TAKEAWAYS — MODULE 1

Designing Efficient ~~Alg~~ Programs.

1. Flow charts
   - a → Flow chart symbols.
   - b → Sample Flow Charts
2. Algorithms
3. Pseudocode.
4. Difference b/w Algorithm & pseudocode.
5. Sample algorithms
6. Sample pseudocodes.
7. Programming paradigms.
   - a → Monolithic Programming
   - b → Procedural programming
   - c → Structured programming
   - d → Object Oriented programming.
   - e → Example Program.
8. Types of Errors.
   - a → Run time error
   - b → Compile time error
   - c → Linker error
   - d → Logical error.

Designing Efficient Prog's

#Program design tools:

Flow chart → Symbolical representation of the process. While designing the flowchart each step in the process is depicted by different symbols & is associated by short description.

→ Start & End Symbol OR → Oval shape represents Start/stop

→ Arrows

→ Processing Steps

→ I/p & o/p Symbols

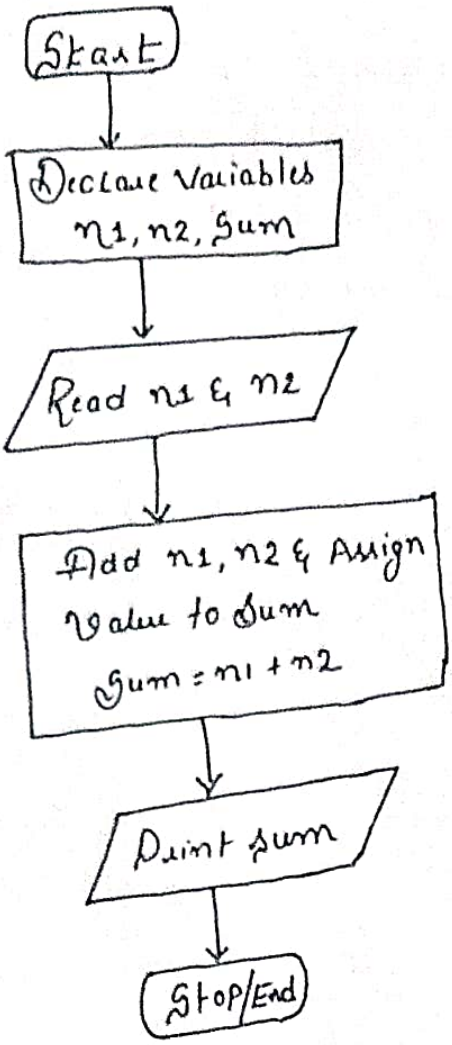
→ Decision Symbol

→ Connector.

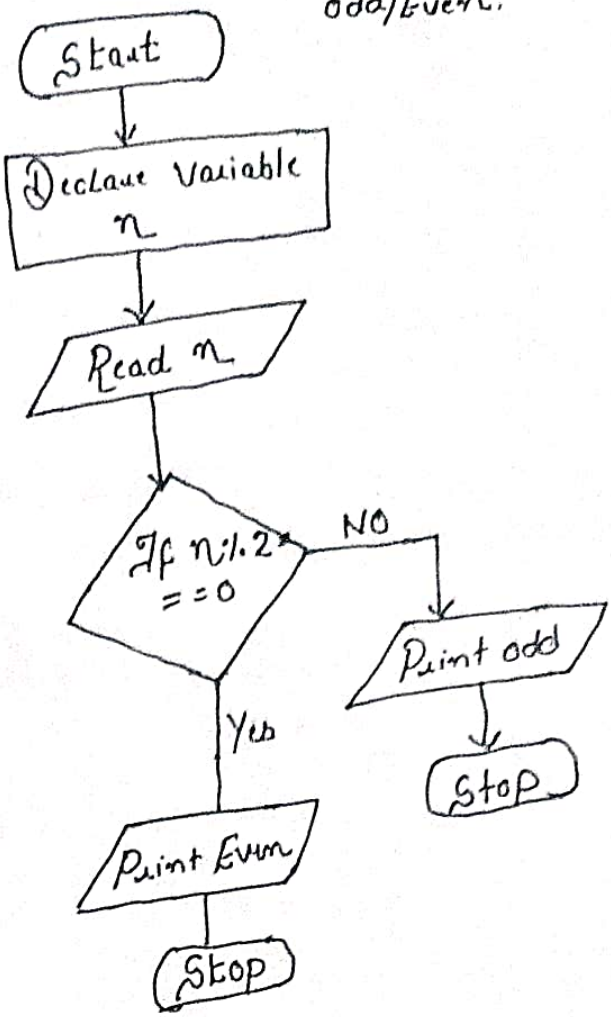
→ Represents Looping structure

Note: This material is for reference Purpose. For detailed Explanation Refer Prescribed text book.

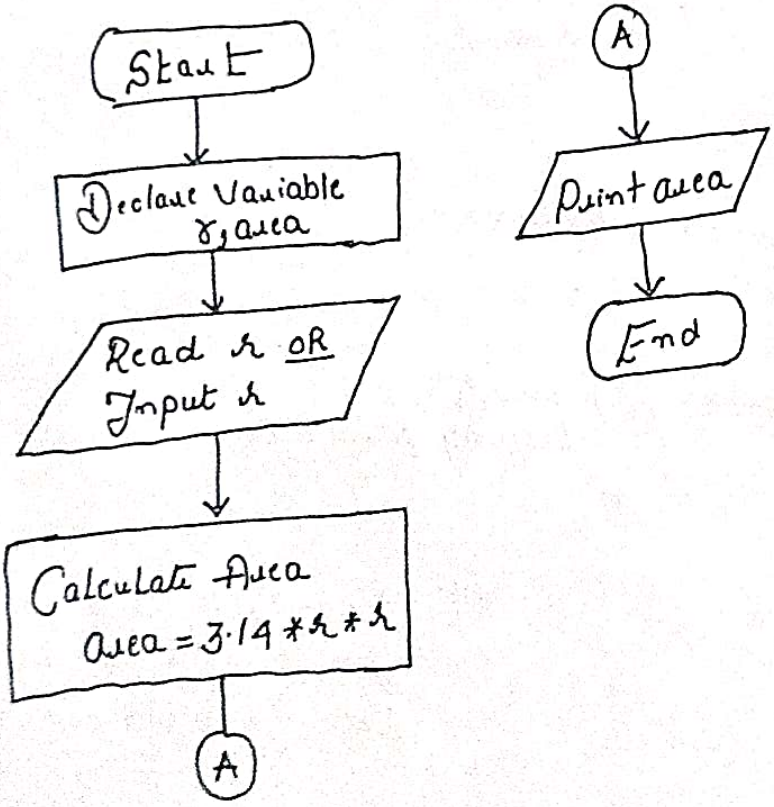
(1) Flow chart to add 2 no's



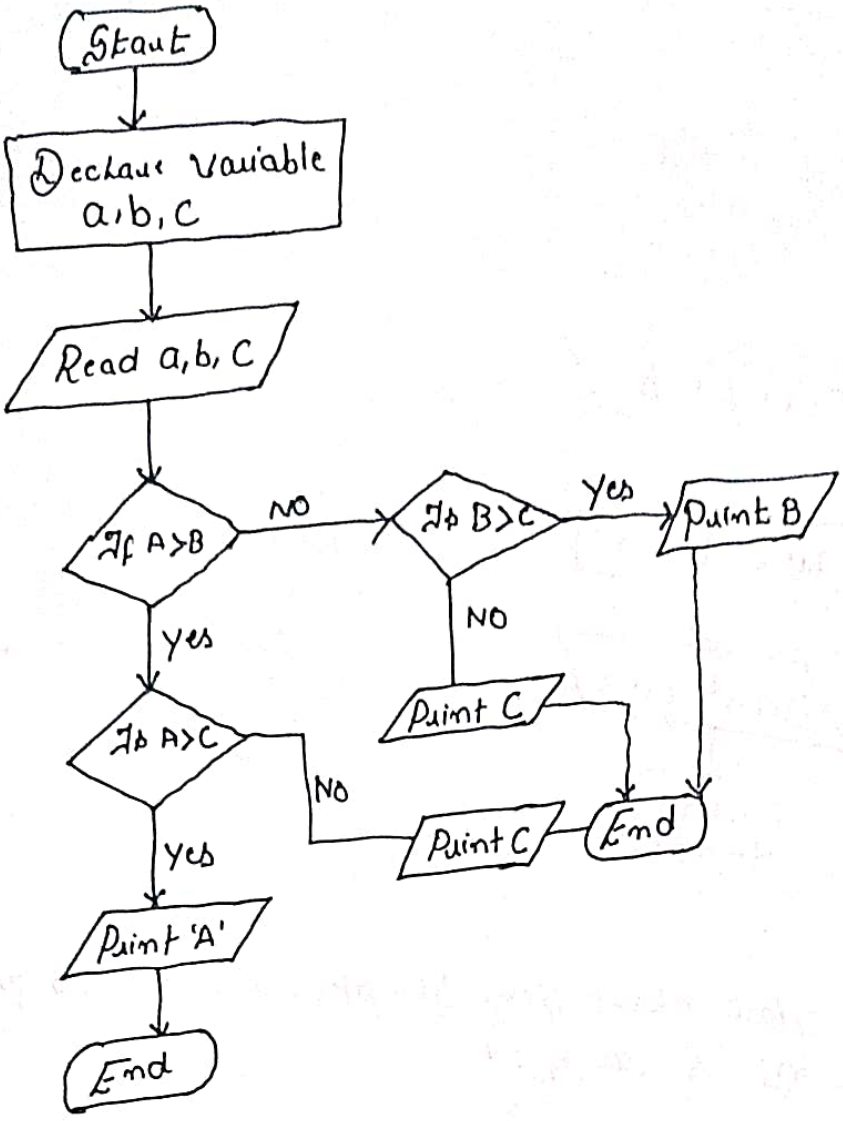
(2) Flow chart to find whether given number is odd/Even.



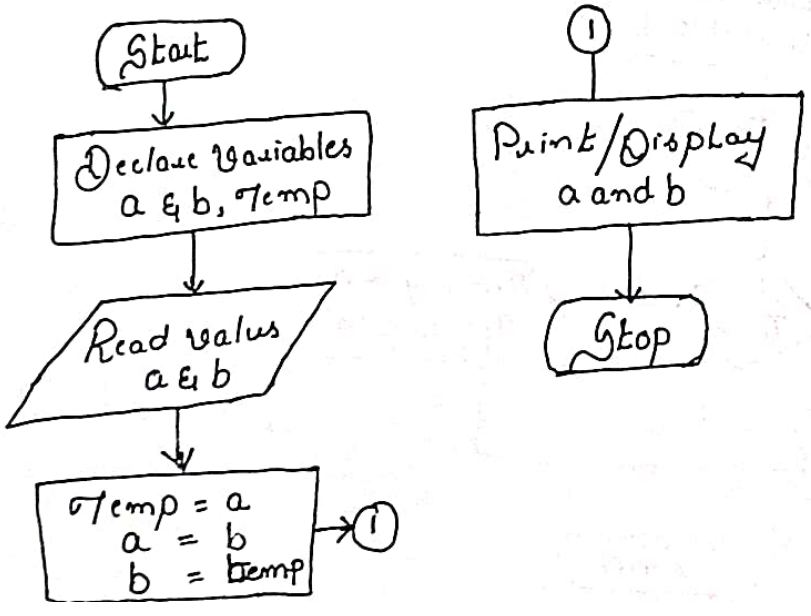
(3) Flow chart to find area of a circle.



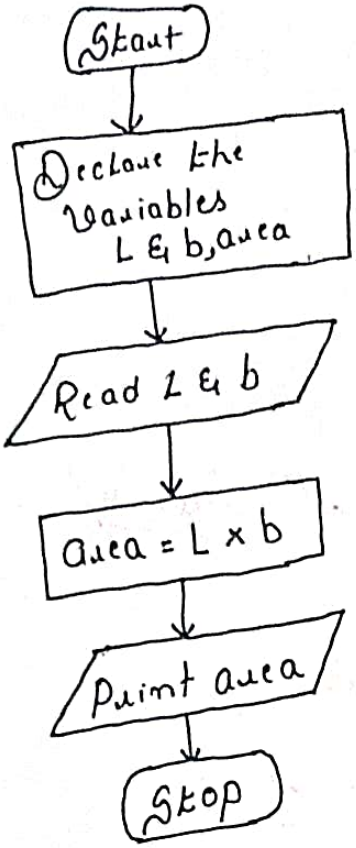
(4) Flowchart to find Largest of three No's



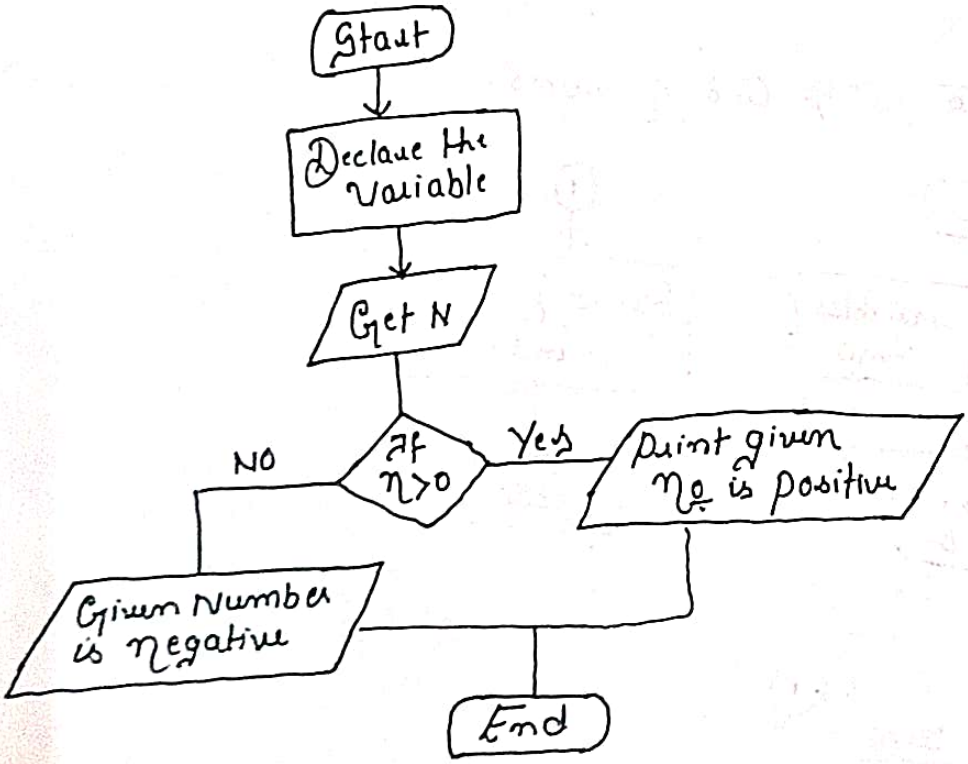
(5) Flow chart to swap two numbers.



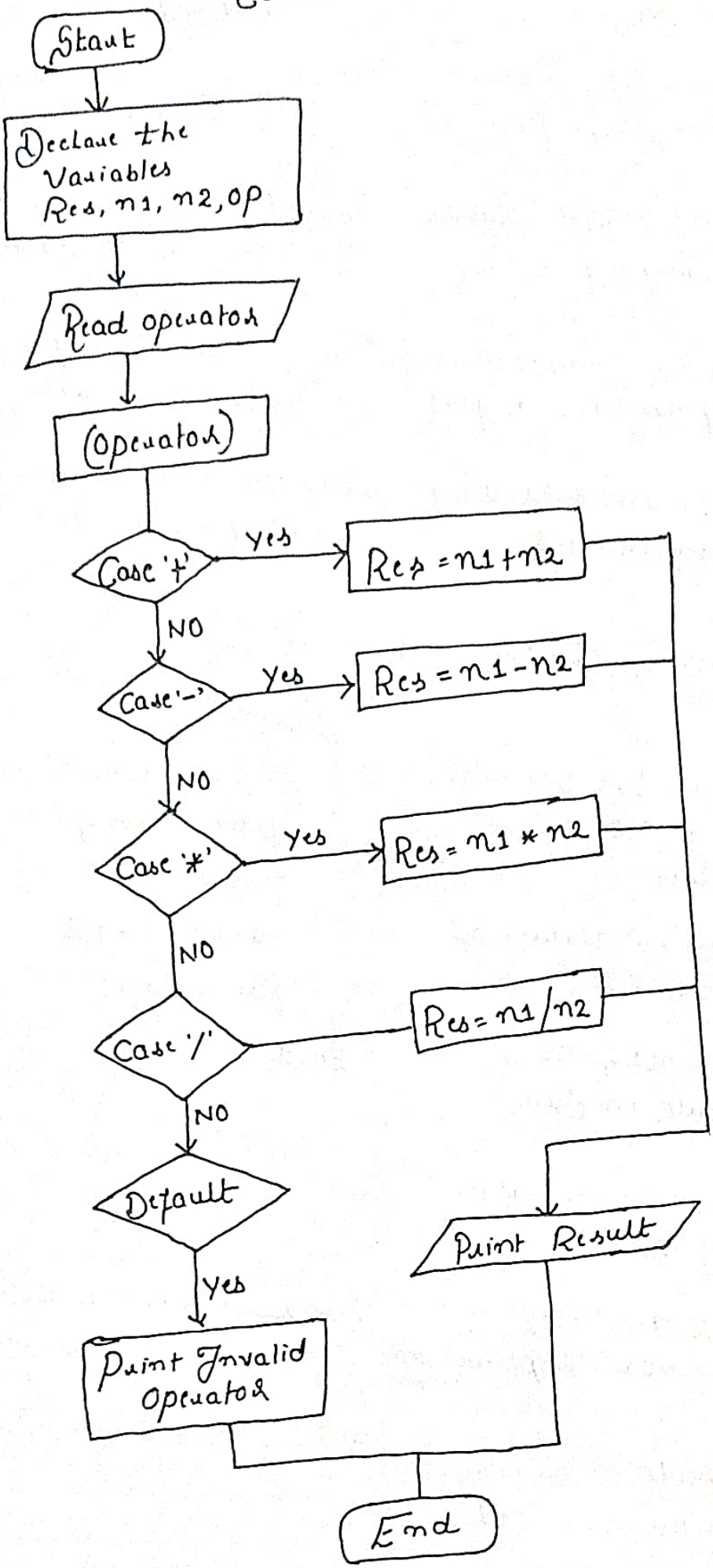
(6) Write a flow chart to find area of rectangle.



(7) Write a flow chart ~~for Simple Calculator~~ to find given no is +ve & -ve.



8) Flow Chart for Simple Calculator.



Difference b/w Algm & Pseudocode.

	Algorithm.	Pseudocode.
Defn	Step by step Procedure to Solve Specific Problem.	It's an Informal High Level Lang of an algorithm.
Purpose	Focuses on Logical approach In solving a problem.	Describes steps of an algorithm In a format closed to code.
Language	Abstract & Independent of any program Language	Mix of Natural Lang & program -ming like structure.
Readability	Easy to understand by Non Programmers	Easy for Programmers to under -stand & Convert into code.
Example.	Addition of two Numbers: 1. Start 2. Declare two variables n1 & n2 to store values. 3. Read the values of n1 & n2. 4. Add n1 & n2 & store result in sum. 5. Display the value of sum	Start Declare n1, n2 & sum as Variables Input num1/n1 Input n2 sum = n1 + n2 Print sum End.
Disadv.	1) Lack details of Practical Implementation. 2) Difficult to translate directly to the Code.	1) Requires basic understanding of Programming Language. 2) No Standard Syntax available.

Algorithms

1) To find whether given number is odd or Even.

1. Start
2. Declare variable 'n' to store value.
3. Input value of n OR Read n
4. If n % 2 = 0, then number is even.
5. Else number is odd.
6. Display/Print result
7. End.

Pseudocode.

```
Start
  Declare n as Variable
  Input n
  If n % 2 == 0 then
    Print "number is Even".
  else
    Print "number is odd".
End.
```

2) To find area of a circle.

1) Start
2) Declare variables r & area to store the values.
3) Input value of r.
4) Calculate area using formula area = $\pi \times r^2$.
5) Display or print area.
6) End.

```
Start
  Declare variables r & area.
  Input Radius of Circle r
  Set pi = 3.1459
  Set area = pi * r * r.
  Output area.
End.
```

Algorithm

3) To find Largest of three numbers.

1) Start.
2) Declare variables a, b & ~~Temp~~ c to store values.
3) Input values for a, b and c
4) If 'a' is greater than both b and c then a is Largest
5) Else if 'b' is greater then 'a' & 'c' then b is Largest
6) Else 'c' is Largest
7) Display Largest number.
8) End.

Pseudocode.

```
Start
  Input a
  Input b
  Input c
  if a>b and a>c then
     Largest = a.
  Else
  if b>a and b>c then
     Largest = b.
  Else
     Largest = c
  Print Largest value.
End.
```

4) To Swap two numbers.

1. Start
2. Declare variables a & b.
3. Input values for a & b.
4. Use Temp variable to store value of a.
5. Assign value of b to a.
6. Assign value of temp to b.
7. Display Swapped values a & b.
8. End.

```
Start
  Declare a, b, temp as variables
  Input a
  Input b
  temp = a  //Store value of a in temp
  a    = b  //assign value of b to a
  b    = temp //assign value of temp
                (original value of a) to
  Print a,b     b.
End.
```

Algorithm.

⑤ To find given number is Positive or negative.

1. Start
2. Declare variable n to store number.
3. Read value of n / Input n.
4. If n>0, then number is +ve.
5. If n<0, then number is -ve.
6. Else given number is zero.
7. Display result positive, negative or zero.
8. End.

Pseudocode.

```
Start
    Declare n as Variable
    Input n.
    if n>0 then
        Print "number is Positive".
    if n<0 then
        Print "number is negative"
    Else
        Print "number is zero".
End.
```

⑥ Simple Calculator.

1. Start
2. Declare n1, n2, sum to store values
3. Declare variable operator to store operations (+, -, *, /).
4. Read values of n1, n2 & operator
5. Based on operator Input:
   - If operator is '+' perform sum=a+b.
   - If operator is '-' perform sum=a-b
   - If operator is '*' perform sum=a*b
   - If operator is '/' perform sum=a/b
     Ensure b != 0 (b not equal to 0)
6. Display result
7. End.

```
Start
    Declare n1, n2, sum, operator as variable
    Input n1
    Input n2
    Input operator
    If operator == '+' Then
        result = n1+n2.
    Else
        If operator == '-' Then
            result = n1-n2.
    Else If operator == '*' Then
        result = n1*n2
    Else if operator == '/' Then
        result = n1/n2. (If n2 != 0)
        Else
            Print "Division by zero not
                   allowed".
            Exit
    Else
        Print "Invalid operator
    Print result.
```

List of Commonly used Pseudo code.

1) Input / Read :

→ Input a.
→ Read a.
→ Get a.
→ Accept a.
→ Obtain a.

2) output / Print

→ Output a
→ Print a
→ Display a
→ Show a
→ Write a.

3) Assignment

→ a = 5
→ Set a to 5
→ Let a be 5
→ Assign 5 to a

4) If-Else Condition.

→ If a>b then
→ If a is greater than b, then
→ If a exceeds b, then
→ If a is Larger than b, then

For Else part

→ Else
→ Otherwise
→ If not.

5) For Loops

For Loop:

→ For each i from 1 to n, do
→ For i = 1 to n
→ Loop through i from 1 to n

While Loop:

→ While Condition is true
→ Repeat while Condition
→ As Long as Condition

6) End Statements.

→ End if / End while / End for Loop
→ Terminate
→ Finish
→ Stop
→ End

7) Functions / Procedures.

Call function_name()
Invoke function_name()
Execute function_name().

Programming Paradigms.

1) Monolithic Programming:
   → Entire Program is written in single block of code without any functions
   → Changes in any one part of code affects entire program.
   → Examples of programming (Monolithic) Languages are:
      • Assemble Language.
      • Basic

2) Procedural Programming:
   → It breaks down the programs into procedures/functions
   → helps in performing specific task with clear flow of control.
   → Emphasis on code reusability
   → focuses on sequence of operations rather than data & behavior.

Adv: a) Prog's are easier to write compared to monolithic.
     b) only goal is to write correct progs.

Disadv: a) No concept of reusability
        b) Requires more time & effort to write prog's.
        c) Chances of changing global data & therefore may get errors.

3) Structered programming:
   → Suggested by mathematicians in 1966 by.
      • Corrado Bohm
      • Guiseppe Jacopini

→ Enforces use of Logical Controls structures like

Sequences, Loops & Conditionals } to improve Code Clarity & Readability.

→ Employs top-down approach.

Advantages

- Easy to understand & Edit.
- Each modules can be shared among programmers & then Integrate.
- Each module performs individual task.
- Each module has its own local data
- Easy to debug & understand.
- Structd prog takes less time to write Compared to any other progs.
- More Emphasis is given on the Code. & Less Importance is given to data.

Object Oriented Programming

→ Organises the code around objects that encapsulates both data & methods promoting.

- Abstract
- Inheritence
- Polymorphism

→ allows for Code-Reusable by focusing Interactions b/w objects.

Ex: To add two numbers.

Monolithic Programming.

```
#Include<stdio.h>
Int main()
{
   Int n1, n2, Sum;
   Printf("Enter two numbers");
   Scanf("%d %d", &a, &b);
   Sum = a+b;
   Printf("Sum = %d", Sum);
   Return(0);
}
```

Procedural Programming

```
#Include<stdio.h>
Int add(Int a, Int b);
{ Return(a+b);
}
Int main()
{
   Int n1, n2, Sum;
   Printf("Enter two numbers");
   Scanf("%d %d", &n1, &n2);
   Sum = add(n1, n2); // Call to the procedure.
   Printf("%d", Sum);
   Return(0);
}
```

Structured Programming.

```
#Include<stdio.h>
Int main()
{
   //Vaiable declaration.
   Int n1, n2, Sum;

   //Input.
   Printf("Enter values");
   Scanf("%d%d", &n1, &n2);

   //Computing/Processing.
    Sum = n1 + n2;

   //output.
   Printf("Sum = %d", Sum);
   Return(0);
}
```

Object oriented Programming.

```
#Include<iostream.h>
using namespace Std;

//Class definition
Class Calculator
{
  Public:
     //Method to calculate sum.
     Int add(Int a, Int b)
     { Return a+b;
     }
};

Int main()
{
  Int n1, n2, Sum;
  Calculator Calc; //object Creation
  Cout<<"Enter values";
  Cin>>n1>>n2;
  Sum = Calc.add(n1,n2);
  Cout<<"Sum:"<<Sum<<endl;
   Return(0);
}
```

(15)

Error	When it happens	Causes	When detected	Examples.
(1) Runtime Error (15)	occurs during Execution of the program.	when Invalid operations are Encountered. Ex: divide by zero, Invalid memory access.	Detected only when program ~~is Executed~~ runs	Segmentation Fault, array out of bound.
(2) Compiler Error	During Compilation	Syntax/Semantic Issues. Ex: Missing semicolons, wrong data type.	Detected when Compiled.	Missing ; type mismatch
(3) Linker Error	During Linking Phase after Compilation.	Missing object files or unresolved External symbols.	Detected during Linking process.	undefined Ref to a fn ~~definition~~ for a given prototype
(4) Logical Errors.	when there are Errors in the program Code.	Faulty program logic Leading to incorrect result.	Programmer has to find out. ~~to~~	Incorrect o/p, Infinite loop.

Types of Errors.

① Runtime
② Logical
③ Linker
④ Logical.

Module - 1

Topics Covered.

Introduction to Computers :

1. What is a Computer?
2. Explain basic Components of Computer system?
3. What are main functions of a Computer?
4. Generation of Computers?
5. List & Explain the main characteristics of Computer?
6. What is Computer Hardware?
7. Explain role of I/P & O/P devices in Computer?
8. Different type of Computers?

1) What is Computer

Computer is an electronic device that takes data, processes it and produces o/p based on the information.

OR

Computer is a programmable m/c designed to perform.
- Arithmetic operations
- Logical operations

Automatically & Sequentially for the given I/P by the user & get desired O/P.

2) Basic Components of a Computer.

Computer Components are categorised into two major Categories. They are.

→ Hardware Component

→ Software Component

Hardware Components

Hardware Components are the physical parts of the Computer such as

1) CPU
2) RAM
3) Mother board
4) Computer Data storage
5) Graphics Card
6) Computer Case.

CPU (Central Processing Unit)

→ It is also known as brain of a Computer.

→ CPU = ALU + CU. The Combination of ALU & CU together give CPU.

→ **ALU**

Arithmetic Logic Unit responsible to perform Arithmetic & Logical Operations such as

- Addition
- Subtraction
- Multiplication
- Division.

(Arithmetic operations)

- AND
- OR
- NOT
- XOR

(Logical operations.)

CU

Control unit does not Perform any actual data Processing but Controls Execution of Instructions by following Processes.

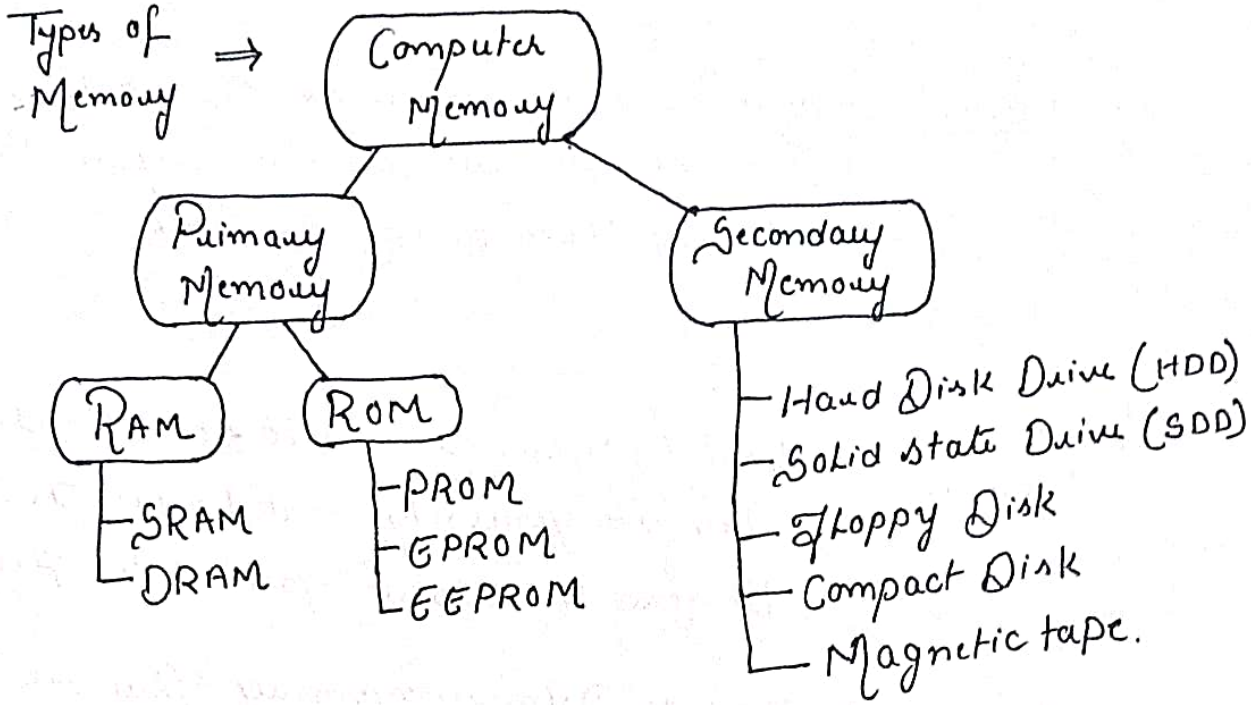
- Fetching Instruction
- Decoding &
- Executing Instn

Mother board → Mother board is the main Circuit board that Connects all hardware Components allowing them to Communicate.

Primary Memory ⇒ Built-in memory used to store the data They are generally located near the CPU to read the data at faster rate. They....

There are 2-types of Primary memory. They are
- RAM (Random Access Memory)
- ROM (Read only memory).

RAM	VS	ROM
1) Random Access Memory		1) Read only memory.
2) RAM is volatile Data is Lost when Computer is Powered-off		2) ROM is Non-volatice. Data is permanantly stored.
3) Read/write operation Can be performed		3) Only Read operation can be performed.
4) RAM is Expensive		4) ROM is Cheap Compared to RAM.
5) Variants DRAM, SRAM		5) Variants PROM (Programmable ROM) EPROM (Erasable programmable ROM) EEPROM (Electrically Erasable PROM).

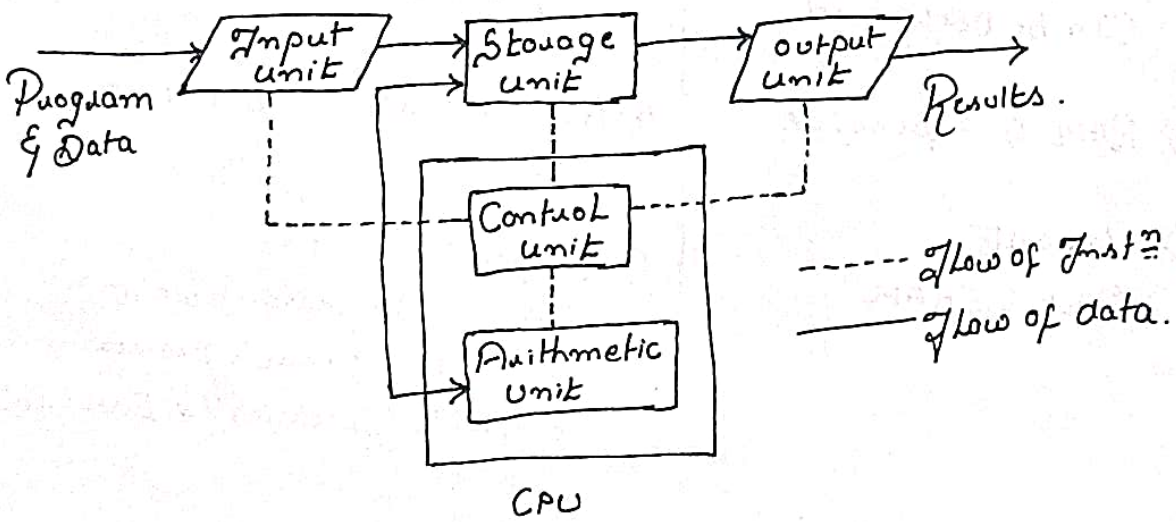


(3) Functions of the Computer :

The Computer performs 5 major functions. They are

→ Accepts data or Instⁿ as Input.
→ Stores data & Instruction
→ Processes data as per Instruction
→ Controls operations in the Computer
→ Produces results & provides o/p.

Basic Compⁿ organisation.



(4) Generation of Computers.

Generation	Time Period	Technology used	Example	Key features.
I st	1940 to 1956	Vaccum Tubes	ENIAC EDVAC UNIVAC IBM-701	• Large size • high power Consumption • Slow Performance.
II nd	1956 to 1963	Transi -stors	IBM 7090 UNIVAC 1108 CDC 1604	• Smaller size • Faster • Less heat • More Reliable.
III rd	1964 to 1971	Integra -ted Circuits	IBM system 360, PDP-8 PDP-11	• Even smaller Faster & more Efficient.
IV th	1971 - Present	Micro- -processor	Intel 4004 Apple II IBM PC	Micro Computers Personal computers GUI
V	Present - Future	• AI, • Quantum -Computing • Nano Technology	• AI applications • Advanced Processing Systems	IBM Watson Quantum Computers.

(5) Characteristics of Generation of Computers.

Generation Component	Programming Language	Memory	I/O devices	Speed & Size.
I st Vaccum Tubes	Machine Level Language	• Magnetic tapes • Magnetic drums	• Paper tape • Punched Cards	• Very slow • Very Large in Size.
II nd Transistors	• Machine Lang • Assembly Language	• Magnetic Core • Magnetic tape • Magnetic disk	• Magnetic tape • Punched Cards	• Smaller in size • Low power Consumption • Generated less heat
III rd Integrated Circuit	• High Level Language	• Large Magnetic Core • Magnetic tape/ disk	• Magnetic tape • Monitor • Keyboard • Printer etc	• Even smaller in size • Computers was Speed & reliability.
IV th VLSI	• High Level & Assembly Lang	• Semi Conductor memory RAM/ROM etc	• Pointing devices • optical Scanning • Keyboard • Monitor • Printer etc	• More speed Compared to Earlier Generations • Consumed less heat
V th • AI • Ultra Large Scale Integration (ULSI) • Parallel Processing methods	• Natural Processing Language (Human Lang)	• Semi Conductors memory	• Touchpad • Touch screen • Voice Recogn • Scanners • Keyboard • Monitor • Mouse	• Portable • Small in Size.

→ Above used millions of transistors in single chip & used 20x more MP to run the task.

Memory Contd

ROM → Non volatile memory that stores data permanantly used to store firmware (Progs which does not need to be modified frequently).

- PROM → Programmed only once. That is, Data can be written once & cannot be Erased.

- EPROM → Programs/Data can be erased by exposing chip to UV Light

- EEPROM → Programs/Data can be erased Electrically & Can be reprogrammed multiple times making more flexible than EPROM.

(8) Types of Computers → Device that transforms data to the meaningful info is know as Computer. The different types of Computer are.

- Super Computer
- Mainframe Computer
- Mini Computers
- Work Stations
- Personal Computer.

Super Computer.

Developed by.
Roger ~~Cray~~
- Cray
(1976)

→ They are biggest & fastest Computers in terms of processing the data.
→ Can process trillions of Instn in one second.
→ used in Scientific & Engg applns (weather forecasting, Scientific Simulation, Stock market, Nuclear Energy research etc).
→ Memory Consumed TB (Tera bytes) to PB (Peta bytes).
→ Disadvantage is Cost is too High.

Mainframe Comps

Mainframe Computers are designed to support hundreds to thousands of users at a time.
→ Supports multiple progs simultaneously.
→ Ideal for big organisations Like banking, Telecom, Medical field etc.
→ Works at very High speed (Millions of transactions/second)
→ Memory varies from Hundreds of GB to TB)
→ Cost is too High.

Mini Computers.

→ Mini Computers are medium sized multiprocessing Computers with 2 or more processors.
→ Support 2 to 400 users at a time
→ Mini Computers are used in places like Institutions Billing accounting etc.

→ Characteristics
- Fast processing
- Less Expensive Compared to Mainframe.
- Low weight.

Work station

→ Designed for technical or scientific application.
→ Consist of fast microprocessors with Large amount of RAM & speed graphic adopter.
→ Generally used by
  - Single user
  - used to perform specific task
  - Greate accuracy.

→ Chacteristics.
- Expensive.
- Designed for Complex Work.
- Has Large storage capacity
- Better graphics Card & powerful CPU Compared to PC.
- used to handle
  - Animation, Audio video Creation/Editing.
  - Data Analysis, CAD.

Personal Computer (Micro Computer)

→ Basically Generally Purpose Computers.
→ They are designed for Individual use.
→ Consists both Hardware & Software Components.
→ Purpose : Primarily used for tasks Like Word processing Browsing, Multimedia, Gaming & Programming.

Characteristics.
- Limited number of Softwares Can be used.
- Smallest in size
- Portable etc.

On Basic of data Handling Capability, there are 3 types of Computers.

① Analog Computer
② Digital Computer
③ Hybrid Computer.

	Features	Analog Computer	Digital Computer	hybrid computer.
①	Nature of data.	Continuous data. (Analog Signals Like voltage Current etc)	Discrete data (Binary data)	Combination of Combines both. Analog & Digital Data.

Feature	Analog Computer	Digital Computer	Hybrid Computer.
(2) Accuracy	Less accurate due to Continuous data repr^n	Highly accurate because of Precise binary Computation.	Balance of accuracy from both Analog & digital Component.
(3) usage	used where we don't need Exact values / need approximate values such as speed temperature, pressure Etc.	used where Exact values are required Ex: office work Gaming Etc.	used where Simulations are required where make use of both Analog & digital data.
(4) Processing	Parallel Processing	Sequential Processing	Combines both.
(5) Examples	Speedometers Voltmeters Thermometer Etc.	Laptop Desktop Smart phones PC, Notepad	Medical devices. ECG Scanning etc.

Ans to Question

(7) I/P and O/P Devices.

I/P Devices → Any device that provides I/P to the Computer.

- There are many I/P devices. Most Commonly used are Keyboard & Mouse.

List of I/p devices.

1) Keyboard
2) Mouse
3) Joystick
4) Lightpen
5) Scanner
6) Barcode Reader.

} Go through the text book.

List of o/p devices

1) Monitors
2) Printers
3) Plotters
4) Speakers
5) Projectors
6) Headphone (headset/Earphones).

1.2 Input and output devices

- These devices are used to enter information
- The functioning of a computer system is based on the combined usage of both input and output devices. Using an input device we can give instructions to the computer to perform an action and the device reverts to out action through an output device.

a) Input Devices

- An input device is any device that provides input to a computer. There are many input devices, but the two most common ones are a **keyboard** and **mouse**. Every key you press on the keyboard and every movement or click you make with the mouse sends a specific input signal to the computer.

List of Input Devices

Given below is the list of the most common input devices along with brief information about each of them.

1. **Keyboard**

- A simple device comprising keys and each key denotes either an alphabet, number or number commands which can be given to a computer for various actions to be performed
- It has a modified version of typewriter keys
- The keyboard is an essential input device and computer and laptops both use keyboards to give commands to the computer

2. **Mouse**

- It is also known as a pointing device
- Using mouse we can directly click on the various icons present on the system and open up various files and programs
- A mouse comprises 3 buttons on the top and one trackball at the bottom which helps in selecting and moving the mouse around, respectively
- In case of laptops, the touchpad is given as a replacement of the mouse which helps in the movement of the mouse pointer

Government

In government sectors, computers are used in data processing, maintaining a database of citizens and supporting a paperless environment. The country's defense organizations have greatly benefitted from computers in their use for missile development, satellites, rocket launches, etc.

Banking

In the banking sector, computers are used to store details of customers and conduct transactions, such as withdrawal and deposit of money through ATMs. Banks have reduced manual errors and expenses to a great extent through extensive use of computers.

Business

Nowadays, computers are totally integrated into business. The main objective of business is transaction processing, which involves transactions with suppliers, employees or customers. Computers can make these transactions easy and accurate. People can analyze investments, sales, expenses, markets and other aspects of business using computers.

Training

Many organizations use computer-based training to train their employees, to save money and improve performance. Video conferencing through computers allows saving of time and travelling costs by being able to connect people in various locations.

Arts

Computers are extensively used in dance, photography, arts and culture. The fluid movement of dance can be shown live via animation. Photos can be digitized using computers.

Science and Engineering

Computers with high performance are used to stimulate dynamic process in Science and Engineering. Supercomputers have numerous applications in area of Research and Development (R&D). Topographic images can be created through computers. Scientists use computers to plot and analyze data to have a better understanding of earthquakes.

3. **Joy Stick**

- It is a device which comprises a stick which is attached at an angle to the base so that it can be moved and controlled
- Mostly used to control the movement in video games
- Apart from a computer system, a joystick is also used in the cockpit of an aeroplane, wheelchairs, cranes, trucks, etc. to operate them well

4. **Light Pen**

- It is a wand-like looking device which can directly be moved over the device's screen
- It is light-sensitive
- Used in conjunction with computer's cathode ray tube

5. **Microphone**

- Using a microphone, sound can be stored in a device in its digital form
- It converts sound into an electrical signal
- To record or reproduce a sound created using a microphone, it needs to be connected with an amplifier

6. **Scanner**

- This device can scan images or text and convert it into a digital signal
- When we place any piece of a document on a scanner, it converts it into a digital signal and displays it on the computer screen

7. **Barcode Reader**

- It is a kind of an optical scanner
- It can read bar codes
- A source of light is passed through a bar code, and its aspects and details are displayed on the screen

All the devices mentioned above are the most commonly used input devices. Several other such types of equipment are used in different fields, which can be counted as an input device.

b) Output devices

Monitor: Programs use it to display text or graphical output and users need to view the input that they key in.

Printers: produce hard copies of output which include text and graphics.

Plotters: Used to makes line drawings, uses one or more automated pens to complete each line before taking up the next one., either the plotter pen moves or the paper moves or both to draw the lines

List of Output Device

The commonly used output devices have been listed below with a brief summary of what their function is and how they can be used.

1. **Monitor**

- The device which displays all the icons, text, images, etc. over a screen is called the Monitor
- When we ask the computer to perform an action, the result of that action is displayed on the monitor
- Various types of monitors have also been developed over the years

2. **Printer**

- A device which makes a copy of the pictorial or textual content, usually over a paper is called a printer
- For example, an author types the entire book on his/her computer and later gets a print out of it, which is in the form of paper and is later published
- Multiple types of printers are also available in the market, which can serve different purposes

3. **Speakers**

- A device through which we can listen to a sound as an outcome of what we command a computer to do is called a speaker
- Speakers are attached with a computer system and also are a hardware device which can be attached separately

- With the advancement in technology, speakers are now available which are wireless and can be connected using Bluetooth or other applications

4. **Projector**

- An optical device which presents an image or moving images onto a projection screen is called a projector
- Most commonly these projectors are used in auditoriums and movie theatres for the display of the videos or lighting
- If a projector is connected to a computer, then the image/video displayed on the screen is the same as the one displayed on the computer screen

5. **Headphones**

- They perform the same function as a speaker, the only difference is the frequency of sound
- Using speakers, the sound can be heard over a larger area and using headphones, the sound is only audible to the person using them
- Also known as earphones or headset

Introduction to 'C'

Features of 'C' programming Language

Features	Description
① Simple	'C' is easy to learn with limited set of keywords & syntax.
② Structured Programming	Supports use of { Functions, Loops, Control structures } for Coding
③ Rich Library & built-in operators	Supports wide range of built-in and library functions for writing 'C' code.
④ Portability	'C' can be compiled with different m/c with little or no modification.
⑤ Modularity	The concept of storing 'C' prog in the form of Library can be further used. This is known as modularity.
⑥ General purpose language	'C' Lang can be used in almost all appln ranging from system s/w to stand alone appln/scientific applications.
⑦ Middle-Level Language	'C' is also middle-level language because uses functionalities of high-level lang combined with low-level memory management & H/w interaction.

History of 'C' Lang

	Year	Development	Description
(1)	1960	ALGOL (Algorithm Language)	Developed by Commitee of European & American Computer Scientist. • Friedrich. L. Bauer (Germany) • Peter Naur (Denmark) • John Backus (USA).
(2)	1967	BCPL (Basic Combined Programming Language.	• Developed by Martin Richards. • Primarily used for writing System Software.
(3)	1970	Creation of 'B' Language	• Developed by Ken Thompson. • Used to Create Early Versions of UNIX operating Systems.
(4)	1972	Birth of 'C' Language	• Developed by Dennis Ritche at Bell Labs • Improved Version of 'B' Lang adding Data type & Structures.
(5)	1973	'C' Lang used in UNIX	• Unix operating System was re-written in 'C' Lang making it portable & more efficient.
(6)	1978	'C' Prog Language by K&R C	• Kernighan & Dennis Ritche Published 'C' prog Lang & became Very Popular.

	Year	Development	Description.
(7)	1983	ANSI 'C' Standardization	American National Standard Institute (ANSI) appointed Technical Commitee to standardize 'C' & was globally known as "ANSI-C"
(8)	1989	ANSI C(C89)	ANSI releases first version of Standard version of 'C' & known as C89.
(9)	1990	ISO Standardi-zation (C90)	ANSI 'C' was then approved by ISO (International Standard Orgn) & officially known as C90
(10)	1999	C99 Standard released	• Introduced new features to 'C' Lang such as a) Inline functions b) Variables-length array & other features of C++/Java.

Summary.

1960 - ALGOL	International group.	
1967 BCPL	Martin Richards	
1970 'B' Lang	Ken-Thompson	
1972 'C' Lang	Dennis Ritche	
1978 K&R 'C'	Kernighan & Ritche.	
1989 ANSI 'C' C89	ANSI Commitee approval.	
1990 ISO C90	ISO Commitee approval	
1999 C99	Standardization Commitee.	

Structure of 'C' Program.

Documentation Section. Used for Comments.
Link Section #Include <Stdio.h> #Include <Conio.h>
Definition Section. Void Func();
Global declaration Section (Variables used in more than one function)

Void main()
{
Declaration part
Executable part
}

Sub programs (user defined functions) Function 1 Function 2 ⋮ Functionn

Example program.

```
/* Program to add 2 numbers */
#Include<stdio.h>
void add (int a, int b);
Int Sum;
void main()
{ Int n1, n2;
  Pf("Enter values");
  Sf("%d %d", &n1, &n2);
  add (n1, n2);
  Pf ("Sum = %d", Sum);
}
Void add (Int a, Int b)
{
  Sum = a+b; //accessing
}                global variables
```

Files used in 'C' Program.

Files used in 'C'
- Source File
- Header Files
- Object File
- Executable File.

Source Code File

- This file includes source code of the program.
- These files are saved with extension .c or .cpp.
  Ex: programname.c / programname.cpp.
- main() is starting point of the program.

Header files.

- These files have an extension .h.
- 'C' provides with some standard header files
- Some standard header files are.

a) Stdio.h → used for Input & output functions.

b) StdLib.h → Contains fn for.
- Memory allocation
- Process Control & Conversations.

Ex: malloc(); , exit();

c) math.h → Contains mathematical functions
Ex: Sqrt(), pow() etc.

d) string.h → Provides string manipulation functions

e) time.h → Provides function for manipulating time & date.
Ex: time(), clock(), date()....

f) limits.h → Defines Constant for size limit for data types.
Ex: INT_MAX, CHAR_MIN....

#object files.

- Object files are the files generated by the compilers as source code file is processed.
- These files have extensions .obj
- These files contains machine code but not executable. binary code.

#Executable file ⇒ It a compiled program that can run directly by the operating system.

- These files have Extension .exe.
- Generated by Compilers & Linkers.

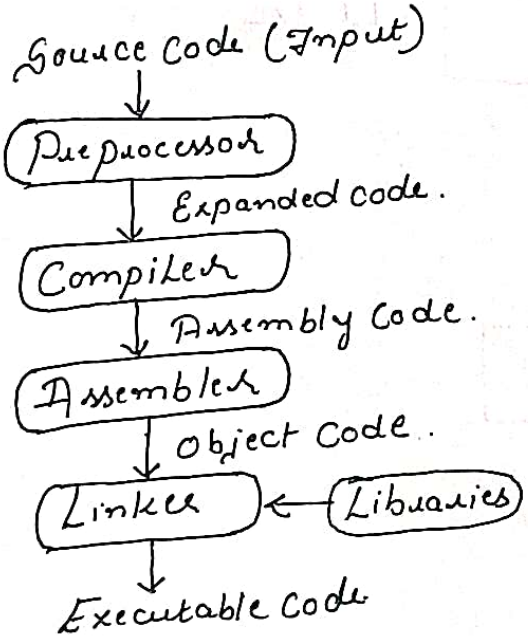
Difference b/w Compiler, Interpretor & Assembler.

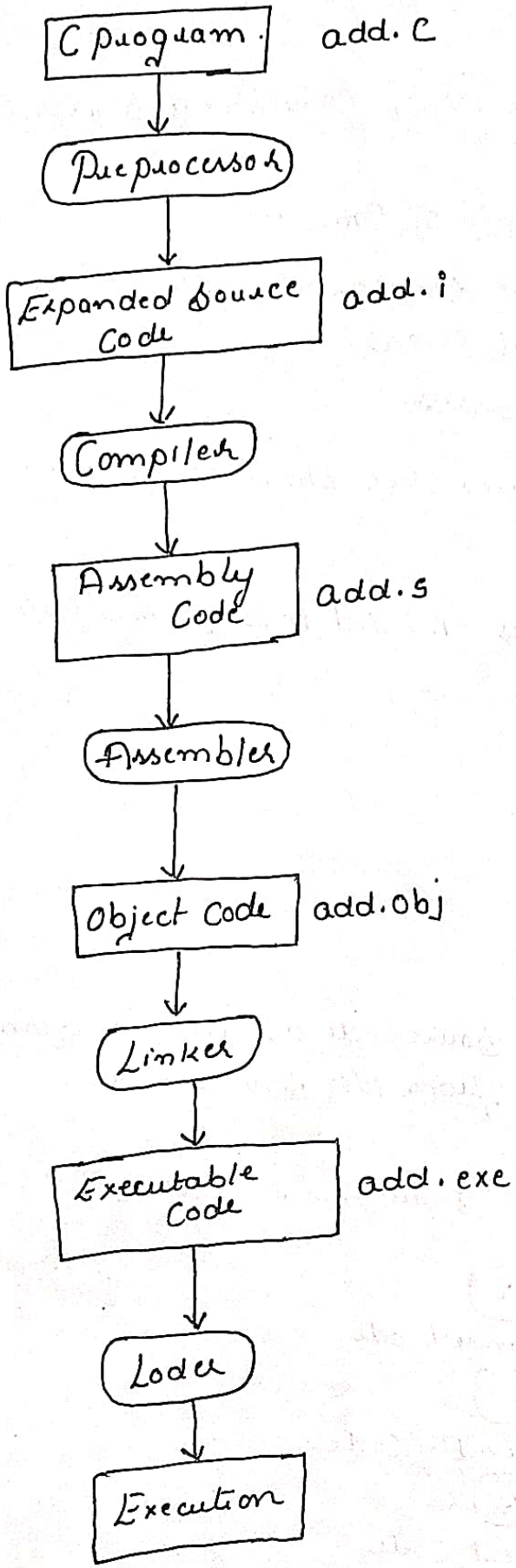
	Compiler	Interpreter	Assembler.
Defn	Translates Entire source code into machine code before Execution.	Translates source code Line-by-Line at run-time	Translates assembly language code into machine code.
outputs	Generate Executable file .exe, .out.....	Executes the code directly.	Generates object file .o, .obj
usage	used for high-level programming lang such as C, C++ & Java	used for scripting languages such as Javascripts, python Ruby Etc.	used for Low Level prog machine languages like such as assembly language.
Memory	Requires more memory for entire program to be Compiled	Requires less memory since it processes one line at a time.	uses small amount of memory to generate machine code.

Compiling & Executing 'C' Progs.

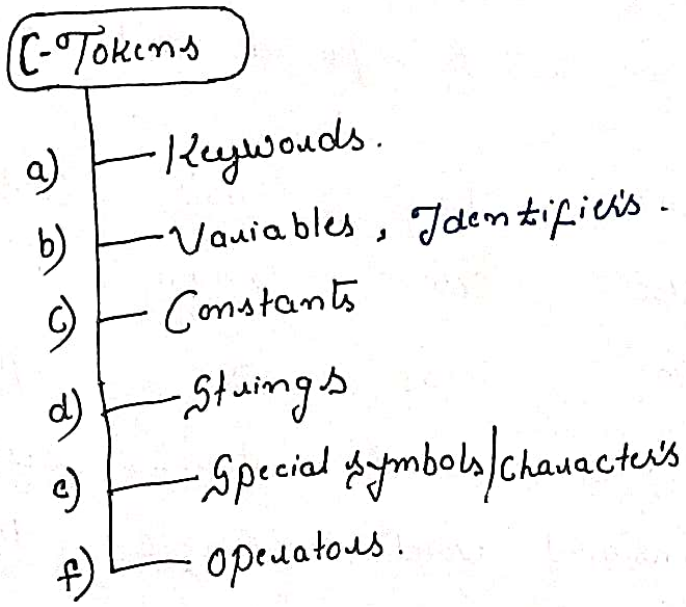
- Compilation is the process of converting source code to object code.
- It is done with the help of Compiler
- COmpiler checks the source code for
  a) Syntactical Errors
  b) Structual Error's.
- If source code is Error free then it generates the object code.
- Compilation process is divided into four steps.
  1) Preprocessor.
  2) Compiler
  3) Assembler
  4) Linker.

Preprocessor takes source code as an I/p, removes all the Comments from the source-code.




C program
add.c
Preprocessor
Expanded Source Code
add.i
Compiler
Assembly Code
add.s
Assembler
Object Code
add.obj
Linker
Executable Code
add.exe
Loder
Execution

C-Tokens → Basic building blocks in 'C' Language or Smallest Individual units in a 'C' program. Every program is constructed using combination of below said tokens.



Like natural Lang Computer Lang also uses a character set that defines fundamental unit used to represent Info. Character set therefore given as

a. English alphabets
   a to z.
   A to Z

b. Digits from 0 to 9

c. Special characters
   ~, @, %, ^, &, *,
   {, }, (,), [,], $, /,
   \, ', ", “, ...

d. white space characters.

e. Escape Sequence.

White space characters are as given below.

\b - blank space.
\t - Horizontal tab
\v - Vertical tab
\r - Carriage return
\f - formal feed
\n - New line.

Keywords : Reserved words OR words that are specific to 'C' Language is known as Keywords.

→ The keywords are basically set of characters that have fixed meaning

→ ALL keywords must be written in Lower case.

→ Below table shows List of keywords used in 'C'.

auto	break	Case	Char	Const	Continue
default	double	else	enum	extern	float
for	goto	int	float	Long	register
return	short	signed	size-of	struct	switch
typedef	union	unsigned	void	volatile	do
if	static	while.			

Identifiers

Identifiers as name suggest help us to identify data & other objects in the program.

→ Identifiers are basically the names given to program Elements such as variables, arrays, fn etc.

→ Identifier may consist sequence of letters, numerals & underscore.

Rules for Naming Identifiers.

1) Allow Characters : Identifiers can only contain Letters digits & underscore. No special char can be used.

A-Z ⇒ Can be used.
a-z ⇒ Can be used
'_' ⇒ Can be used.
Special Char ⇒ Cannot be used.

Special Characters
Ex: #, $, ^, ?, "
etc.

2) ~~Key words should~~ Identifiers must start with Letter or underscore. They cannot start with digit.

Ex: _myvar → valid Identifier.
$um → Valid Identifier
num1 → valid Identifier.
1stnumber → Invalid.
1myvar → Invalid.

3) Identifiers are Case Sensitive.

Ex: Num1, num1
num2, Num2.

Num1 & num1 are treated as two different identifiers.

4) Keywords cannot be used as Identifiers such as int, return, float etc (Refer Pg 26 for keywords).

5) Spaces are not allowed while writing keywords.

Ex: myvar → valid
my var → Invalid.

6) Length of Identifiers:

There is no Limit to the Length of Identifiers as such. But keeping identifiers reasonable for readability is a good practice.

Note: Some Compilers Consider first 31 characters to distinguish identifiers. So it is advisible for Identifiers to be with-in this Limit.

Variables:

Variable is a name given to the memory location where the value is not fixed & can change with course of time. There are basically two kinds, they are.

a) Numerical variable

b) Character variables.

a-Numerical variable

→ used to store Integer values or Floating point values.

→ Numerical variables may also be associated with modifiers such as Long, short, signed & unsigned.

b - Character variables.

→ Stores only single character enclosed with single quote.

→ These characters could be any character from ASCII Character set. i.e..

Letters ⇒ ('a' to 'z', 'A' to 'Z')

Numerals ⇒ (0 to 9) Ex ('2')

Special characters.

Declaring variables.

→ Each variable must be declared with data type before its name.

Ex: int num1;.
datatype ↳ variable.

→ Variable name must follow rules of Identifiers. (Refer pg-27)

- Cannot contain Letters & underscore.
- Must begin with Letter & underscore.
- Variable name should not contain Reserved keywords
- No spaces or special Characters allowed.

→ Declare the variable before you make use of it.

→ Initialization of value to the variable may be optional.

Ex: int num1 = 10
float num2 = 10.5.

Note: If variables are declared but not initialized, they usually contains garbage values.

There might be small Confusion b/w Identifiers and Variables.

Let us Consider one simple prog "Addition of two numbers". to know which are the Identifiers & what are all the variables.

```
#Include<stdio.h>
Int main()
{
   Int n1, n2, Sum;
   Printf("Enter two numbers");
   Scanf("%d %d", &n1, &n2);
   Sum = n1 + n2;
   Printf("Sum = %d", Sum);
   Return(0);
}
```

main :

main is name of function name & we cannot declare or initialize the value & hence this is identifier.

n1, n2, Sum :

Identifiers → n1, n2, Sum are names (Identifiers) for the variables.

Variables → These Identifiers represents memory location

n1 is a variable to store first number.

n2 is a variable to store second number

Sum is variable to store the result.

Return :

This is neither Identifier or variable. They are keyword.

Printf, scanf :

Identifiers → Names of standard Input & O/p functions in 'C'. These are not variables but does specific operation.

Constants: Constants are Identifiers whose values do not change during the execution of a program. Unlike variable, Constants holds fixed value throughout the program.

Types of Constants in C.

a) Integer Constant
- → Represents whole number
- → They can be positive, negative or zero.
- → They can be decimal, octal or hexadecimal.

Ex:
```
Int num1 = 10    //Decimal Constant
Int num2 = 032   // octal Constant
Int num3 = 0x15  // Hexadecimal Constant.
```

b) Floating Point Constants
- → Represents real number that has fractional part
- → They may be expressed in standard notation or Scientific notation.

Ex:
3.14, -2.5... (Standard Notation)
1.5e (Scientific Notation equivalent to 1500)

6.02×10^3 — Scientific
6020 — Standard

6.02×10^{-3} — Scientific
0.00602 — Standard.

Note: if Exponent is +ve shift decimal places to right else left.

c) Character Constant

→ Represents single character enclosed with-in single quote.

→ They store ASCII value of the character.

d) String Constants

→ Represents sequence of characters enclosed with-in double quote.

→ Always string constants end with null char ('\0') which indicates end of the string.

Ex: char name[] = "Ramesh"; // String Constant

5) Enumerated Constants

→ User defined datatypes that consists of named Integer Constants.

Ex: enum week {Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday};

Here Sunday is 0
Monday is 1 & so on.

Example prog that demonstrates Different types of Constants

```
#include<stdio.h>
int main()
{
    int decimalNum = 100   // Decimal Constant
    int octalNum = 012     // Octal Constant
    int hexNum = 0xA       // Hexadecimal Constant
```

```
float PI = 3.14159; //Floating point Constant
Char grade = 'A' ;   //Character Constant
Char name [] = "Ramesh"; //String Constant.
Enum week {Sunday, Monday, Tuesday, Wednesday,
           Thusday, Friday, Satuday} ;
Enum week today = Monday;
```

//Enumerated Constants

```
Printf ("Decimal number = %d", decimalNum);
Printf ("Octal number = %0 \n", octalNum);
Printf ("Hexa number = %x \n", hexNum);
Printf ("Floating point Num = %f \n", PI);
Printf (" Character Const = %C \n", grade);
Puntf (" String = %s\n"; name);
Printf("Today is : %d[Monday) \n";today);
Return(0);
}
```

O/P

```
Decimal number = 100
Octal number   = 12
Hexa number    = A
Floating point Num = 3.14159.
Character Const = A
String = Ramesh.
Today is : 1 (Monday).
```

Rules Operators.

Operators are special symbols used to perform operations on variables & values. They can be categorized into.

→ Arithmetic operator
→ Relational operator
→ Equality operator
→ Logical operator
→ Unary operator
→ Conditional operator
→ Bitwise operator
→ Assignment operator
→ Comma operator
→ Sizeof operator.

Arithmetic operators.

- Used for mathematical operations.
- Some of them are +, −, *, /, %
- Below table shows arithmetic operators. their syntax & usage.

Ex: a=9, b=3, Result.

Operation	operator	Syntax	Comment	Result.
Multiply	*	a * b	Result = a * b	27
Divide	/	a / b	Result = a/b	3
Addition	+	a + b	Result = a+b	12
Subtraction	−	a - b	Result = a-b	06
Modulus	%	a % b	Result = a % b	0

Note: While performing modulo division, sign of the result is always sign of first operand

Ex: 16 % 3 = 1
16 % -3 = 1
-16 % 3 = -1
-16 % -3 = -1

Relational operator.

- Used to Compare two values b/w two operands
- We can also check for Equality of operands whether the operand is greater, smaller than the other.
- Below table shows Relational operators, usage & its syntax.

operator	Meaning	Example.
<	Less than	3<5 gives 1
>	Greater than	7>9 gives 0
>=	Greater than or Equal to	100<=100 gives 1
<=	Lesser than or Equal to	50>=100 gives 0
==	Equal to	100==100 gives 1
!=	Not Equal to	100!=90 gives 1

Prog to demonstrate Relational operators.

```
#include<stdio.h>
void main()
{
    int x=10, y=20;
    Printf("%d < %d = %d", x, y, x<y);
    Printf("%d == %d = %d", x, y, x==y);
    Printf("%d != %d = %d", x, y, x!=y);
    Printf("%d > %d = %d", x, y, x>y);
    Printf("%d >= %d = %d", x, y, x>=y);
    Puntf("%d <= %d = %d", x, y, x<=y);
}
```

o/p

10 < 20 = 1
10 == 20 = 0
10 != 20 = 1
10 < 20 = 0
10 >= 20 = 0
10 <= 20 = 1

Logical operators.

→ used to combine or invert boolean Expression.

→ There are 3 types. They are

- Logical AND
- Logical OR
- Logical Not

Logical AND:

Logical AND (&&) operator is a binary operator which return the o/p if both the values are true.

Truth table for Logical AND.

A	B	A&&B
0	0	0
0	1	0
1	0	0
1	1	1

Returns o/p if both are true.

{ 0 → False.
{ 1 → True.

Ex: (a<b) && (b>c)

Expression always is evaluated from left.

{ a = 2
{ b = 10
{ c = 5.

a < b b > c.

2 < 10 ⇒ True. 10 > 5 True.

if both Expressions are true the whole Expression is True.

Logical OR:

Logical OR (||) return true if any one of the operand is true Else returns False.

A	B	A || B
0	0	0
0	1	1
1	0	1
1	1	1

Ex: (a<b) || (b>c)

a < b 10 > 5

2 < 10 = True. True

o/p ⇒ True.

a = 2
b = 10
c = 5

Logical NOT (!)

Logical NOT operator takes single expression & negates the value of the Expression. In other words it just reverses value of the expression.

A	!A
0	1
1	0

Ex: int a=10, b;
b = !a

Sample prog illustrating Logical operator.

```
#include <stdio.h>
int main()
{
    int a=5, b=10, c=0;
    if (a>0 && b>0)
    { printf("Both a & b are positive");
    }
    if (a>0 || c>0)
    { printf("Either a or c is positive");
    }
    if(!c)
    { printf("C is zero");
    }
  return(0);
}
```

Logical AND

Logical OR

Unary operators

Unary operator act on single operand. 'C' Lang supports three unary operators.

- Unary Minus
- Increment operator
- Decrement operator

Unary Minus (-)

unary minus is different from Arithmatic subtraction operation. In arithmatic operation it gives the difference of two operations. But in unary minus it negates the value & produces the o/p.

Ex: Int a, b=10;
a = -(b);.

Now a = -10. value b is positive number. After applying unary minus operator on b, values becomes -10 & is assigned to a.

Increment operator and Decrement operator.

→ Increment operator is a unary operator which increments the value by 1 and Decrement is a unary operator which decrements the value by 1.

→ They have two variants < Prefix. (++a, --a)
postfix. (a++, a--).

Note a++ is not same as ++a.
a-- is not same as --a.

Sample Prog illustrating Unary operators.

```
#Include<stdio.h>
Int main()
{
    Int a=10, b=-5, c=0;
    Printf("unary plus on a: +a = %d\n", +a);
    Printf("unary plus on b: -b = %d\n", -b);

    // Increment/Decrement Operators.
    Printf("Initial value of a = %d", a);
    Printf("Post Increment: a++ = %d", a++); // Returns 10 & then Increments to 11
    Printf("Pre Increment: ++a = %d", ++a); // Increments to 12 & then returns 12
    Printf("After post Increment: a = %d", a);
    Pf("Initial value of b: %d", b);
    Pf("Post decrement: b-- = %d", b--); returns -5 & then decrements to -6.
    Pf("After post decrement: b = %d", b);
    Pf("Pre decrement: --b = %d", --b); decrements to -7 & then returns -7

    Pf("Logical Not on c: !c = %d", !c); // 0 becomes 1

    return(0);
}
```

O/P

```
Unary plus on a : +a = 10
unary plus on b : -b = 5
Initial value of a: 10
Post. Increment : a++ = 10
After post Increment : a = 11
Pre Increment : ++a = 12
Initial value of b: -5
Post decrement: b-- = -5
after post dec : b-- = -6
Pre decrement : --b = -7
Logical NOT on C: !C = 1
```

Conditional operator (?:)

The Conditional operator or the ternary (?:) is just like an if Else statement that can be used within expressions These Expressions are useful in situations ~~like~~ in which there are two or more alternatives for an Expression. 'C' allow nested Conditional Statements also.

Ex: Jnt a,b,Large.
Large = (a>b)?a:b

Ex 2: Jnt a,b,C,Small.
5 3 7
Small = (a<b ? (a<c ? a : c) : (b<c ? b : c))

Ex prog to find Largest of 3 numbers using Conditional operator.

```
#include<stdio.h>
void main()
{
    Jnt n1, n2, n3, Large;
    Printf("Enter the values : \n");
    Scanf("%d %d %d", &n1, &n2, &n3);
    Large. = n1 > n2 ? (n1 > n3 ? n1 : n3) :
                       (n2 > n3 ? n2 : n3));
    Printf("Largest of three numbers is %d", Large);
}
```

Bitwise operators

Bitwise operators are those operators that performs operations at bit Level. These operators include

1. bitwise AND
2. bitwise OR
3. bitwise XOR
4. Shift operators

} These operators expect their operands to be Integer & treat them as Sequence of bits.

Bitwise AND

Bitwise AND performs operation on bits rather than bytes. Character & Integers. When we use bitwise AND operator the bit in first operand is ANDed with corresponding bit with second operand.

A	B	AND
0	0	0
0	1	0
1	0	0
1	1	1

Ex: Consider a = 10, b = 20
Then it's written as.
c = a & b; in the code.

a = 10 ⇒ 1010 (Binary)
b = 20 ⇒ 10100 (Binary)

To make the comparison easier we shall represent both numbers with 8-bit.

ie 10 ⇒ 0000 1010
20 ⇒ 0001 0100 &
0000 0000

10 & 20 = 0 in decimal.

Steps to follow

1) Convert to Binary
2) Perform bitwise AND
3) Conclude result.

// Sample 'C' Program to Show bitwise AND.

```
#Include<Stdio.h>
Int main()
{
    Int a = 4; // In binary 0100.
    Int b = 3; // In binary 0011
    Int Result = a&b;
    Printf("Bitwise AND operation Performed on
            %d and %d is %d"; a, b, Result);

    Return(0);
}
```

Step 1 : Convert 4 & 3 to binary.

0100 & 0011.

Step 2 : Perform AND operation.

```
  0100
& 0011
------
  0000
------
```

Ex: 2 : Consider a = 5 (0101)
b = 3. (0011)

```
  0101
& 0011
------
  0001
------
```

Bitwise AND of 5 & 3 is 1.

Bitwise OR (|)

Bitwise OR operator makes use of the symbol (|). The bit in the first operand is ORed with the bit in second operand.

A	B	OR
0	0	0
0	1	1
1	0	1
1	1	1

Ex: a=10, b=20

C = a|b.

```
  00001010  ⇒10
| 00010100  ⇒20
----------
  00011110  ⇒ 30 Result.
```

```
//Sample program to demonstrate bitwise OR
#include<stdio.h>
int main()
{
   int a=10,b=20, result;
   result = a|b  //Bitwise OR operation.
   printf("Bitwise OR of %d and %d is %d",
                    a,b, result);

   return(0);
}
```

Bitwise XOR

Bitwise XOR operator (^) performs operation on individual bits. The bit in first operand is XORed with corresponding bit of second operand.

A	B	XOR (A^B)
0	0	0
0	1	1
1	0	1
1	1	0

If both the I/p's are '0' or '1' then O/p is zero else 1.

Ex: a = 10, b = 20.

10 in binary 0000 1010
20 in binary 0001 0100

Then
```
  00001010
^ 00010100
----------
  00011110  (30)
```

Bitwise XOR of 10 & 20 is 30

// Sample Prog to illustrate bitwise XOR.

```
#Include<stdio.h>
void main()
{
    Int a=10, b=20, c;
    Result = a^b;
    Printf("Bitwise XOR of %d and %d is %d",
                a,b,Result);
}
```

Bitwise NOT (~)

Bitwise NOT or Complement is unary operator that performs logical negation on each bit of the operand. It actually produces 1's Complement of the given binary value.

// Sample prog demonstrating bitwise NOT.

```
#Include<stdio.h>
void main()
{  Int a=10, Result;
   Result = ~a;
   Printf("Bitwise NOT of %d is %d\n", a, Result);
}
```

Step 1: Convert value into binary.
a = 10 ⇒ 0000 1010

Step 2: Flip all bits
1111 0101 (-11)

Shift operators.

'C' supports two bitwise shift operators
1) Shift Left (<<)
2) Shift Right (>>)

Ex: if x = 0001 1101
x << 1 produces 0011 1010

x << 4 ⇒ 0001 1101.
0001 1101 0000
o/p.

x >> 1 (Right shift by 1).

x ⇒ 0001 1101.
x >> 1 = 0000 1110

//Sample prog to demonstrate shift operators.

```
#include<stdio.h>
int main()
{
  int a = 10; LS, RS;
    LS = a<<1;     //Left shift by 1-bit
    RS = a>>1;     //Right shift by 1-bit.
  Pf("Original value = %d", a);
  Pf("Left shift of a by 1 is %d", LS);
  Pf("Right shift of a by 1 is %d", RS);
  Return(0);
```

Assignment operators

These operators are responsible to assign values to the variables. The (=) sign is fundamental assignment operator.

Ex: x = 10 // assigns value 10 to variable x.

Note : Assignment operators always has right to left associativity.

Ex: a = b = c = 10 ;

is Evaluated as

(a = (b = (c = 10)));

Now value 10 is assigned to 'c', then value of 'c' is assigned to 'b' then finally 'b' is assigned to 'a'.

other assignment operators.

1) /= ⇒ variabe /= Expression ⇒ variable = variable / Expression.

Ex: a = 6, b = 2 } a /= b ⇒ a = a/b ;

2) += ⇒ var += Exp ⇒ var = variab + Exp.

Ex: a = 6, b = 2 } a += b ⇒ a = a + b ;

III) *= ⇒ a *= b ⇒ a = a * b ;

-=

%=

^=

<<=

>>=