8 4 2 1 8 4 2 1 8 4 2 1.

B F A . 3.

1 0 1 1 1 1 1 1 1 0 1 0 . 0 0 1 1 0 0

5 7 7 2 . 1 4.

$(5772.14)_8$ Ans.

★ ARITHEMATIC OPERATIONS —

→ Binary Arithematic Operations :- Digital computer is capable of performing arithematic operation on binary no.

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

Date: _____ Page: _____

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

Ques:- Add $(1101)_2$ & $(101)_2$.

$$\begin{array}{r} 1\ 1\ 0\ 1 \quad (13) \\ +\ 1\ 0\ 1 \quad (5) \\ \hline 1\ 0\ 0\ 1\ 0 \quad (18). \text{ Ans.} \end{array}$$

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

$$\begin{array}{r} 1\ 0\ 1\ 0\ 1 \\ +\ 1\ 1\ 0\ 1\ 1 \\ \hline 1\ 1\ 0\ 0\ 0\ 0 \quad \text{Ans.} \end{array}$$

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

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

Ques:- Substract $(110)_2$ & $(10)_2$

$$\begin{array}{r} 1\ 1\ 0 \\ -\ 1\ 0 \\ \hline 1\ 0\ 0 \quad \text{Ans.} \end{array}$$

10101
-11011
0

Date: _____ Page: _____

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

```
  1 0 1 0 1
- 1 1 0 1 1
-----------
  1 1 1 0 1 0   Ans.
```

(3). Binary Multiplication :-

```
      1 1 0
    * 1 1 1
    -------
      1 1 0
    1 1 0 x
  1 1 0 x x
  -----------
  1 0 0 0 1 0   Ans.
```

(4). Binary Division

```
          1 0 1 1
      ------------
100 ) 1 0 1 1 0 1 (
      1 0 0 ↓
      -------
        0 0 1 1 0
          1 0 0 ↓
          -------
            0 1 0 1
              1 0 0
              -----
              0 0 1.
```

```
4 ) 4 5 ( 11
    4 ↓
    ---
    0 5
      4
    ---
      1.
```

Date: _____ Page: _____

```
      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 zero's to one's & one's to zero's. 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 substracting 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 $(245)_{10} = (755)_{10}$

Date: _____ Page: ____

* 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) Definetness
(4) Effectiveness
(5). Finitness

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) Definetness :- each instructions must be declared, well defined & precised there should not be any ambiguity.

Date: _____ Page: _____

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

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

* Types Or Approches 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.

Date: 18/sept Page: ____

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.

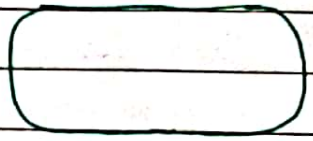
① Start
② Print or write (first no. say a)
③ Print or write second no. say b
④ c = a + b
⑤ Print c
⑥ Stop.

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

① Start
② Print or write (Enter any two no.)
③ Read or input (a, b)
④ Sum = a + b & Product = a * b.
⑤ Print "Sum" = Sum
⑥ Print "Product" = Product.
⑦ stop.

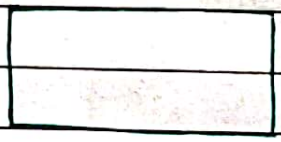
Date: _____ Page: _____

* Flow chart.



Oval shape.
(start / end).

Parallelogram.
(Input / output) symbol.



Rectangle (Process Box)

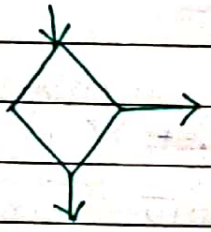
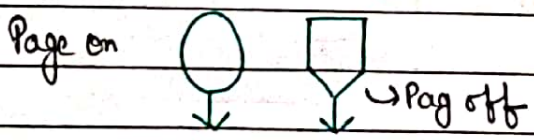


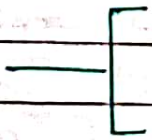
Diagram (Decision box)

Predefined Process Symbol.

Arrow symbol
Flow lines / Arrow heads.



Page off Connector.



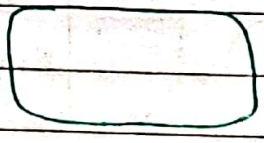
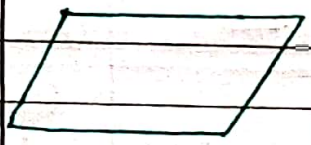
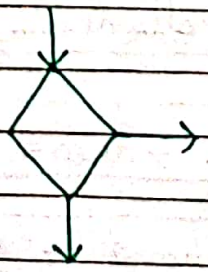
Annotation or Comment symbol.

Q→ A flow chart is a pictorial / diagram representation of the sequence of operations necessary to solve the problem.
The program flow chart show as a ~~syst~~ sequence of a instruction in a program

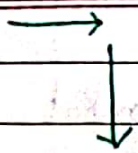
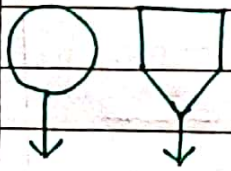
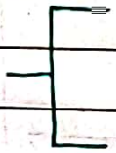
Date: _____ Page: _____

or a sub-routine.

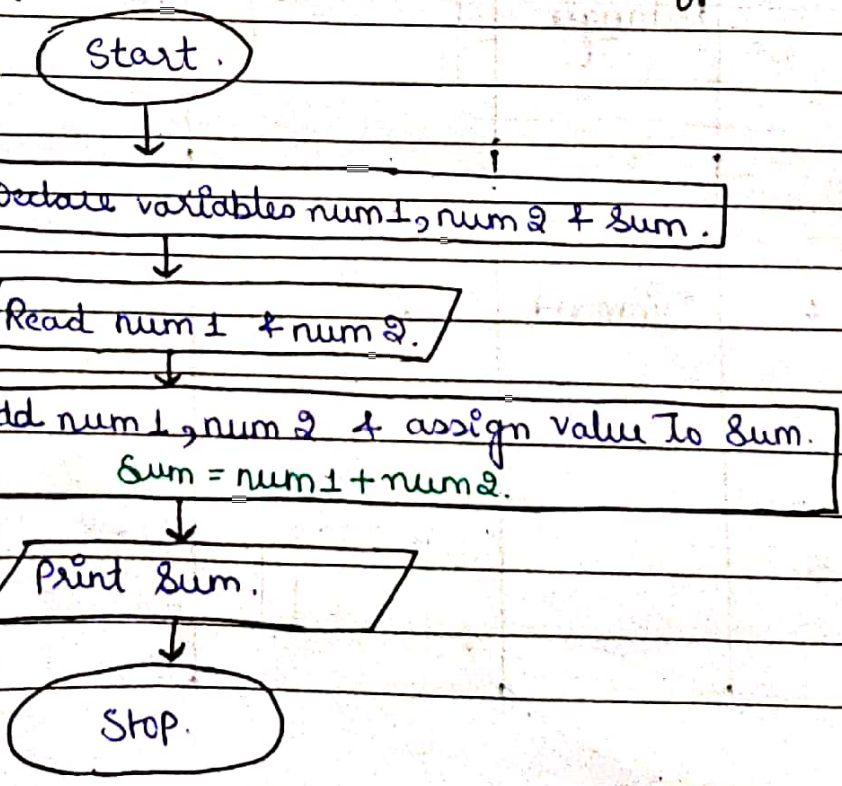
The various symbol used to draw a flow chart is given ~~above~~ below.

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.			Predefined process Symbol	It is used to represent a group of statement that together accomplishes one task.

Date: _____ Page: _____

6.		flow line arrow heads.	It is used to connect symbols & indicate the sequence of operations.
7.		Connector. Page On Connector Page off Connector	It's use to 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.



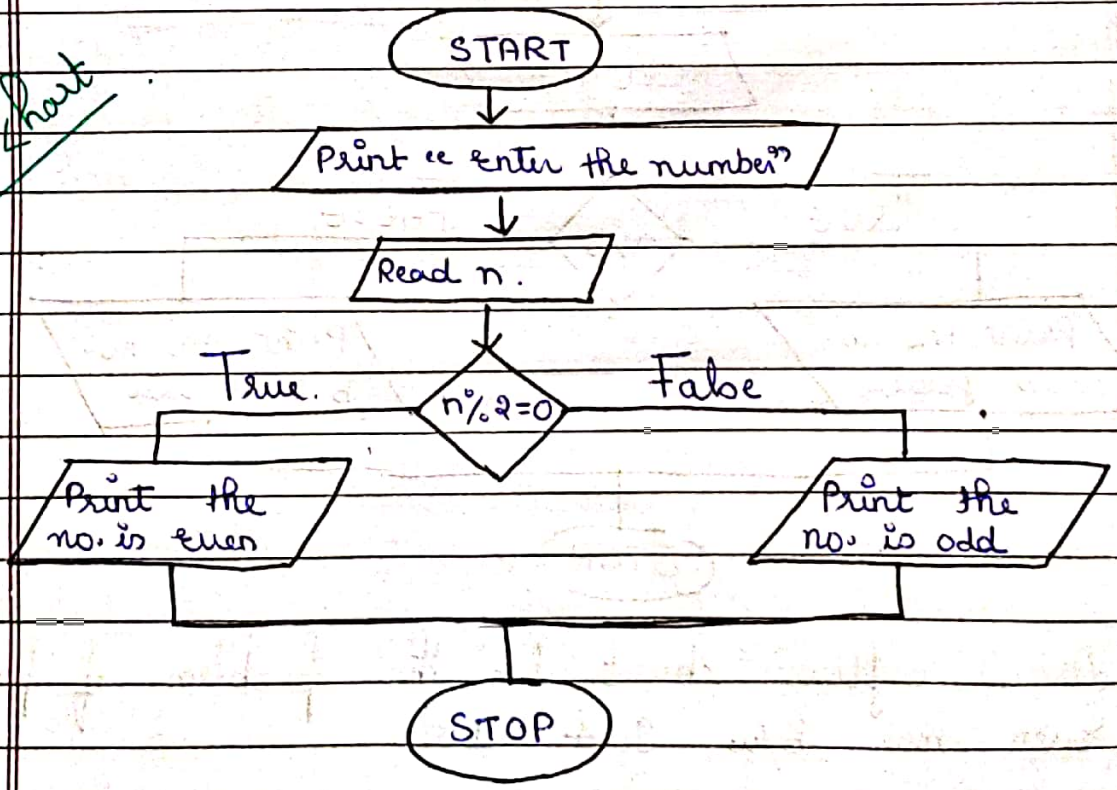
Date: _____ Page: _____

Quest- Write an algorithm to check the no. is even or odd & draw the flow chart.

Soln-

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"
[End of if structure as step 4].
⑧ stop.

Flow Chart



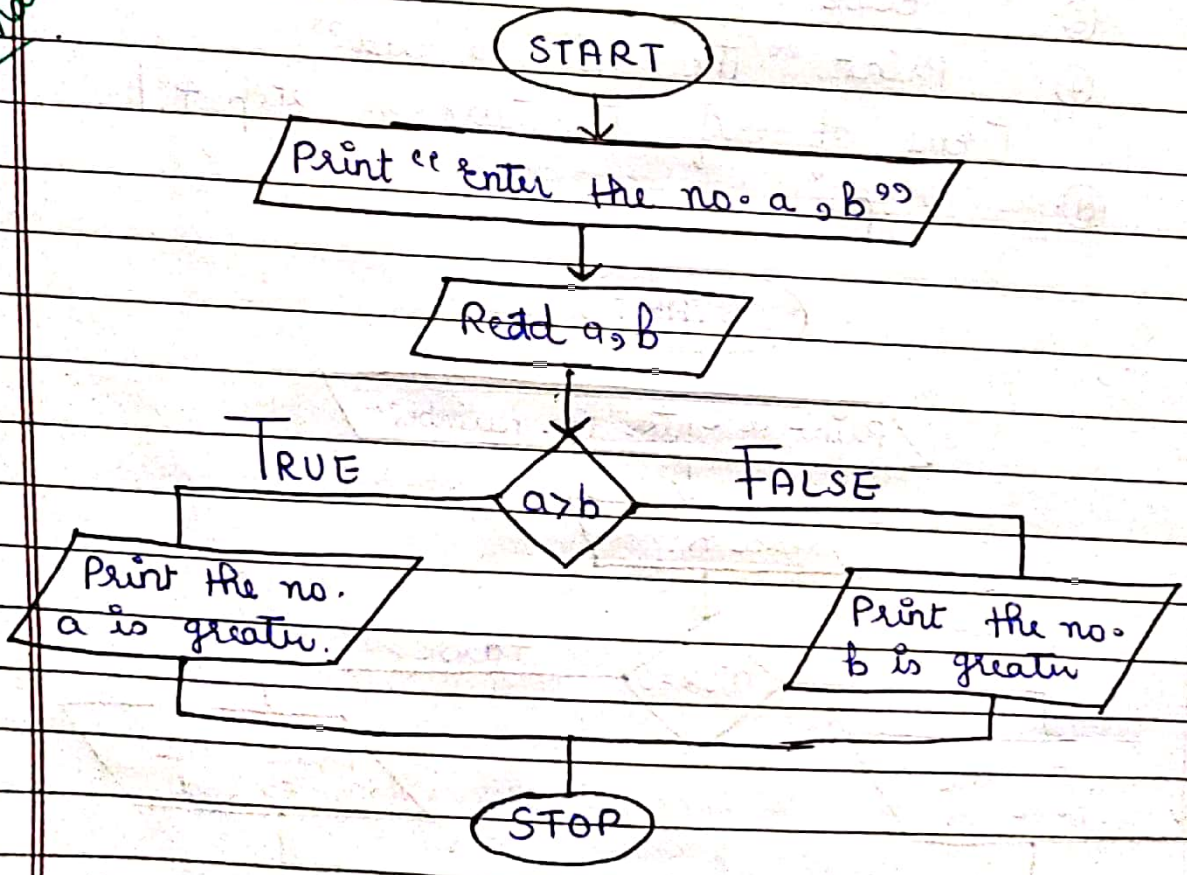
Quest- Write an algorithm to find the large no. among the two no. & Draw the flow chart.

Date: _____ Page: _____

Algorithm.

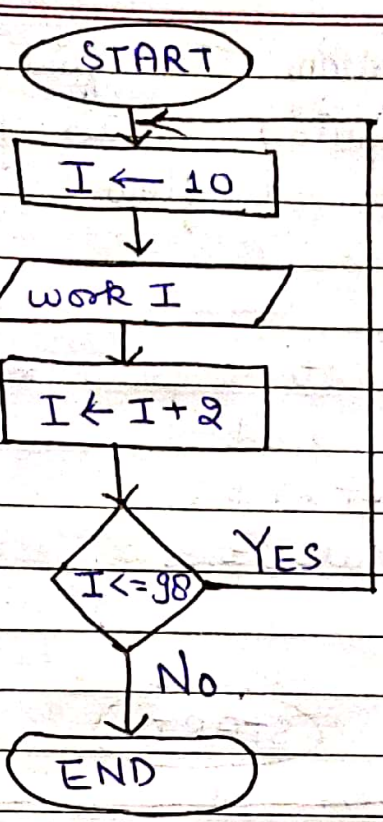
1. START
2. Print / write "enter the no. a, b"
3. Input / Read a, b.
4. If (a > b)
   Print "a is greater than b"
   else
   Print "b is greater than a"
   (end of if structure as step 4).
5. STOP.

Flow chart.



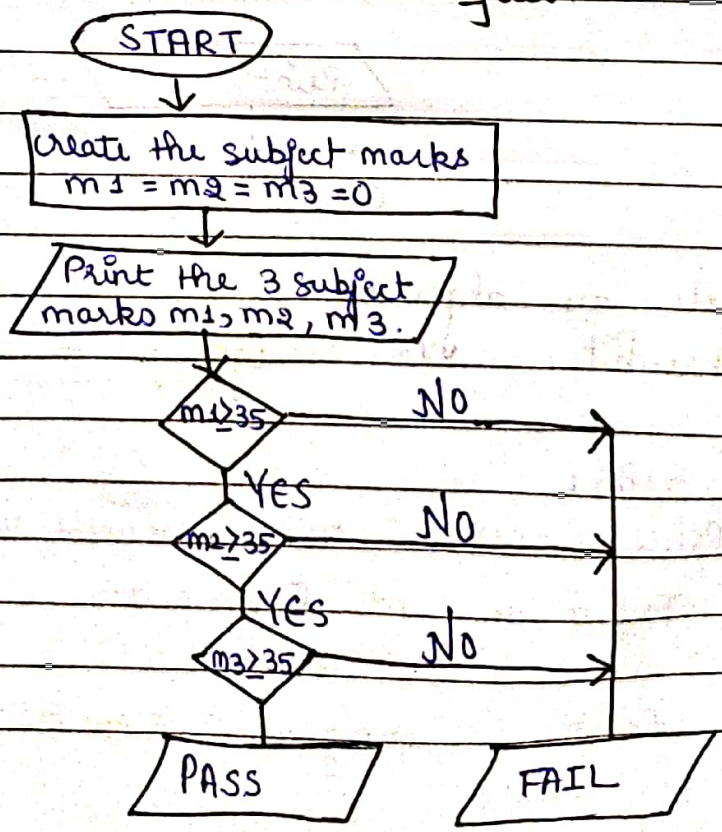
Ques 1- Draw flow chart for the problem of printing even no. b/w 9 & 100.

Date: _____ Page: _____



for (i=10; i<=98; i=i+2)

Ques1- 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 other wise fail.



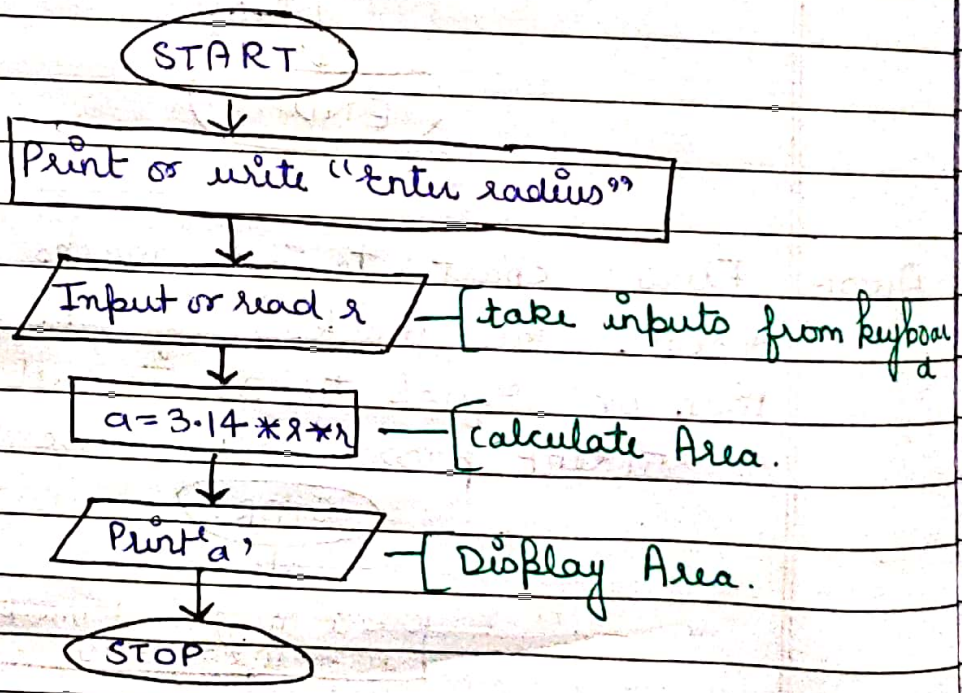
Date: _____ Page: _____

Ques- Write an algorithm To find the area of circle by inputing 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

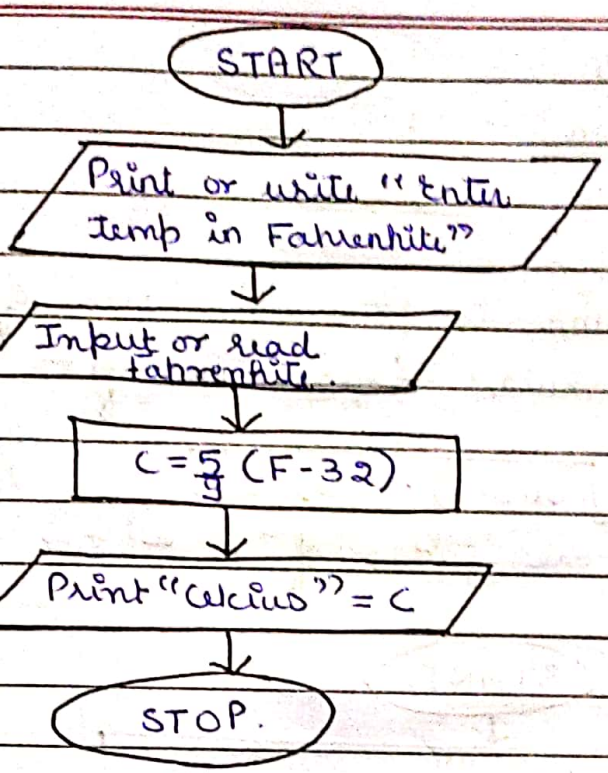


Ques- Write an algorithm to convert the temp from fahrenhite to centigrade & draw flow chart.

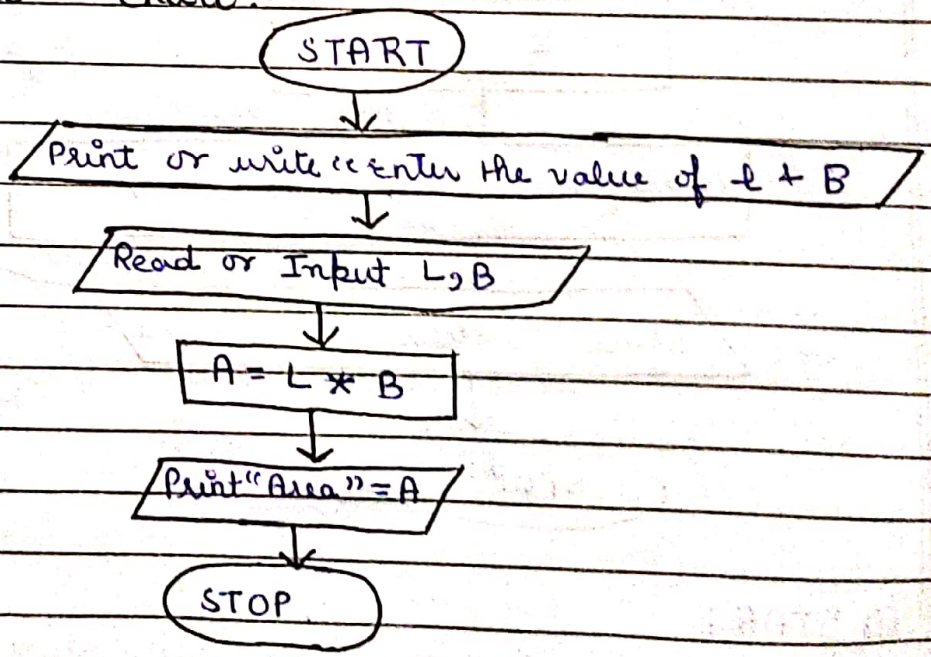
① START
② Print or write "enter Fahrenhite Value" (temp Value)
③ Input or read fahrenhite
④ $C = \frac{5}{9}(F - 32)$
⑤ Print "centigrade" = C
⑥ STOP.

Date: _____ Page: _____

Flow Chart



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



① START

② Print or write "enter the value of L & B".

③ Read or Input L, B

Date: _____ Page: _____

④ A = L * B.

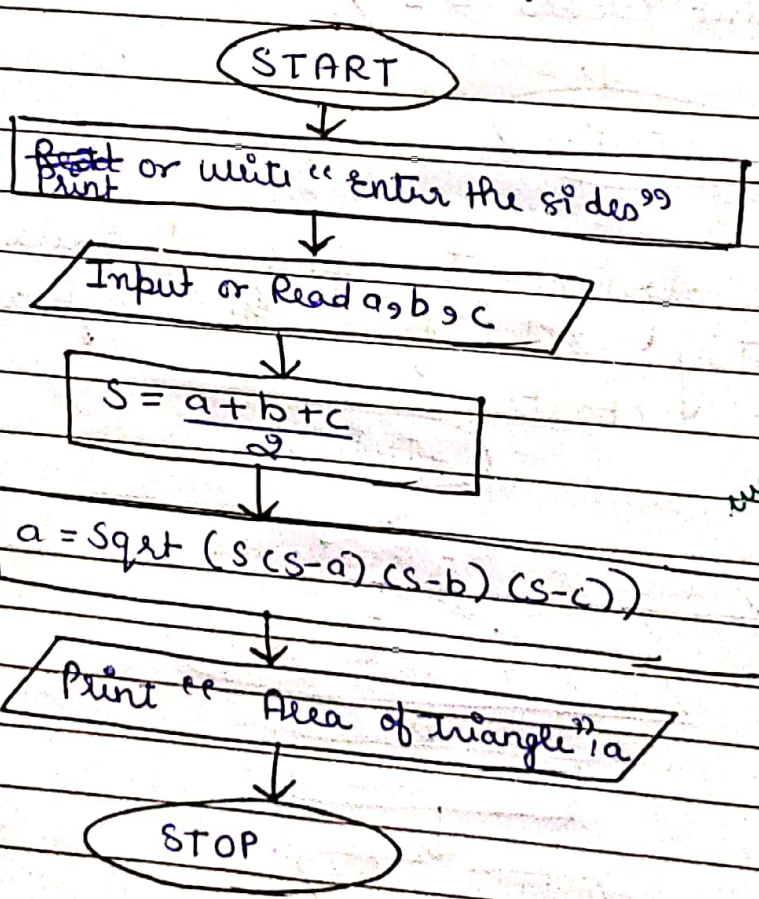
⑤ Print "Area" = A

⑥ STOP.

Ques- Write an algorithm to find the area of Δ using the given formula & draw the flow chart.

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

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



we should use predefined because multiple statement are done.

① START

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

③ Read or Input a, B, c

④ $S = \frac{a+b+c}{2}$

Date: _____ Page: _____

⑤ $A = Sqrt\ [S(S-a)(S-b)(S-c)]$.

⑥ Print "Area" = A

⑦ STOP.

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

(I) with using third variable.

(II) without using third variable & draw flow chart.

(I).

① START

② Print or write "Enter the Two nos".

③ Input or read a, b.

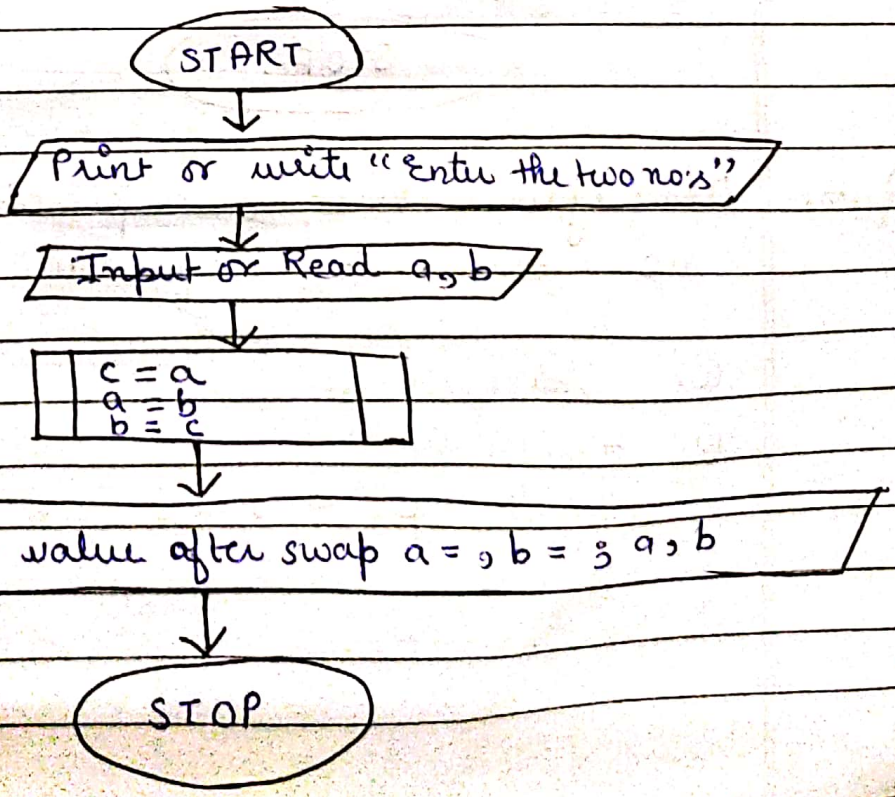
④ c = a,

⑤ a = b,

⑥ b = c,

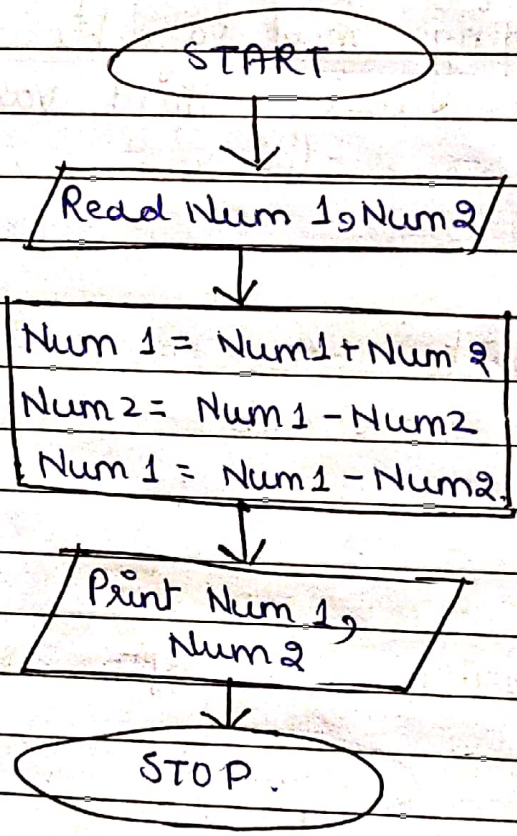
⑦ Print "Result value after swap" a = , b = ; a, b.

⑧ STOP.



Date: _____ Page: _____

(ii). ① START
② Read the value of Num 1 & Num 2.
③ Num 1 = Num 1 + Num 2
④ Num 2 = Num 1 - Num 2
⑤ Num 1 = Num 1 - Num 2.
⑥ Print the value of Num 1 & Num 2
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 till i = 5 reaches

Unit-2

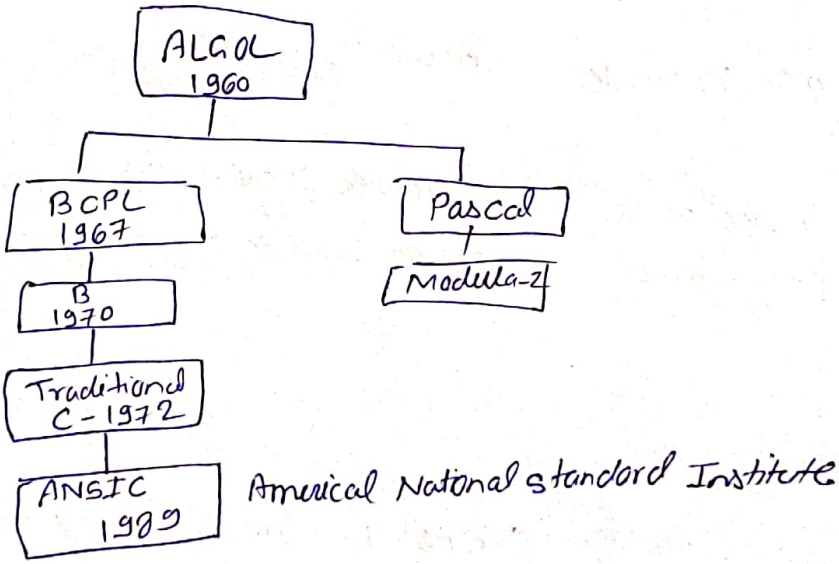
What is C:-

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 functionalism of assembly language.

Characteristics of C

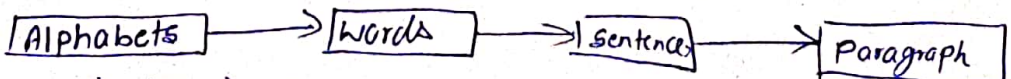
C has become a popular programming language because of its many features. The important Characteristics of C are.

* 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 procesor registers.



Fundamental data types :-

Steps in learning English languages:-



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

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

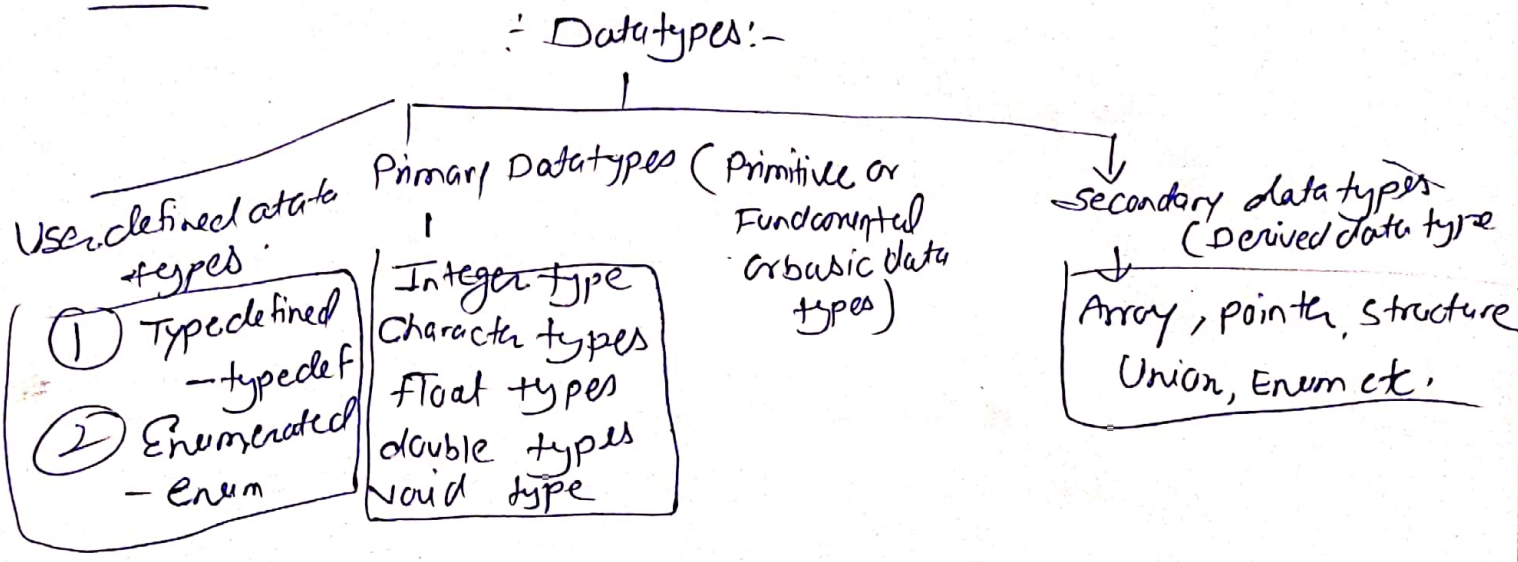
V-Bar

Backslash

Colon

Semicolon

Datatypes:-

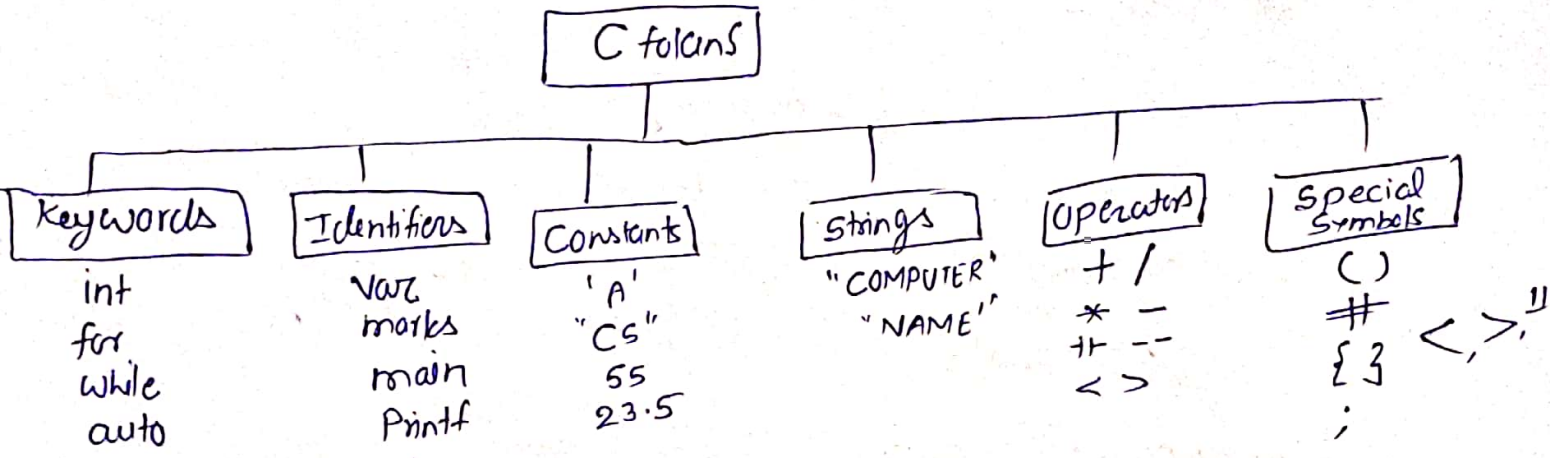


integer type: Integer represent in datatypes C → int

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 कहीं और जाएं ये सभी का use नहीं कर सकते हैं।
(2) Keywords cannot be used as variables names.
(3) All keywords are predefined and have fixed meanings.
(4) 32 keywords that are generally used in ~~traditional~~ traditional C.
(5) All keywords are in lower case. * (keywords always shown in white color as C compiler)
(6) ANSI C contains 48 keywords.

Identifiers:-

An identifiers is a name for a variable, Symbolic constants, functions, arrays, and so on.

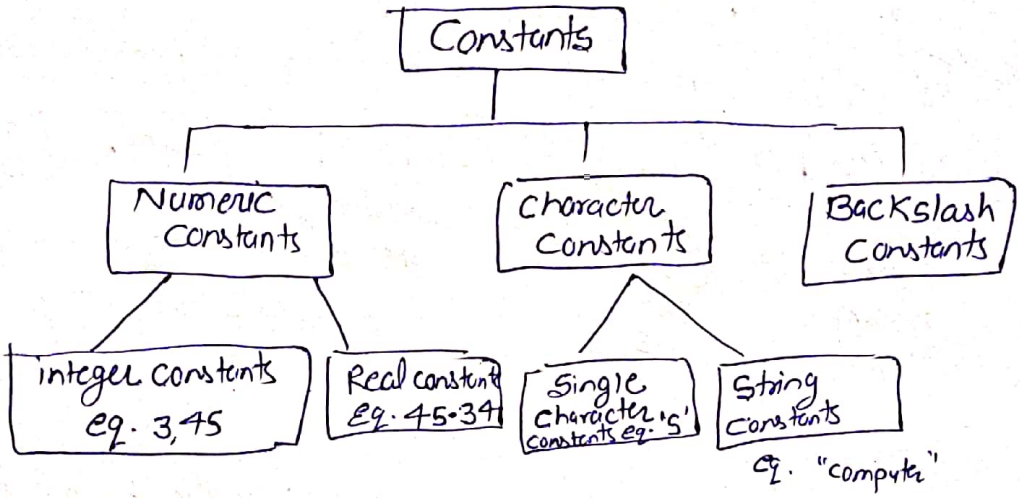
Rules for writing Identifiers:-

(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, int1, Salary_amount, A[20], i, j, k, fun, etc. valid identifiers

1Number, "y", math-no, matrix first, 1fun, 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:-

(i) Decimal constants: These are the collection of digits 0, 1, 2, ...9. The format specifier is %d. Examples are 9870, 356

(ii) 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.

(iii) Hexadecimal Constants: These constants begin with 0X and 0x 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, 0X34AB, 0x61fc.

(2)

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.

(11) Exponent Form: This constant consist 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) Mantiss 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
542 (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:-

(i) Printf ("Hello word");
- Printf: // identifier
- ("Hello word"): // String
- ; : Angular bracket // C symbol

(ii) int x = 5;
- int: keywords
- x: identifiers
- =: sumobles
- 5: Constants (Numberic)
- ;: sumobles

Date: _____ Page: _____

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 type.

→ 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.

Date: _____ Page: _____

→ 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 perform on a specific type of data.

*o→ 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 1 byte = 8 bit 2 byte = 16 bit	-2^n to $2^n - 1$ -32,768 to 32,767 (signed). 0 to 65536 (unsigned).	

Date: _____ Page: _____

b. Characters 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	char	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 no.s) 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	float	4 byte 4 byte = 32 bit	3.4e-38 to 4.e+38

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.

A Double data type no. used 64 bits given a precision of 14 digits.

Date: Page:

The no. that double Type represents the same data Type that float represent but with a greater precision. To extent 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 17e + 308

(e). Void data types :- The void type has no values 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 +ve & -ve values can be stored

Date: _____ Page: _____

→ 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.

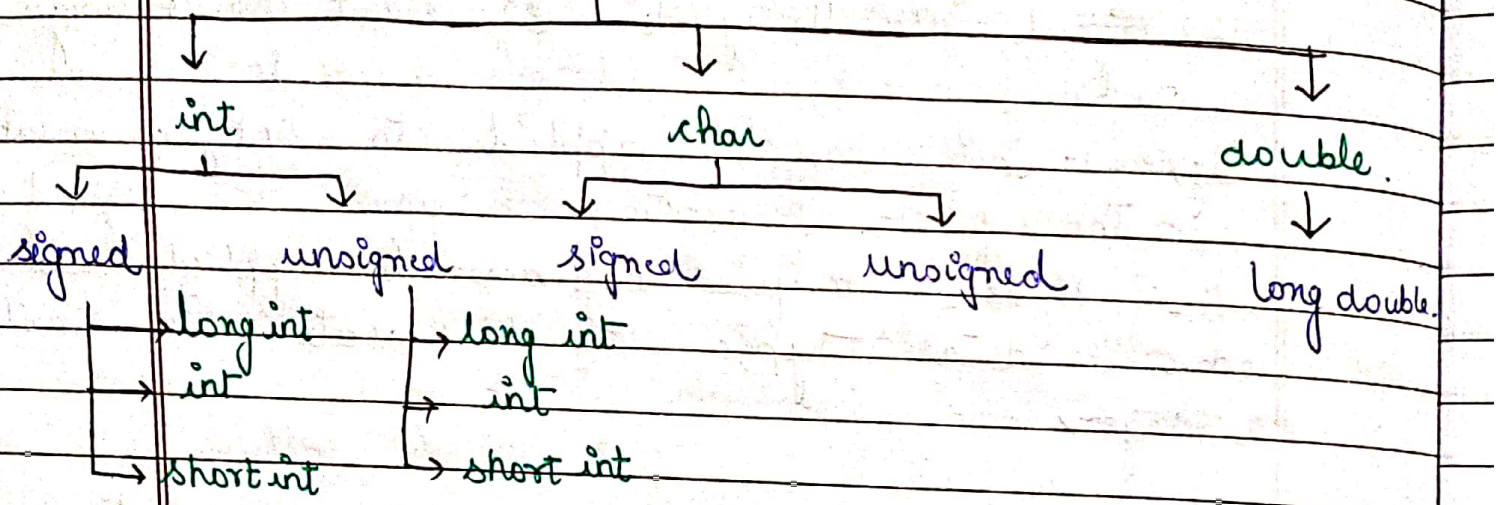
Unsigned :- This can be used for the both integer & characters. It is used to create an unsigned integer & it can also be used in combination with long & short.

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.

Date: _____ Page: _____

Modifiers



SNo.	Data Type	Keyword	format Specifi	Size in bit	Size in byte	Range
(1)	Integer	int	%d	16	2	-32768 to +32769
		signed int	%d	16	2	-32768 to +32769
		unsigned int	%u	16	2	0 to 65535
		short int	%d	16	2	-32768 to +32767
		Signed short int	%d	16	2	〃
		Unsigned short int	%u	16	2	0 To 65535
		long int	%ld	32	4	-2147,483648 to +2147,483647

Date: _____ Page: _____

		Signed long int	%ld	32	4	-2147,483648 to +2147,483647
		unsigned long int	%lu	32	4	0 To 4294967 295.
②	Character	Char	%c	8	1	-128 To 127.
		Signed char	%c	8	1	-128 to 127
		unsigned char	%c	8	1	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

★ imp 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 resistors is called the storage location or memory location.

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

(4) Life Time :- The declaration of execution of a program during which the variable retains its value into memory is called it's 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

(2) 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.

Date: _____ Page: _____

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

*(3) Static Storage Class of x

(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 it's 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

Date: _____ Page: _____

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

Result → i = 1 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 → x = 0
x = 6
x = 0

Date: _____ Page: _____

* Q→ Differences b/w different Storage classes

S.No.	Storage class	Where Stored (location)	Initial Or default Value	Scope Visiblity	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.	~~fal~~ 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	Intire program execution.

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

Date: _____ Page: _____

(1) Unary operators :- The operator that operate over a single operands is called unary operators for exp:- logical Not (!), bitwise Complement (~) & Unary Plus/Minus (+/-)

(2) Binary operators :- The operator that operates upon Two operands is called Binary operators. exp:- logical AND (&&), logical OR (||) & Relational operators.

(3). Ternary operators :- The operator that operates over three operands is called as ternary operators for exp:- Conditional operator (?:)

* Types of operators :- Based on classification, operators are of eight Type.
Arithmetic operators.
Relational "
Logical "
Assignment "
increment & decriment 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.

Date: _____ Page: _____

→ 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 = 8 + 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 exp :- we may compare the age of two person or the price of two items & so on.

"The result of relational operation is either 1 (true) or 0 (false)"

→ C support six relational operator in all that are given below -

Date: _____ Page: _____

	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 additional to the relational operators, C has the following 3 logical operators :-

(&&) Logical AND.
(||) Logical OR
(!) Logical NOT.

for eg:- 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.OP = expression

Date: _____ Page: _____

or, **Exp 1 OP = Exp 2**

→ where V = variable, OP = operator & Exp = expression

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

Exp:- (1). (x+[redacted]) = (x+1)

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

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

(a). x+ = 3.

∴ **x = x+3**

(b) a * = a+1.

∴ **a = a* (a+1)**

(c). a/ = 3

∴ **a = a/3**

(d). a%=b

∴ **a = a%b**

Ques:-

```
# define N 100
# define A 2
main ()
{
    int a;
    a = A;
    while (a < N)
```

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

Result → 2, 4, 16 } output contains only 3 values i.e 2, 4, 16

(iv). Increment & Decrement operator - The operator that add one & substract one to the operant are called increment (++) & decrement (--) operators respectively.
These operators are of two types -

a. Preincrement or predecrement (prefix) :- These operators add or substract one of the operant & then assign the result to the variable respectively.
eg int x = 5, y

```
y = ++x ;  | x = 6 , y = 6
y = --x ;  | x = 4 , y = 4.
```

b. Post increment or post decrement (Postfix) :- These operators add or substract one to the operant & then assign the result to the variable respectively. eg:- int x = 5, y.

```
y = x++ ;  | y = 5 , x = 6
y = x-- ;  | y = 5 , x = 4.
```

Date: _____ Page: _____

tq(i).

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

exp of (Increment).

Result 4, 5.

tq(ii).

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

{exp of decrement}

Result 3, 1.

(6) Conditional operator:- A ternary operator pair (?:) is available in C to construct conditional expression of the term.

Exp 1 ? Exp 2 : Exp 3.

Date: _____ Page: _____

where Exp 1, 2, 3 are expressions.

→ Exp 1 is evaluated 1st. 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)

Eg 1- 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 followed.

→ if (a > b)
x = a;
else
x = b;

(7). Bitwise operators :- C has supportive special operators i/e 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.

Date: _____ Page: _____

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.

X	Y	Result X & Y	X | Y	X ^ Y
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

(8). Special operators :- C supports some special operators of ~~es~~ 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 right most

Date: _____ Page: _____

expression is the value of combined expression
Exp:- 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 ~~qualifier~~ 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= size of (int); or x= size of (char);
printf("%d", x);
getch();
}
```

Result → x= 2 ~~or~~ or 1

Date: _____ Page: _____

* → 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 %d %d", x, y, z);
  getch();
}
```

Result → x = 2, y = 8, z = 2.

* Operator precedence & associativity :- Determines which operator is evaluated first when an expression has more than one operator.

Date: _____ Page: _____

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

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

(1) High priority (*, /, & %)

(2) Low priority (+, -)

→ for exp:- a = 100 - 2 * 30

② ① Precidence.

a = 100 - (2 * 30)

a = 100 - 60

a = 40 Ans.

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

① ②.

x = 9 - 4 + (3 * 2) - 1 (/ & * has precendence same then use left to right operator)

x = 9 - 4 + 6 - 1

①

x = 5 + 6 - 1

①

x = 11 - 1

x = 10 Ans.

Ques :- a = 100 + 200/10 - 3 * 10.

① ②

a = 100 + 20 - (3 * 10) (/ & * has same precedence then we use left to right operator).

a = 100 + 20 - 30

① ②

a = 120 - 30

a = 90 Ans.

Date: _____ Page: _____

* Associativity in C lang :- Associativity is used when there are two or more operators of same precedence is present in an expression. for exp :- 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) → first expression is evaluated.

[0 != 3]

(1).

i.e (1 == 2) executes 1st precision of both statement is same so we use LTR. 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 → (1 > 2 + 3 && 4). (precidency of + is greater among all the sign.)

(1 > (2+3) && 4).

(1 > 5 && 4). (> and && has same precidence, so we use LTR)

(0 && 4)

(0) → 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
}
```

Date: _____ Page: _____

Result → c = 1, a = 1, b = 0, d = 0. c = a || --b
c = 1 || 0
c = 1

Ques:- int y = 10.
++y + y++ + y-- + y++;
{(++y) + (y++) + (y--) + (y++)};
{ 11 + 12 + 12 + 12 } = 46 Ans.

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 lang 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

Date: Page:

any expression. it is called implicit type conversion.

exp1- (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

bcoz, this is an integer type variable so, only integer value of actual (after point value) is store truncate (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)

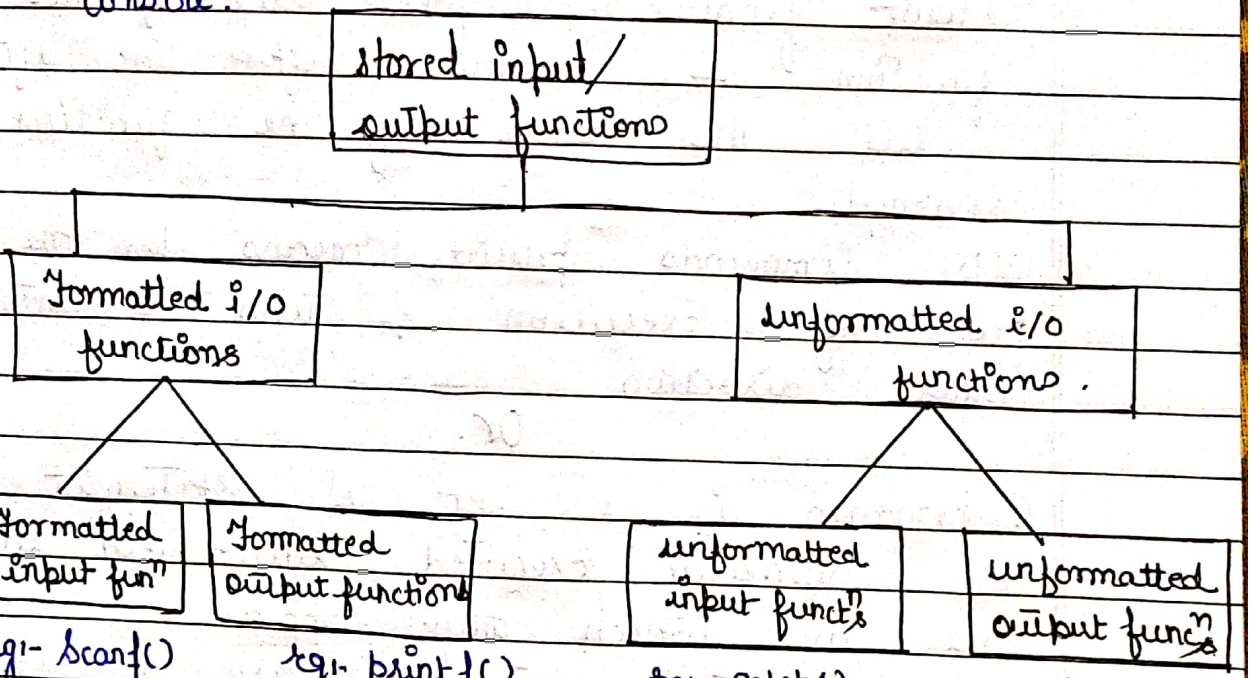
exp1-
```
main ()
{ int a;
  int b;
  float c;
  c = float (a/b);
  print ("c%f", c);
}
```

Date: _____ Page: _____

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.



* Basic Structure of C programming.

① Documentation Section
//* comments section
objective.

② Link Section. (Pre-processor Section).
include<stdio.h
include <conio.h>
<maths.h>
<Graphics.h>.

③ Definition Section.
Void fun()

④ Main()
① Decration part
② execution part

⑤ sub function
function 1
function 2
n function

Tuesday

Date: 22/Oct/19 Page:

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.

Ctrl 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 sequencially in the order in which they appear.

→ In C ctrl statements are devided into four categories.

(a) Sequencial statement.

(b) Conditional statements (if, if else, nested if else & switch)

(c) Iterative or looping statements (for, while, do while)

(d) Unconditional or jumping statements (break, continue, return, exit, goto & so on)

Date: _____ Page: _____

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

The following are the conditional statement-

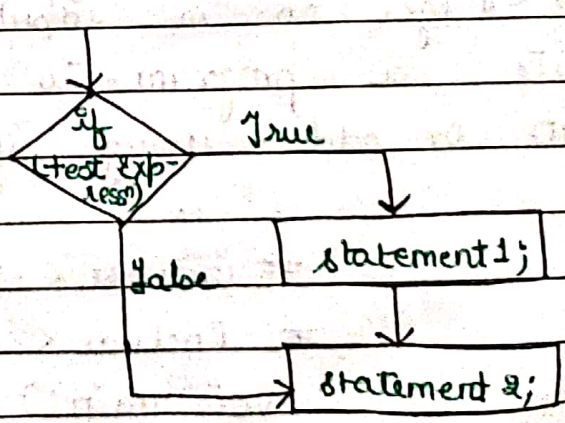
(a). if statement :- The if statement is a conditional statement or decision making statement if is a two way branching statement & uses a logical statement that result in either true or false. If the result is true, the body of the if statement is executed otherwise, the ctrl will be transfered to outside 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 write the program to find out the greater no. using if statement.

Date: _____ Page: _____

Sol^n :-

```
#include <stdio.h>
#include <conio.h>
void main()
{
    clrscr();
    int x, y;
printf ("enter the two no.s 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^n :-

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

Date: _____ Page: _____

```
scanf("%d", &x);
    if (x % 2 == 0)
      {
        printf("x is even no")
      }
       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 controlled is transfered to the next statement.

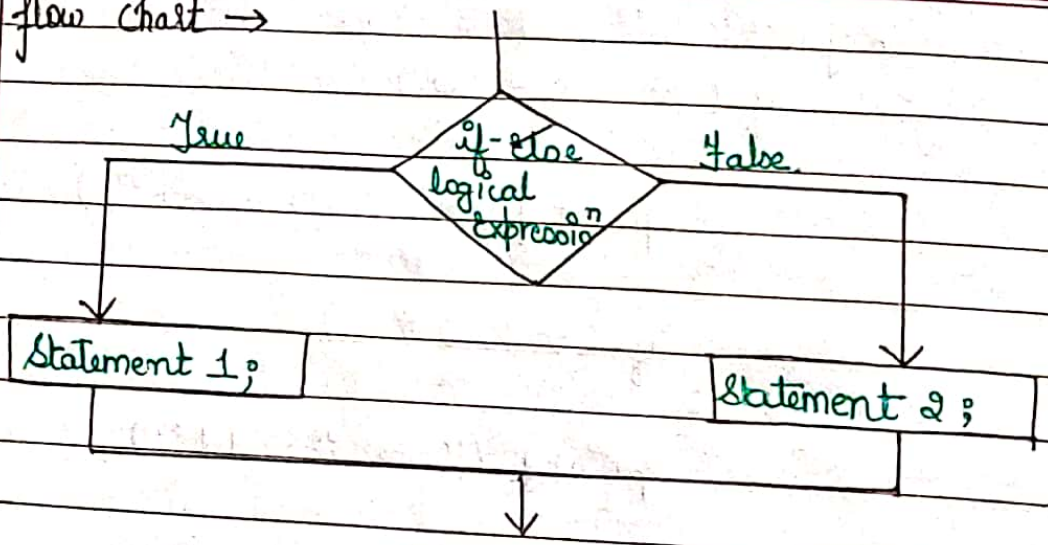
→ The syntax & flow chart are as follows-

Syntax ⟶

```
if (logical expression)
  {
    statement 1;
  }
   else
     {
       statement 2;
     }
```

Date: _____ Page: _____

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 function 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();
   }.
```

Date: _____ Page: _____

(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 statement are used when multiple condition are specify To perform a given task. So, an if else statement may have multiple if-else inside it's body.

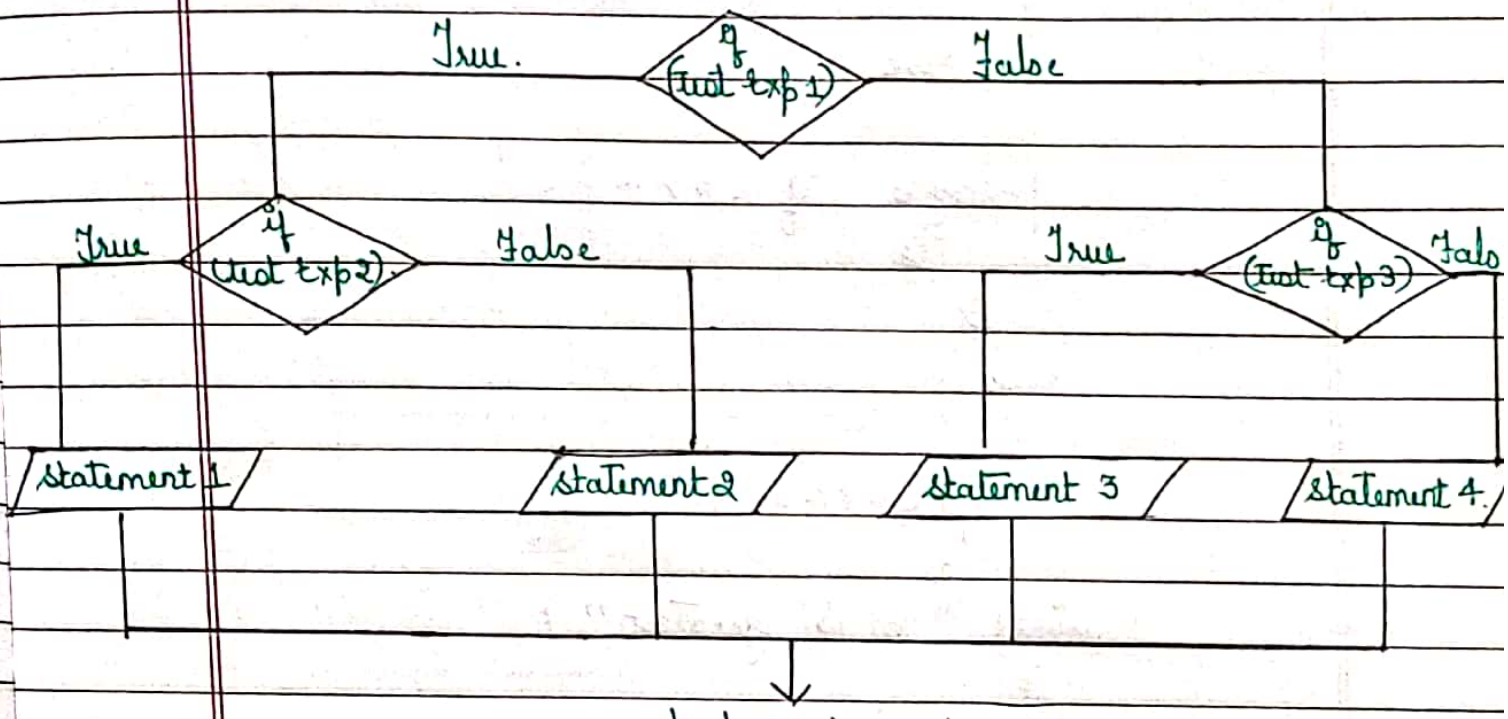
→ The syntax & flow chart are as follows -

Syntax →

```
if (test - exp 1)
  {
    if (test - exp 2)
    {
      statement 1;
    }
    else
      { statement 2;
      }
  } else
      { if ( test exp 3)
        {
          statement 3;
        }
        Else
          {
            statement 4;
          }
  }
```

Date: _____ Page: _____

flow chart →



Ques- WAP to find the largest no. among three no.s 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 no.s a,b & c");
scanf("%d %d %d",&a,&b,&c);
if(a>b)
{
    if(a>c)
    {
```

Date: ______ Page: ______

```
printf(" a is greatest")
}
} else
{
if (b>a)
else
{
if (b>c)
{
printf(" b is greatest")
}
else
{
printf(" c is greatest")
}
}
```

(d) The else if ladder. Their is another way of putting ifs statement together when multi path decisions are involved.

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

Date: _____ Page: _____

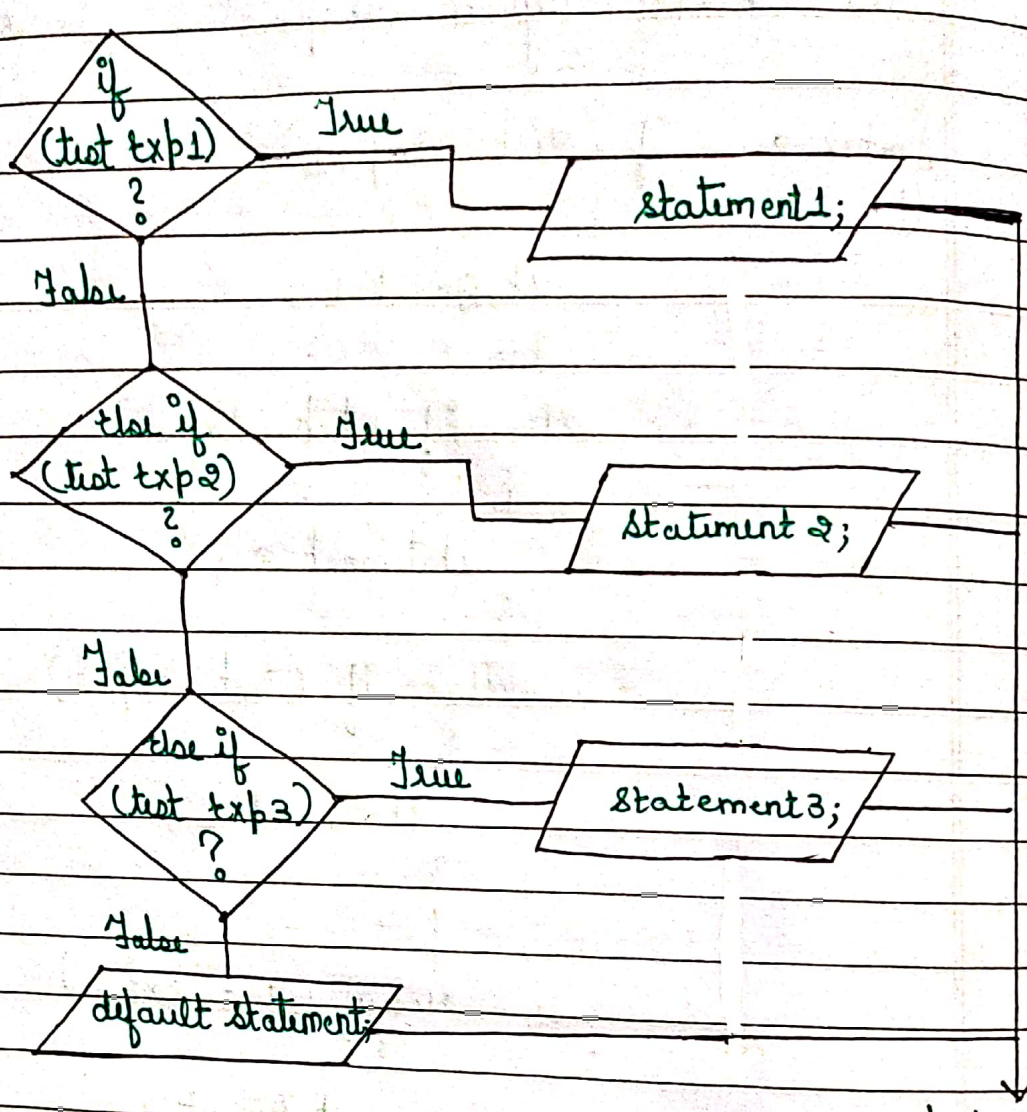
then the body of the first is executed otherwise the next logical expression is evaluated & so on.

Syntax →

```
if (test exp 1):
{
    statement 1;
}
else if (test exp 2)
{
    statement 2;
}
else if (test exp 3)
{
    statement 3;
}
  :
else if (test exp n)
{
    statement n;
}
else
{
    default statement;
}
```

Date: _____ Page: _____

flow chart →



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

Average Marks	Grade.
80 - 100	Honours
60 - 79.	Ist Devision
50 - 59	IInd Division
40 - 49	IIIrd Division
0 - 39	Fail.

Date: _____ Page: _____

```
#include <stdio.h>
#include <conio.h>
void main()
{
clrscr();
int marks, grade
if (marks > 80)
{
printf ("honours")
}
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/oct 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 compose 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.

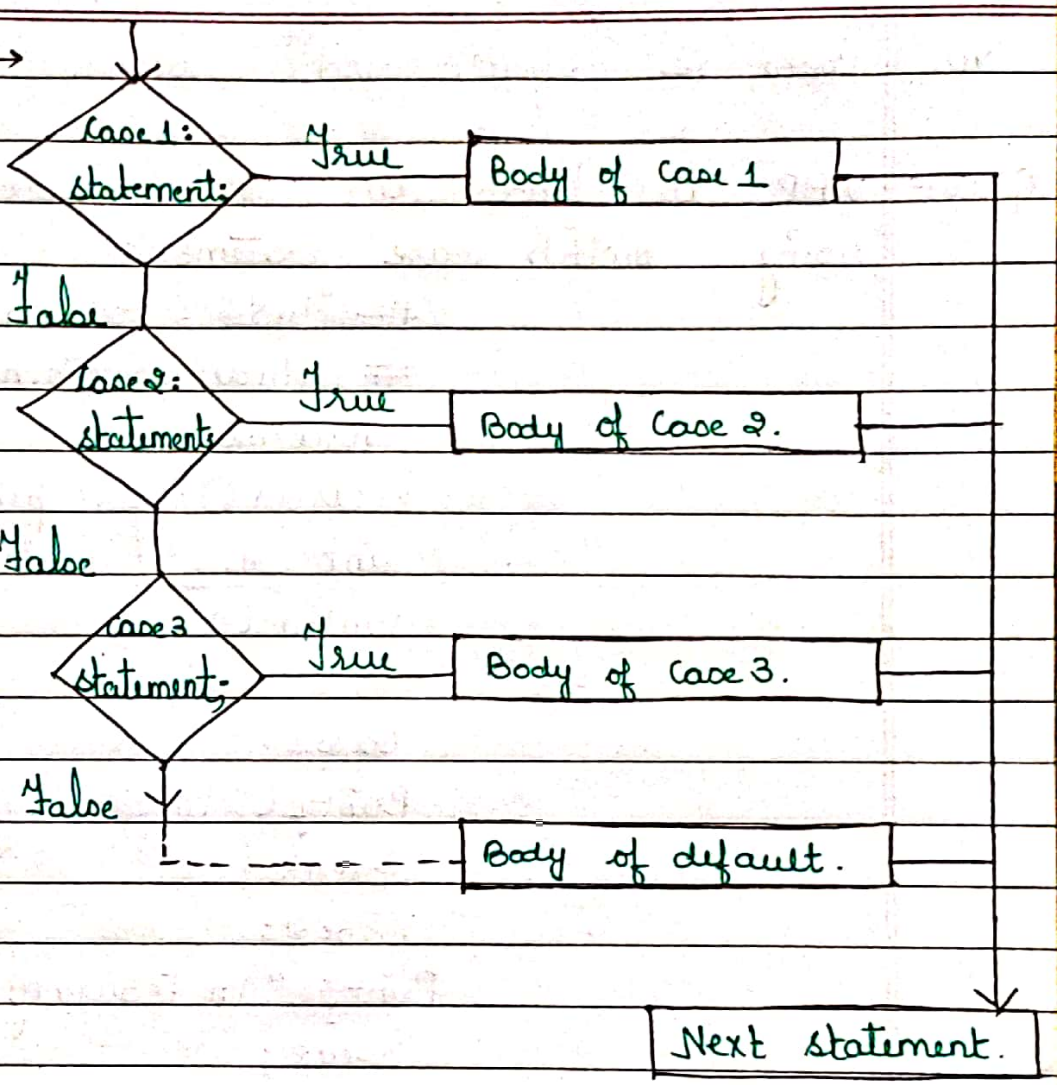
→ The gensal syntax & flow chart of the switch statement is -

Syntax →

```
Switch (expression)
{
Case 1 :
statement ;
break ;
Case 2 :
statement ;
break ;
Case 3 :
statement ;
break ;
.
.
.
default n :
statement n ;
}
```

Date: _____ Page: _____

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 characters.

(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

Date: _____ Page: _____

(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("\n January = 31 days");
break;
Case 2:
Printf("\n February = 28 or 29 days");
break;
Case 3:
Printf("\n March = 31 days");
break;
Case 4:
Printf("\n April = 30 days")
!
!
default:
Printf("\n No choice")
break;
}
getch();
}
```

Date: _____ Page: _____

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", &a)
switch(x)
{
case 1:
printf("\n Monday");
break;
case 2:
printf("\n Tuesday");
break;
case 3:
printf("\n Wednesday");
break;
case4:
printf("\n Thursday");
break;
case 5:
printf("\n Friday");
break;
!
default("\n No choice")
break;
} getch();
}
```

Friday

Date: 25/oct Page:

Ques- WAP to find the value 'y' for a particular value of 'k' if → k=1 then $y = ax+b$

if k=2 then $y = ax^2 + b^2$.

if k=3 then $y = ax$.

if 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");
```

Date: _____ Page: _____

```
break;
}
getch();
}
```

Ques:- WAP to calculate the addition, Substraction, 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("\n value is not valid")
    break ;
    }
    getch();
    }.
```

[2] Repeative statement or Iterative Statements or loops Statement statement that may repeat other statement more than one until a logical statement evaluates to True is called a loop, Repeatative 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 loops 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

Date: _____ Page: _____

→ 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 controlled is transfer out of the loop & exist the program continues with the statement immediately after the body of the loop.

```
void main()
{
  clrscr();
  i=1; // initialization or Counter Variable
  while (i<=5) // condition
  {
    printf("\n Allmin");
    i++; // increment
  }
  getch();
}
```

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

Date: Page:

```
#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();
}
```

i	i<=n	sum = sum+i	i++
1	1<=9 (T)	sum = 0+1 = 1	2
2	2<=9 (T)	sum = 1+2 = 3	3
3	3<=9 (T)	sum = 3+3 = 6	4
4	4<=9 (T)	sum = 6+4 = 10	5
5	5<=9 (T)	sum = 10+5 = 15	6
6	6<=9 (T)	sum = 15+6 = 21	7
7	7<=9 (T)	sum = 21+7 = 28	8
8	8<=9 (T)	sum = 28+8 = 36	9
9	9<=9 (T)	sum = 36+9 = 45	10
10	10<=9 (F)	~~sum =~~	

Output → enter the range = 9
sum = 45.

Date: _____ Page: _____

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

Imp:-

```
#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

Date: 31/oct Page:

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

Result -

m = n * i = 10 x 1 = 10
m = n * i = 10 x 2 = 20
m = n * i = 10 x 3 = 30
m = n * i = 10 x 4 = 40
m = n * i = 10 x 5 = 50
m = n * i = 10 x 6 = 60
m = n * i = 10 x 7 = 70
m = n * i = 10 x 8 = 80
m = n * i = 10 x 9 = 90
m = n * i = 10 x 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

Date: _____ Page: _____

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.

exp →
```
main()
{
clrscr();
i = 1;
do
{
printf("\n Allmin");
i++;
}
while(i<=5);
getch();
}
```

Ques 1- WAP to print sum of n natural no.s 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 using do while loop.

inp.

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

Date: _____ Page: _____

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

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

```
for( initalization ; test condit" ; increment/decrement)
  {
      body of the loop
  }.
```

Exp :-

```
for (i=1; i<=5 ; i++)
  {
    printf("\n Allmin");
  }
    getch();
}
```

Friday

Date: 1/Nov Page:

Ques:- WAP to find out the sum of first n integer no.s using for statement.

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

Ques:- WAP to evaluate the factorial of an integer no. 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;
     i++;
   } while (i <= 10)
   Print f("\n %d * %d = %d", n, i, m);
   getch(); }.
```

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

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

Date: _____ Page: _____

Ques5: WAP to calculate fabonacci series 0, 1, 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

= a+1 = 1

Ques6: WAP to calculate sum of series $2^0, 2^1, 2^2, 2^3 \ldots, 2^n$

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

Date: _____ Page: _____

```
        sum = sum + pow (2, i)
    }
    getch();
}
```

Ques7:- WAP to print sum of series $1^4+2^4+3^4+4^4---+100^4$

```
#include <Maths.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();
}
```

Ques8:- WAP To print sum of series $1+\frac{1}{2}+\frac{1}{3}+\frac{1}{4}+---+\frac{1}{n}$

```
#include <stdio.h>
#include <conio.h>
#include <Maths.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", \n 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( exp 1; condition 1; exp 2)
  {
    for (exp 3; condition 2; exp 4)
      {
        statement 1;
        statement 2;
      }
  }
```

Ques 1:- WAP to find the sum of digits

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

Date: _____ Page: _____

```
clrscr();
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
= sum");
getch();
}.
```

Ques2- WAP to print the following

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

```
# include <Maths.h>
# include <conio.h>
# include <stdio.h>
void main()
{
clrscr();
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++)
```

Date: _____ Page: ____

```
{
    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();
            }.
```

Date: _____ Page: _____

Ques. WAP 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

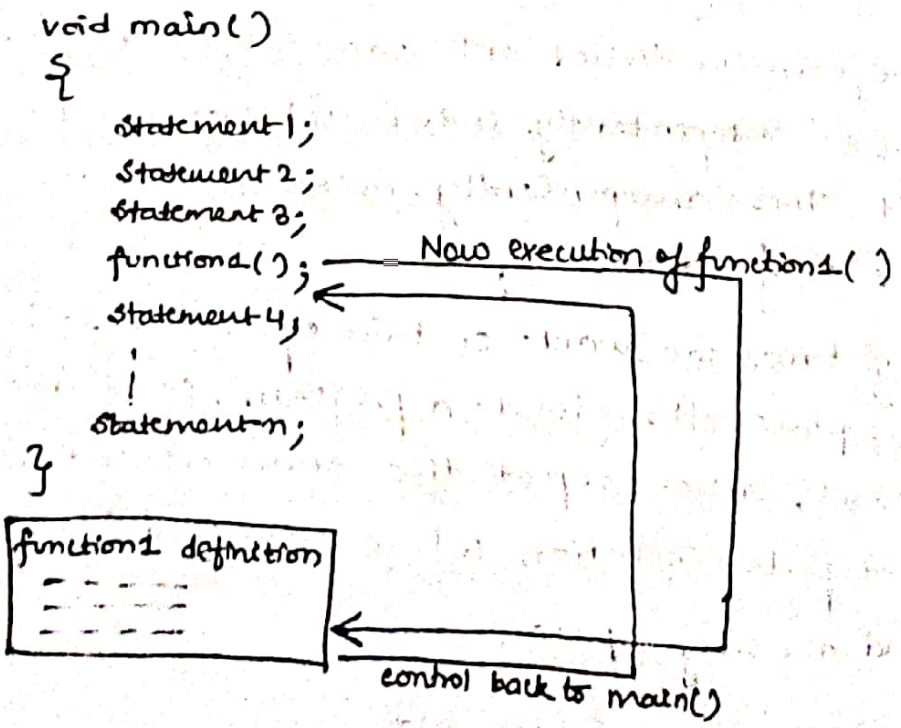
①

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 →



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 it's 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.

∴ 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.

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 seperately 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");
}
```

output:

```
Welcome to world of C Programming
@@@@@@@@
Programming is LOGIC
```

Note: main( ) ...

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 seperated by commas and use of parameters names is optional.
* return type void means function doesn't return any value
* Empty parantheses 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);

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;

}

NOTE: ✓ The parameter list is a comma seperated 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 braces acts as return.

✓ syntax of return :→

return; or return (variable);

[returning value of variable]

eg.

(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.

Eg: 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.

```
int factorial(int);
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;
  }
```

WAP

Q. WAP to find sum of series $2^0+2^1+2^2+\cdots+2^n$ 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 x)
{
    int i, s=0;
    for(i=0; i<=x; i++)
    {
      s = s + pow(2, i);
    }
    return s;
}
```

Q. 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.

In function call → Actual parameter
In function definition → Formal parameter

Call by Value.

Here actual parameters are evaluated and are copied into formal parameters. ∴ 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

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

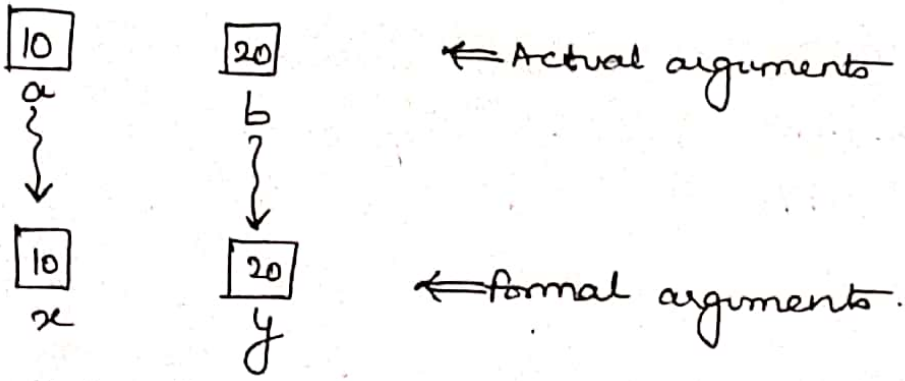
Output:

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.

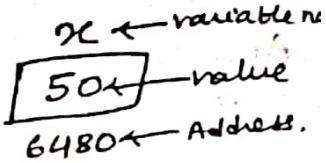
This is CALL BY VALUE.

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 th address of x, we use operator ampersand (&) for getting locatio 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 Variabl

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 ;]`

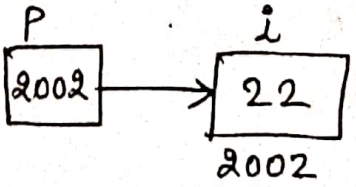
When declared, it is a good programming practice to initialize it to a value.

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

NOTE: [& : "Address of" operator
* : "Value at" operator]

eg.

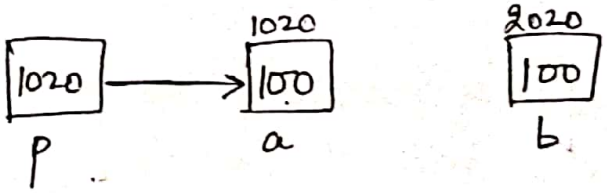
```
int i, *p;
i = 22;
p = &i;
printf("%d", i);  //22
printf("%d", *p); //22
printf("%d", p);  //2002
```



*p ≡ *(2002) ≡ 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
```

*(&a) = *(1020)
= 100

Call by Reference

Here address of actual arguments is copied into the formal arguments ∴ 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.

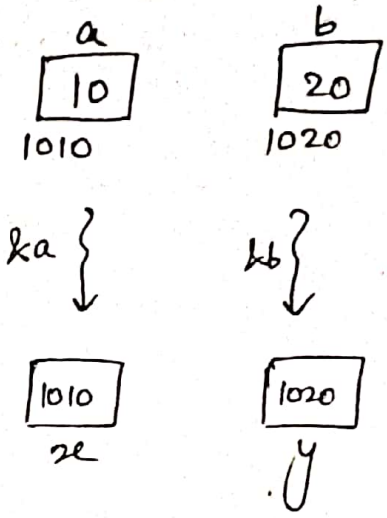
eg

```
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.



∴ x ⟶ 10 (a)

y ⟶ 20 (b)

∴ 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 → p = p+1 = 1020 + 4 = 1024 as float takes

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:

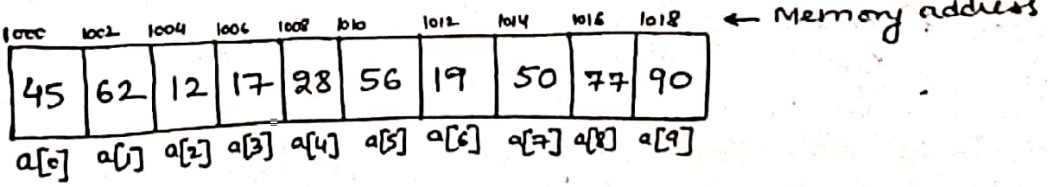
```
Void Swap(int *x, int *y)
{
    int temp;
    temp = *x;
    *x = *y;
    *y = temp;
}
```

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.

eg: collection of 10 integers



where 'a' is the name of array containing 9 integers.

for accessing 1st element → a[0]
for accessing 5th element → a[4]

Number within square brackets is called subscript.

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

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.

eg: int arr[5];

declares an integer array containing maximum 5 elements.

In C, array index begins with 0.

∴ a[0] → 1st element
a[1] → 2nd element
a[2] → 3rd element
a[3] → 4th element
a[4] → 5th element.

size = 5×2 = 10 bytes

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

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

∴ Total bytes = sizeof(datatype) × size of array

eg int arr[5];

Total bytes = 2 × 5 = 10 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, sizeof 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'.

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 seperate variables so array simplifies our code. It is used for concise and efficient programming.

```
int a[10], i, sum=0, float avg;
means Enter the 10 value by user
printf ("Enter the 10 number");
for (i=0; i<=9; i++)
    scanf ("%d", &a[i]);
for (i=0; i<=9; i++)
  sum = sum + a[i]        sum: 0+a(0)
                             =0+
  avg = sum/10.0;
printf ("Average is %d", avg);
getch();
```

eg. 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

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
```

~~Eg~~ WAP to print total positive numbers, negative numbers & zeros 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

Eg 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

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.

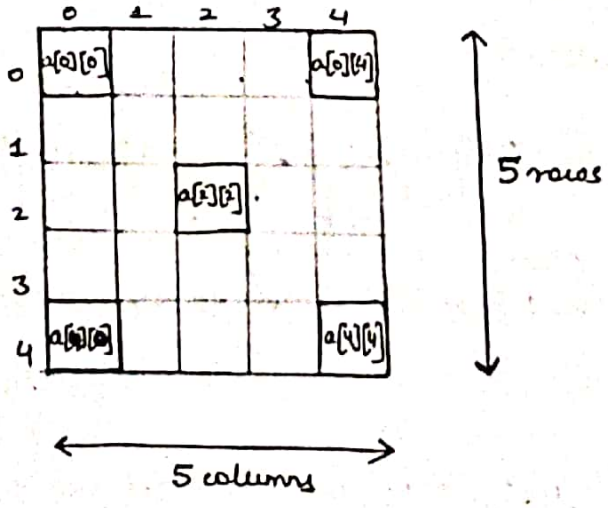
A two dimensional array is a grid having rows and columns in which each element is specified using two subscripts. The first subscript tells the row number, whereas second subscript tells the column number. Remember, the counting of rows and columns begin with Zero.

Declaration of 2D Array

syntax ⇒ datatype arrayname [no. of rows] [no. of columns];

eg: 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 ⇒ 5×5×2 ⇒ 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

$$0 <= i <= (m-1)$$
$$0 <= j <= (n-1)$$

Total bytes needed to store 2D array is ⇒

$$m \times n \times sizeof(datatype)$$

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 seperated 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 dimensional is optional.

int a[3][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

a[0][0]	a[0][1]	a[0][2]	a[1][0]	a[1][1]	a[1][2]
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 entered array */
printf("Entered array is:");
for(i=0; i<m; i++)
  {
    for(j=0; j<n; j++)
      {
        printf("%d", a[i][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

Q. 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(i=0; i<n; i++)
    {
        for(j=0; j<n; j++)
        {
            scanf("%d", &a[i][j]);
        }
    }
    printf("Entered Matrix is : ");
    for(i=0; i<n; i++)
    {
        for(j=0; j<n; j++)
        {
            printf("%d", a[i][j]);
        }
        printf("\n");
    }
    printf("Diagonal Elements are :-> ");
    for(i=0; i<n; i++)
    {
        for(j=0; j<n; j++)
        {
            if(i==j)
                printf("%d", a[i][j]);
        }
    }
    getch();
}
```

Output.

Enter order of square matrix

4

Enter way 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();
}
```

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 is

11	22	33
44	55	66
77	88	99

Eg. WAP to print TRANSPOSE of a matrix.(3×3)

```
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 3x3 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]==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
```

Prg:- WAP to multiply Two matrices of nxn 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]);
      }
```

output :

Enter dimension of square matrix

3

Enter Matrix A of order 3×3

1 2 3 0 0 1 1 2 0

Enter Matrix B of order 3×3

2 0 1 1 2 3 2 3 1

A×B is :
$$\begin{bmatrix} 10 & 13 & 10 \\ 2 & 3 & 1 \\ 4 & 4 & 7 \end{bmatrix}$$

concept

$$\begin{bmatrix} 1 & 2 & 3 \\ 0 & 0 & 1 \\ 1 & 2 & 0 \end{bmatrix} \times \begin{bmatrix} 2 & 0 & 1 \\ 1 & 2 & 3 \\ 2 & 3 & 1 \end{bmatrix} = \begin{bmatrix} 1\times2+2\times1+3\times2 & 1\times0+2\times2+3\times3 & 1\times1+2\times3+3\times1 \\ 0\times2+0\times1+1\times2 & 0\times0+0\times2+1\times3 & 0\times1+0\times3+1\times1 \\ 1\times2+2\times1+0\times2 & 1\times0+2\times2+0\times3 & 1\times1+2\times3+0\times1 \end{bmatrix}$$

Structures

Structures

* 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.

eg: 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;
```

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 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;
} b1, b2;
```

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.	It is used for storing database
eg: int arr[10]; float a[20]; char x[30];	eg. struct student { int rollno; char name[30]; }; 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.

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

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 name[30];
        char designation[25];
        char dept[20];
    } e1, e2;
```

Prg.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();
 }
```

11) WAP to copy a structure to another structure using pointer.

```
#include <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 (" Department : %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. emp[0] → first employee
emp[1] → second employee
emp[99] → Hundredth employee

* We can read and print the value of structure members using a loop.

11g Declare a structure which contains following 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();
}
```

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" };
```

8421
1010
0101
8421
-5