

FRANKLIN'S  
LECTURES

S2 KTU Exam

ക്ലാസ്സേറ്റ്

PROGRAMMING IN C



20 MAY MON 10 AM

Franklin Founder & CEO

EST102

# PROGRAMMING IN C

MODULE 1



# **SYLLABUS**

## **Basics of Computer Hardware and Software**

Basics of Computer Architecture: processor, Memory,  
‘Input & Output devices

Application Software & System software: Compilers,  
interpreters, High level and low level languages

Introduction to structured approach to programming,  
Flow chart Algorithms, Pseudo code

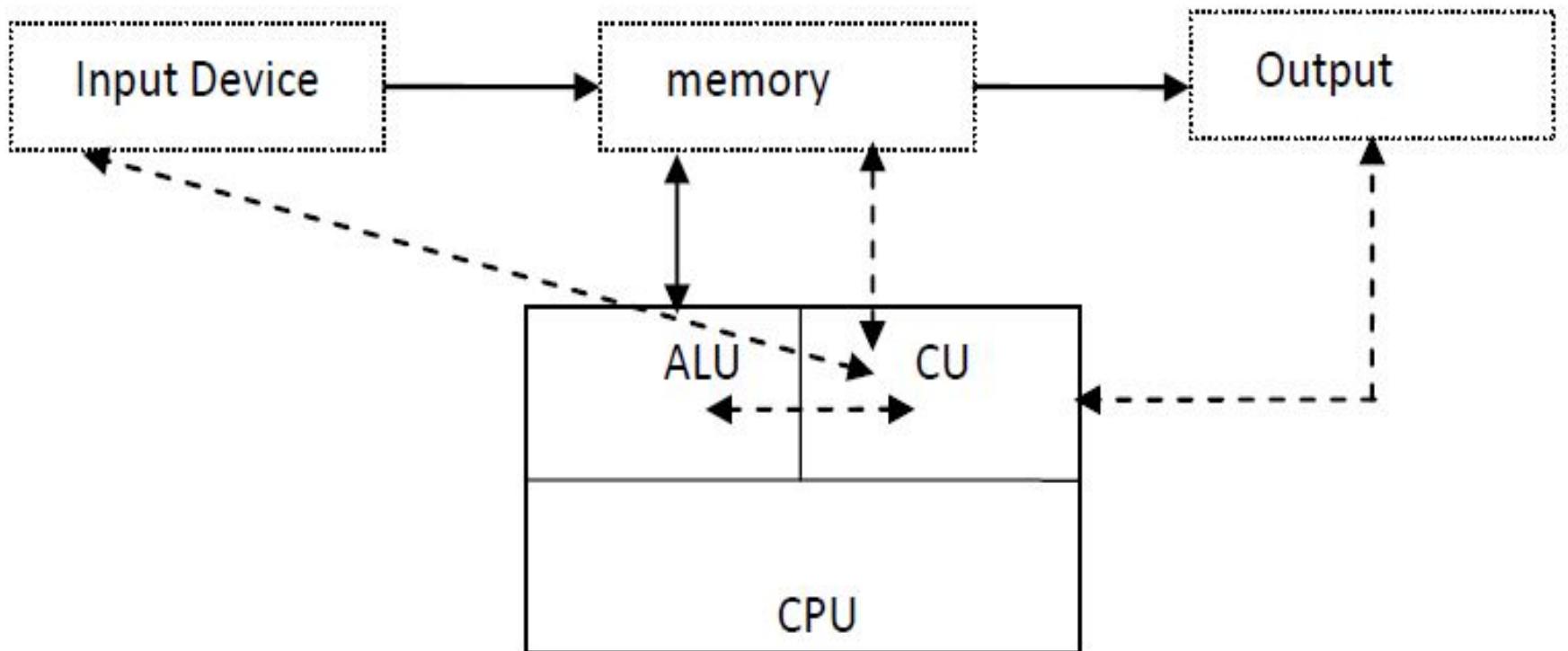
# *Important!!*

- *Memory Hierarchy*
  - *RAM VS ROM*
  - *System Software VS Application Software*
  - *Language Processors*
  - *High Level Languages VS Low Level Languages*
  - *Properties of good algorithm*
  - *Symbols used in flowchart*
  - *Algorithm*
  - *Flowchart*
  - *Pseudocode*
- } *Linear Search, Bubble Sort*

# Basics of Computer Architecture



# Block Diagram of Computer



# Central Processing Unit ( CPU )

- CPU is responsible for the overall working of all components of the computer.
- It consists of two parts:
  - 1) Arithmetic and Logic Unit(ALU)
  - 2) Control Unit (CU)

# Central Processing Unit ( CPU )

## 1) Arithmetic and Logic Unit(ALU)

The ALU performs arithmetic operations and conducts the comparison of information for logical decisions.

# **Central Processing Unit ( CPU )**

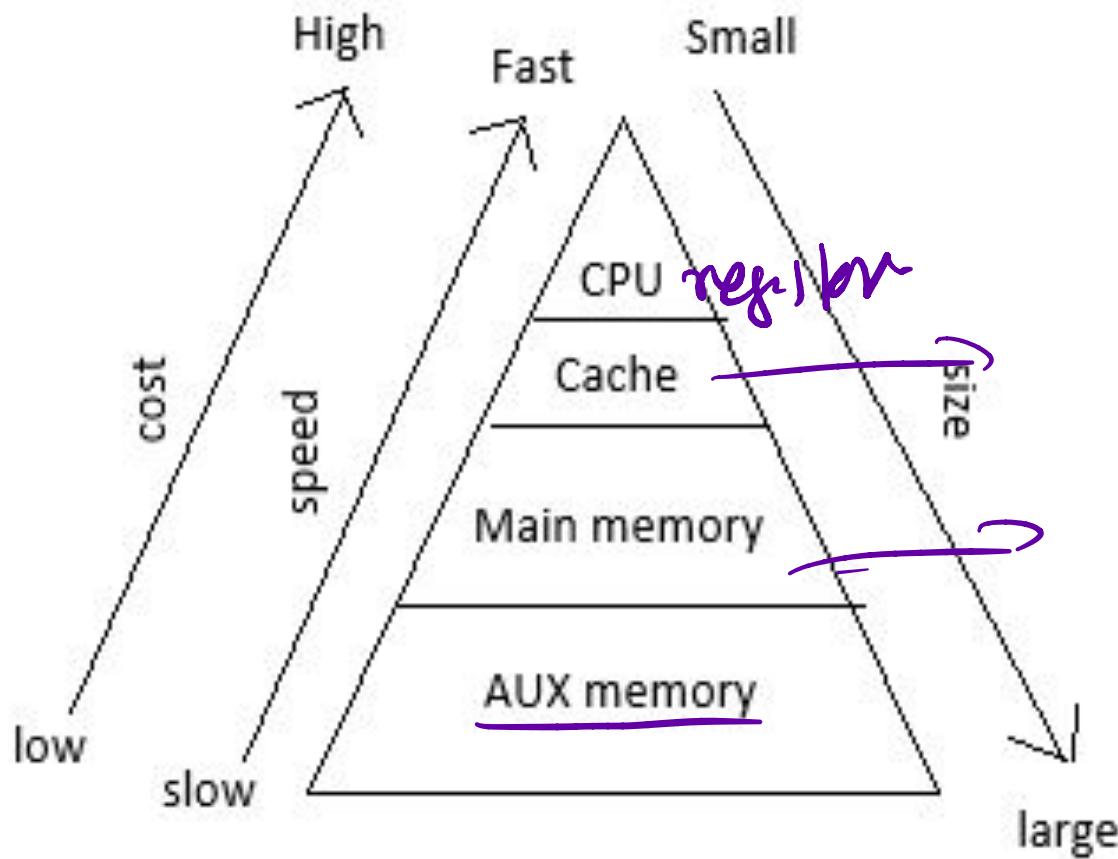
## **2) Control Unit (CU)**

The control unit is responsible for sending / receiving the control signals from / to all components.

# Memory

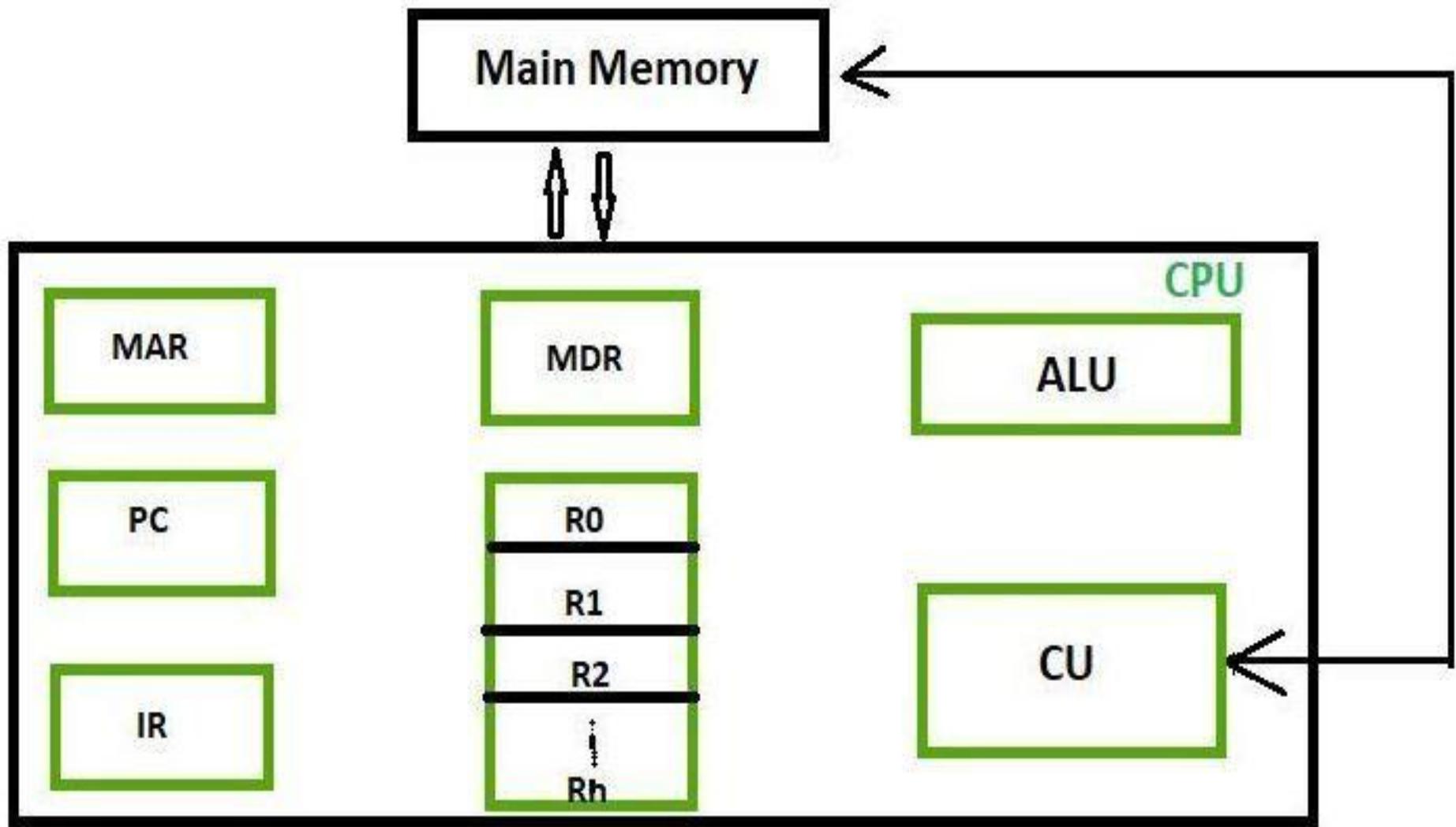
- The memory unit of the computer is used to store data, instructions for processing data, intermediate results for processing and the final processed information.
- The memory of the computer of two types:
  - 1) Primary Memory
  - 2) Secondary Memory.

# Memory Hierarchy



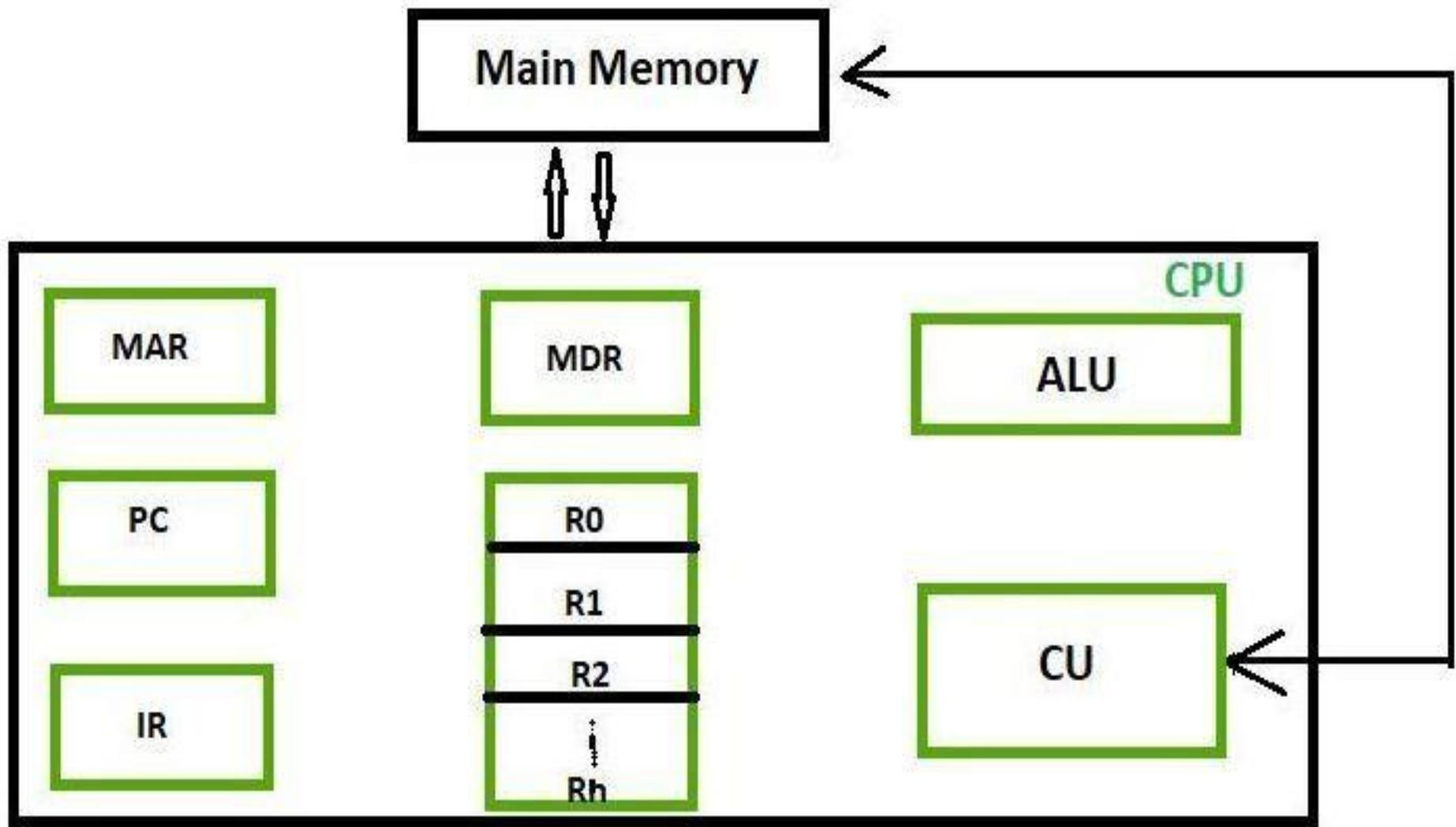
# CPU Registers

- Registers are very fast computer memory which are used to execute programs and operations efficiently.
- This is done by giving access to commonly used values.



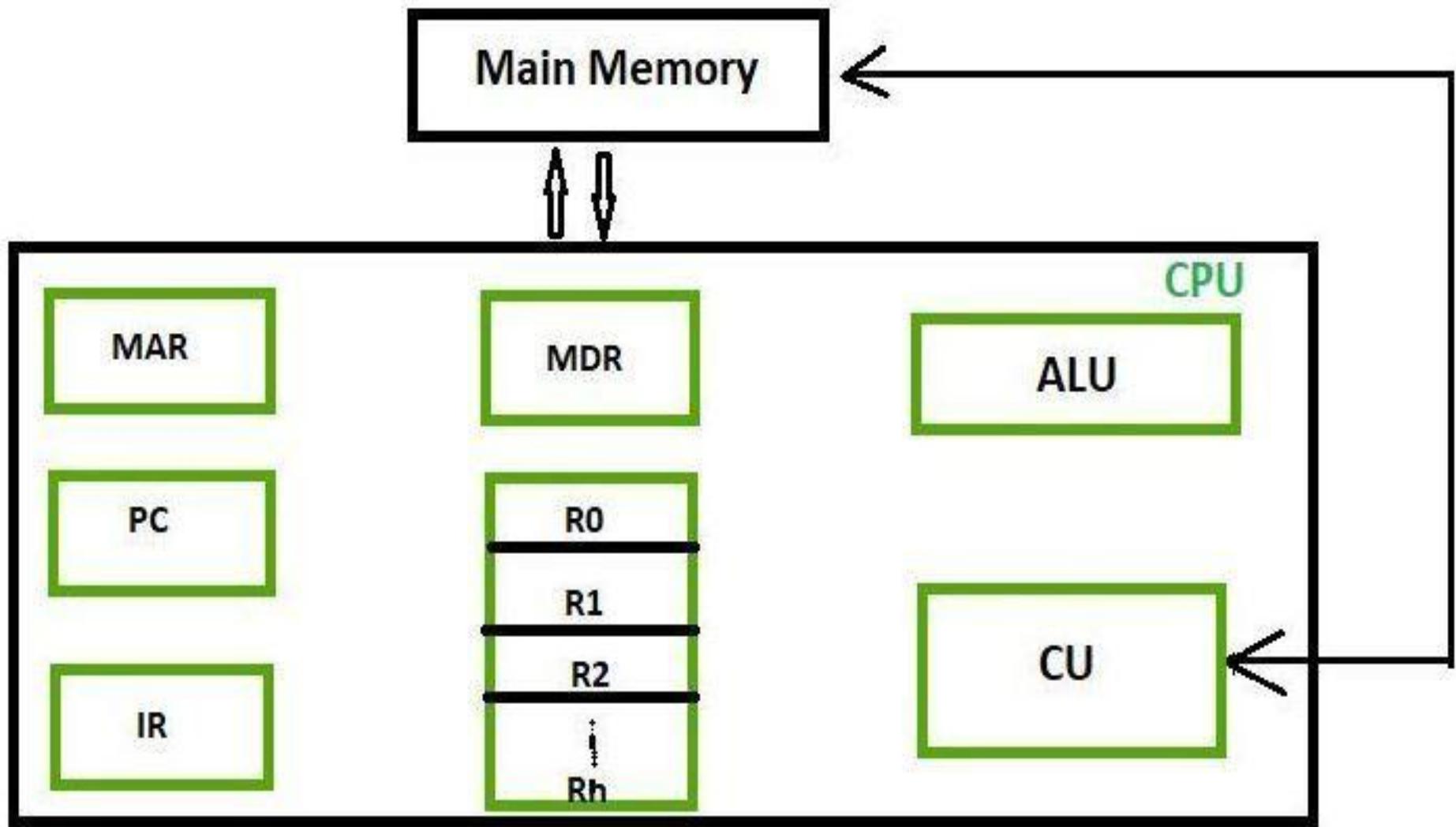
Accumulator ✓

Most frequently used register used to store data taken from memory.



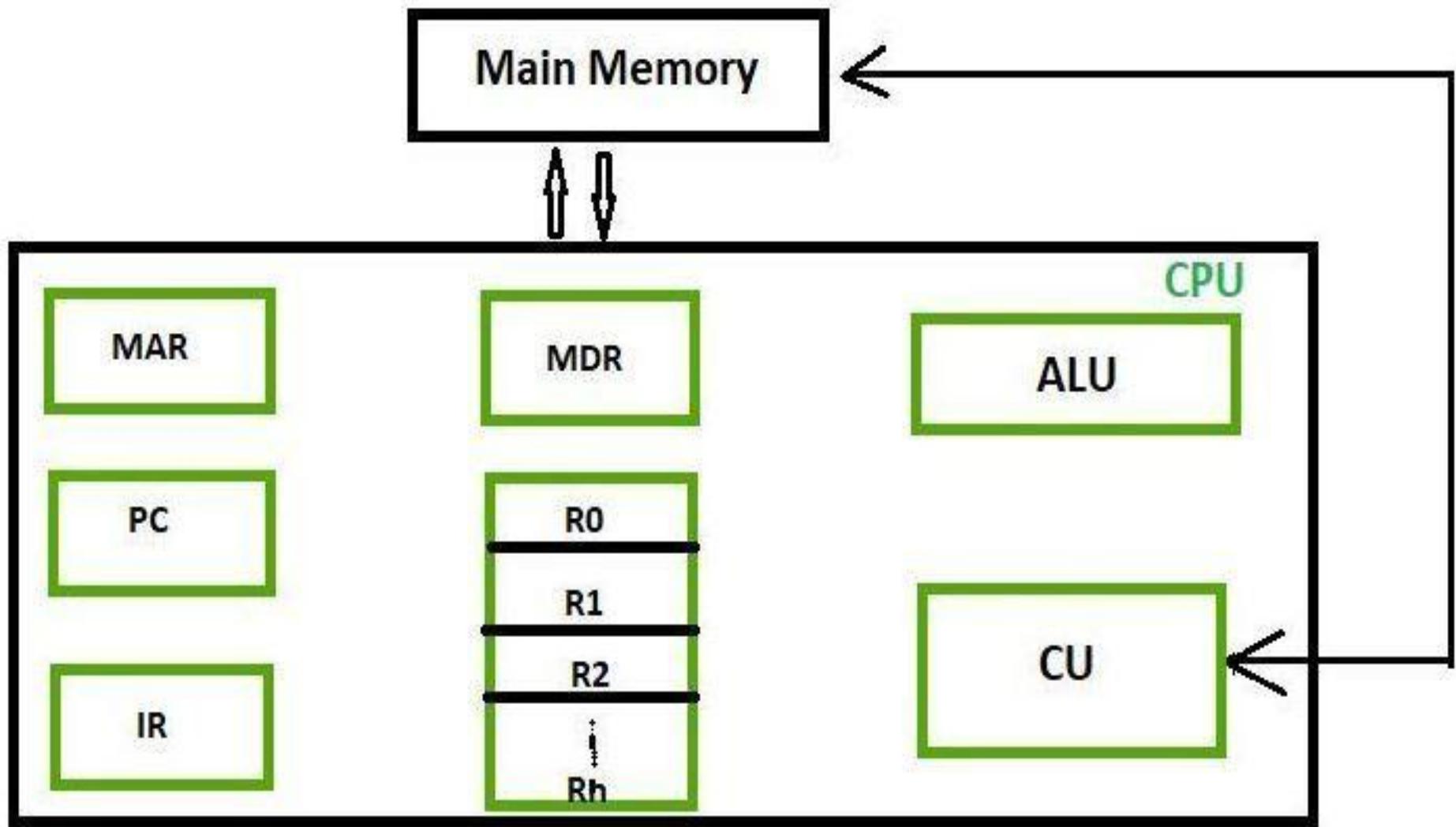
## MAR – Memory Address Register

It holds the address of the location to be accessed from memory.



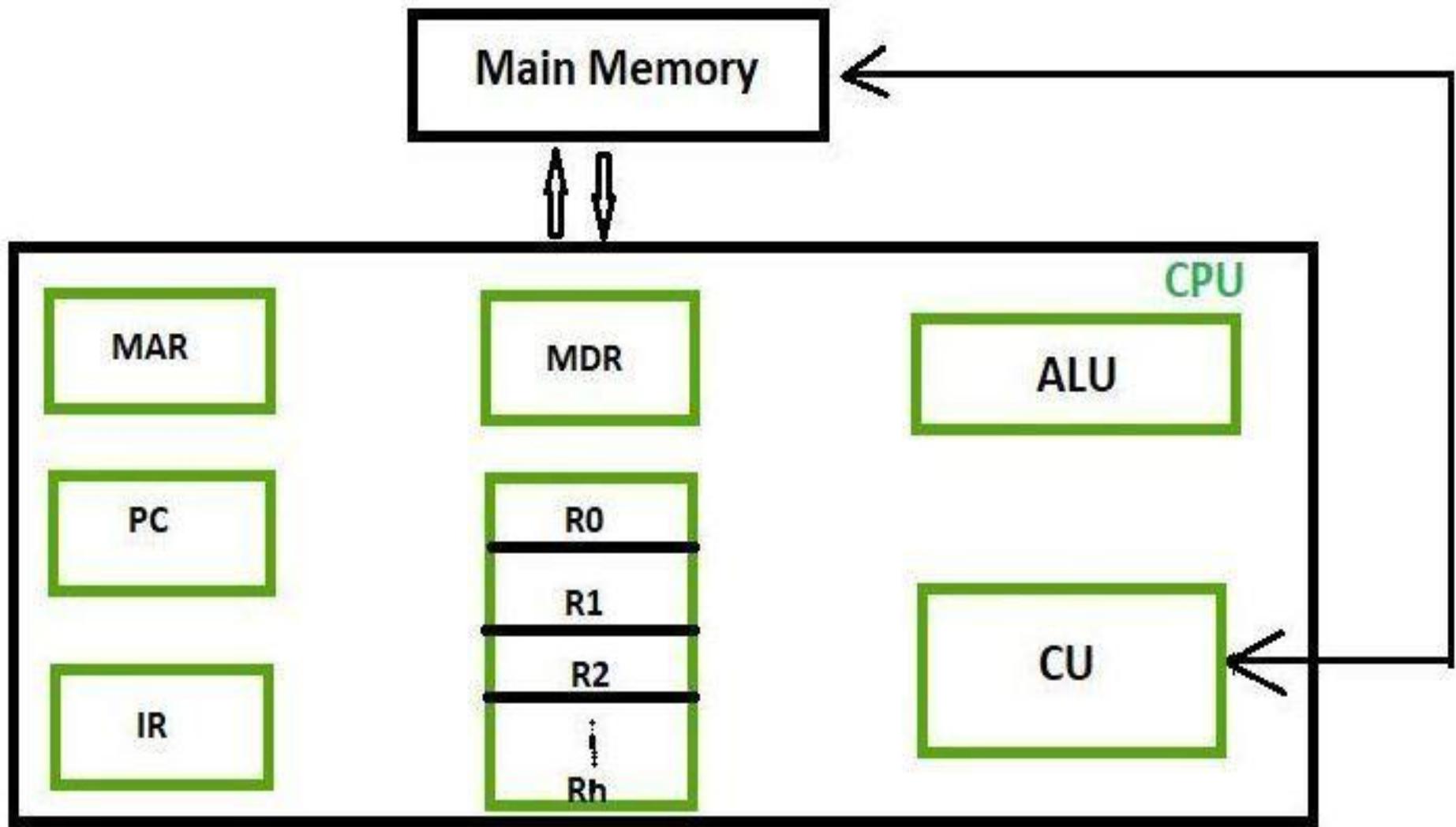
## MDR – Memory Data Register

It contains data to be written into or to be read out from the addressed location.



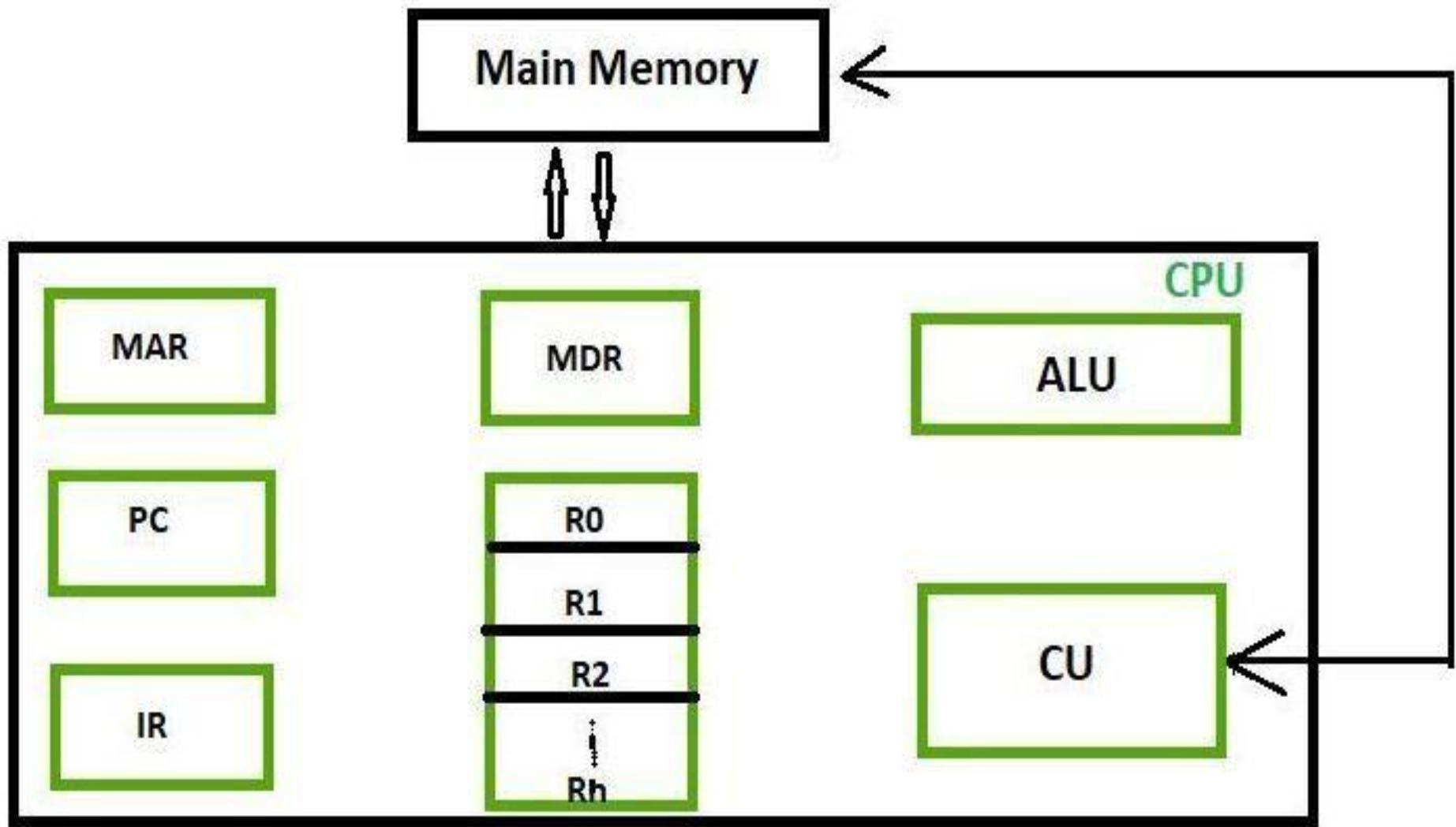
## General Purpose Registers

These are numbered as R<sub>0</sub>, R<sub>1</sub>, ..., R<sub>n</sub> and used to store temporary data.



## PC - Program Counter

It contains the memory address of next instruction being executed.



## IR – Instruction Register

It holds the instruction which is currently been executed.

# Cache Memory

- Extremely fast memory.
- Act as a buffer between RAM and CPU.
- It holds frequently requested data & instructions so that they are immediately available to the CPU when needed.

# Primary Memory

- Primary memory is faster in speed, less in size and costlier.
- It consists of :
  - 1) Read Only Memory(ROM)
  - 2) Random Access Memory (RAM)

<b>Properties</b>	<b>RAM</b>	<b>ROM</b>
<b>Data retention</b>	RAM is a <u>volatile</u> memory which could store the data as long as the power is supplied.	ROM is a <u>non-volatile</u> memory which could retain the data even when power is turned off.
<b>Working type</b>	Data stored in RAM can be <u>retrieved</u> and <u>altered</u> .	Data stored in ROM can <u>only be read</u> .
<b>Use</b>	Used to store the data that has to be currently processed by CPU temporarily.	It stores the <u>instructions</u> required during <u>bootstrap</u> of the computer.
<b>Speed</b>	It is a <u>high-speed</u> memory.	It is much slower than the RAM.
<b>CPU Interaction</b>	The CPU can access the data stored on it.	The CPU cannot access the data stored on it unless the data is stored in RAM.
<b>Size and Capacity</b>	Large size with higher capacity.	Small size with less capacity.
<b>Used as/in</b>	CPU Cache, Primary memory.	Firmware, Micro-controllers
<b>Accessibility</b>	The data stored is easily accessible	The data stored is not as easily accessible as in RAM
<b>Cost</b>	Costlier	Cheaper than RAM.

# Secondary Memory

- Anything stored in secondary memory remains available even if the computer is switched off.

# Secondary Storage Devices

- Magnetic Storage

- 1) Magnetic Tape ✓
- 2) Magnetic Hard Disk ✓
- 3) Floppy Disk ✓

- Optical Storage ✓

# Input Devices

- Input devices accept data and instructions from the user or from another computer system.
- The most common input devices are:
  - 1) Keyboard
  - 2) Mouse
  - 3) Scanner
  - 4) Joystick
  - 5) Microphone
  - 6) Optical Character Reader (OCR)

# Output Devices

- Output devices return processed data to the user or to another computer system.
- The most commonly used output devices are:
  - 1) Monitor
  - 2) Printer
  - 3) Speaker
  - 4) Plotters

# Bus

- Defined as a set of physical connections which can be shared by multiple hardware components in order to communicate with one another.
- Three fundamental buses are:
  - 1) Control Bus ✓
  - 2) Instruction Bus ✓
  - 3) Address Bus ✓

- 1) ***Control Bus*** - Used to transfer the control signals from one component to another. It is a bidirectional bus and can carry control signals in both directions.
- 2) ***Instruction Bus*** - Also known as data bus. It is a bidirectional bus and can carry the data in both the direction.
- 3) ***Address Bus*** - Used to transfer the address of memory location from one component to another. It is unidirectional bus.

# Factors affecting computer performance

- 1) The speed of the CPU
- 2) The size of the RAM (Random Access Memory)
- 3) The speed of the hard disk
- 4) Hard disk space
- 5) Multiple applications running on the computer
- 6) Type of graphic card
- 7) Defragmenting files

# Application Software

&

# System Software

# Software

- Software refers to the set of computer programs, procedures that describe the programs and how they are to be used.
- Computer software is normally classified into two broad categories:
  - 1) System Software
  - 2) Application Software

# System Software

- It is a general purpose software which is used to operate computer hardware.
- It provides platform to run application software.

*Example:* OS, Compiler, Assembler, Interpreter etc.

# Application Software

- It is a specific purpose software which is used by user for performing a specific task.
- Application software can't run independently.

*Example:* MS Word, Notepad, Windows Media Player etc.

<b>System Software</b>	<b>Application Software</b>
System Software maintain the <u>system resources</u> and give the path for application software to run.	Application software is built for specific tasks.
Low level <u>languages</u> are used to write the system software.	While high level <u>languages</u> are used to write the application software.
It's a <u>general-purpose</u> software.	While it's a <u>specific purpose</u> software.
Without system <u>software</u> , system can't run.	While without application software system always runs.
System software runs when system is turned on and stops <u>when system</u> is turned off.	While application software runs as per the <u>user's request</u> .
Example of system software are <u>operating systems</u>	Example of application software are <u>Photoshop, VLC player etc.</u>
System Software programming is complex than application software.	Application software programming is simpler as comparison to system software.

# Language Processors / Translators

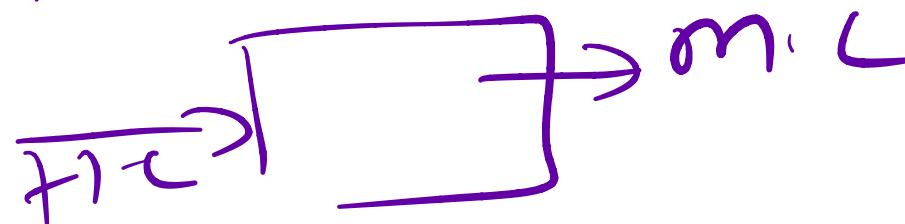
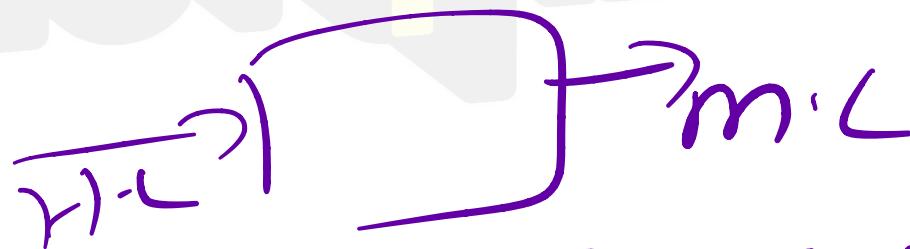
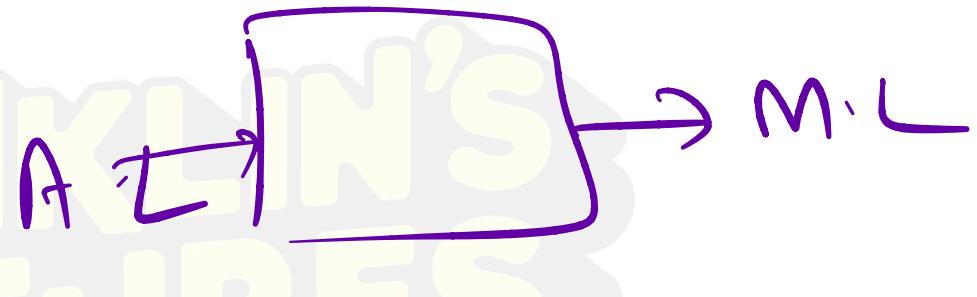
*A special translator system software is used to translate the program written in high level language to machine code is called language processors.*

# Language Processors / Translators

1) Compiler

2) Assembler

3) Interpreter



# Language Processors

## 1) Compiler



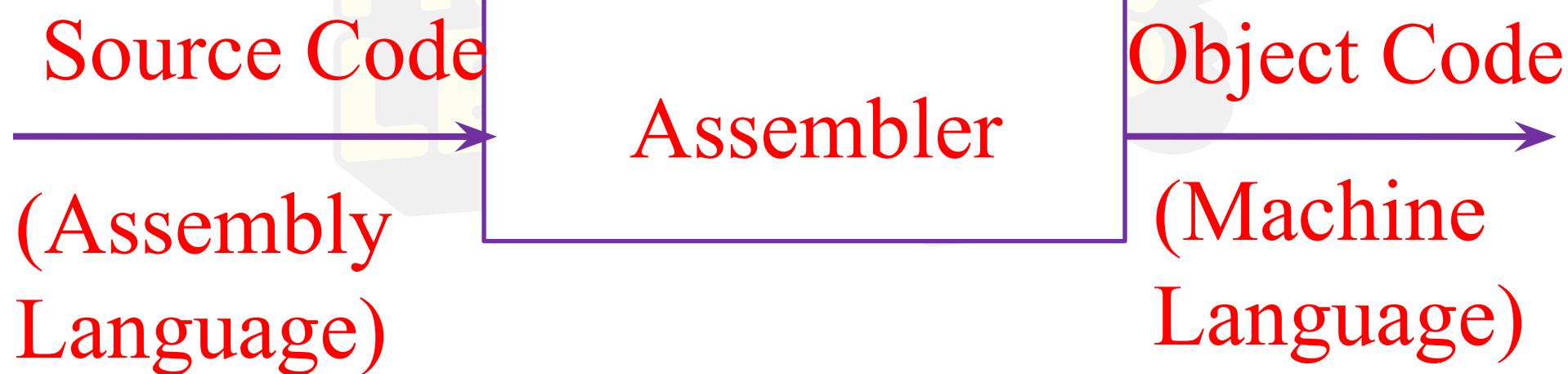
## 2) Assembler

## 3) Interpreter

# Language Processors

1) Compiler

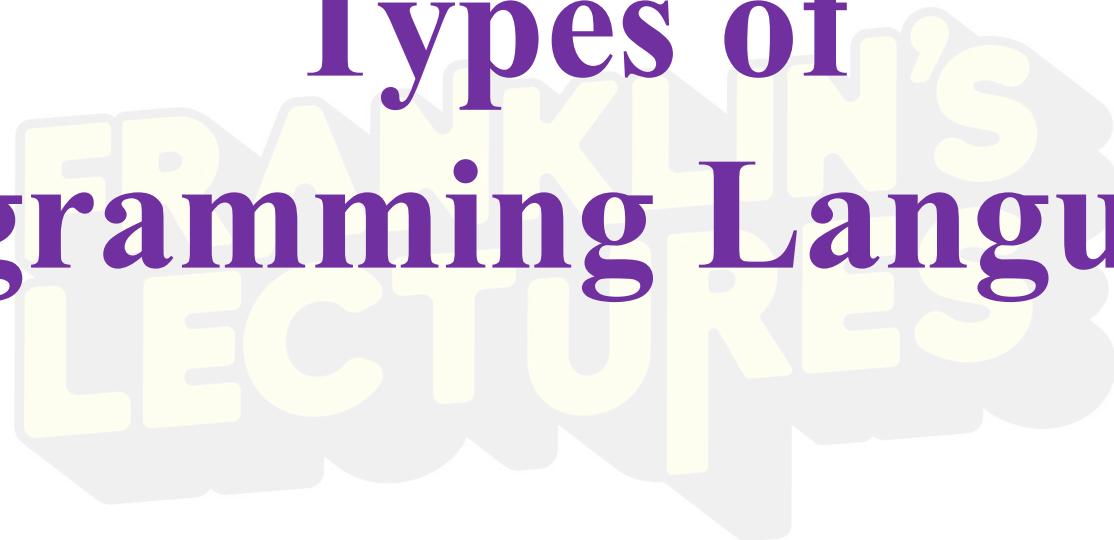
2) Assembler



3) Interpreter

<b>COMPILER</b>	<b>INTERPRETER</b>
Compiler looks at the entire source code.	Interpreter looks at a source code line-by-line
Compiler converts the entire source code into object-code and creates the object code. The object code is then executed by the user.	Interpreter converts a line into machine executable form, executes the line, and proceeds with the next line
For a given source code, once it is compiled, the object code is created. This object code can be executed multiple number of times by the user.	Interpreter executes line-by-line, so executing the program using an interpreter means that during each execution, the source code is first interpreted and then executed.
During execution of an object code, the compiler is not required	Both interpreter and the source code is required during execution
Faster interpretation compared to	Interpreter interprets line-by-line, the interpreted code runs slower than the compiled code.

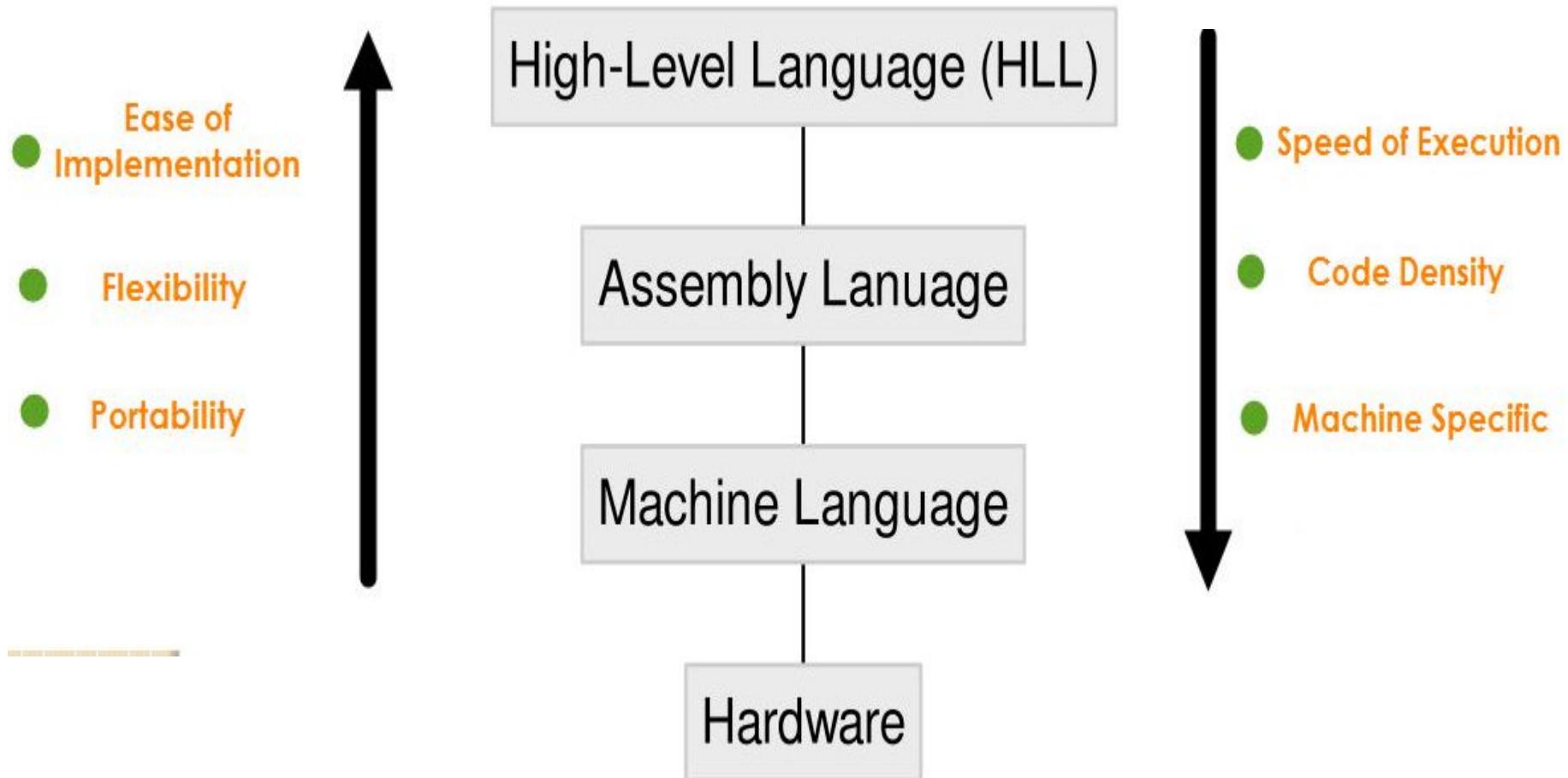
# Types of Programming Languages



# Programming Languages

- The operations of computer are controlled by a set of instructions called a computer program.
- The language used in the communication of computer instructions is known as the programming language.

# Programming Language Hierarchy



# Types of Programming Languages

- 1) Low level Language ✓
- 2) High level language ✓

# Types of Programming Languages

- 1) Low level Language
  - » Machine Language
  - » Assembly(Symbolic) Language
- 2) High level language

- **High Level Language**

High level language is written in English like language. Programs are easier to write, read or understand in high level language. Programs written in high level language should be converted into the object code using translator software like compiler or interpreter.

➤ *Advantages:*

1. Higher-level languages have a major advantage over machine and assembly languages that higher-level languages are easy to learn and use.
2. The programs can easily be debugged and are machine independent.

➤ *Disadvantages:*

1. Compared to low level languages, they are generally less memory efficient.
2. It takes additional translation time to translate the source to machine code.

Programming Language usually refers to high-level languages like COBOL, BASIC, FORTRAN, C, C++, Java etc.

- **Assembly Language**

A program written in assembly language uses symbolic representation of machine codes. Small English like representation [Mneumonics] is used to write the program in assembly language. Assembly languages are easier to write than machine language programs, since it uses English like representation of machine code.

Eg: ADD A, B

## Assembly Language

```
mov ecx, ebx  
mov esp, edx  
mov edx, r9d  
mov rax, rdx
```

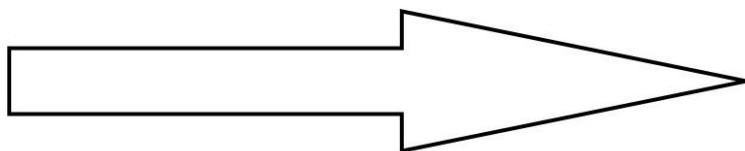
## Machine Language

```
100101011001  
010011111011  
111010101101  
01010101010
```

Assembler

Programmer

Processor



The program written in assembly language is the source code, which has to be converted into machine code called object code using translator software called assembler.

➤ *Advantages:*

1. Assembly Language is easier to understand and saves a lot of time and effort.
2. It is easier to correct errors and modify program instructions.

➤ *Disadvantages:*

1. Assembly language is machine dependent. A program written for one computer might not run in other computers with different hardware configuration.

● **Machine Language**

The lowest level of the computer language is machine language or machine code, which includes binary code. A program written in machine language is a collection of binary digits or bits that the computer reads and interprets. It is also referred to as machine code or object code. It is written as strings of 0's and 1's.

➤ *Advantages:*

The only advantage is that program of machine language run very fast because no translation is required.

➤ *Disadvantages:*

1. It is very difficult to program in machine language. The programmer has to know details of hardware to write program.

2. Machine language is hardware dependent.
3. The programmer has to remember a lot of codes to write a program, which results in program errors.
4. It is difficult to debug the program.

<b>High Level Language</b>	<b>Low Level Language</b>
It is <u>programmer/user friendly language</u> .	It is a <u>machine friendly</u> language.
High level language is less <u>memory efficient</u> .	Low level language is <u>high memory efficient</u> .
<u>It is easy to understand</u> .	<u>It is tough to understand</u> .
<u>It is simple to debug</u> .	<u>It is complex to debug comparatively</u> .
<u>It is simple to maintain</u> .	<u>It is complex to maintain comparatively</u> .
<u>It is portable</u> . ✓	<u>It is non-portable</u> . ✓
<u>It can run on any platform</u> .	<u>It is machine-dependent</u> .

# Introduction to Structured Approach to Programming

# Structured Programming

- Defined as a programming approach in which the program is made as a single structure.
- The code will execute the instruction one after another.

Structured programming is a programming paradigm aimed at improving the clarity, quality, and development time of a computer program by making extensive use of the structured control flow constructs of selection (if/then/else) and repetition (while and for), block structures, and subroutines.

Structured programming can be performed in two ways:

- Procedural Programming
- Modular Programming

### ***Procedural Programming***

It requires a given task to be divided into smaller procedures, functions or subroutines. A procedural program is largely a single file consisting of many procedures and functions and a function named main(). A procedure or function performs a specific task. The function main() integrates the procedures and functions by making calls to them, in an order that implements the functionality of the program. When a procedure or function is called, the execution control jumps to the called procedure or function, the procedure or function is executed, and after execution the control comes back to the calling procedure or function.

### ***Modular Programming***

It requires breaking down of a program into a group of files, where each file consists of a program that can be executed independently. In a modular program, the problem is divided into different independent but related tasks. For each identified task, a separate program (module) is written, which is a program file that can be

Modular Programming is of two types:

i. **Top down modular programming**: In this approach, a large project divides into small programs, and these programs are known as modules. C programming language supports this approach for developing projects. It is always good idea that decomposing solution into modules in a hierachal manner. The basic task of a top-down approach is to divide the problem into tasks and then divide tasks into smaller sub-tasks and so on. In this approach, first we develop the main module and then the next level modules are developed. This procedure is continued until all the modules are developed.

ii. **Bottom up modular programming**: It is an alternative approach to the top-down approach. In this approach, bottom level modules developed first (Lower level module developed, tested and debugged). Then the next module developed, tested and debugged. This process is continued until all modules have been completed. This approach is exactly opposite to the top-down approach. This approach is good for reusability of code. C++ used the bottom-up approach.

# Algorithms

- An algorithm is a set of steps for solving a particular problem.
- It is a precise specification of a finite sequence of instructions to be carried out in order to solve a given problem.
- It can written in any natural language like English.

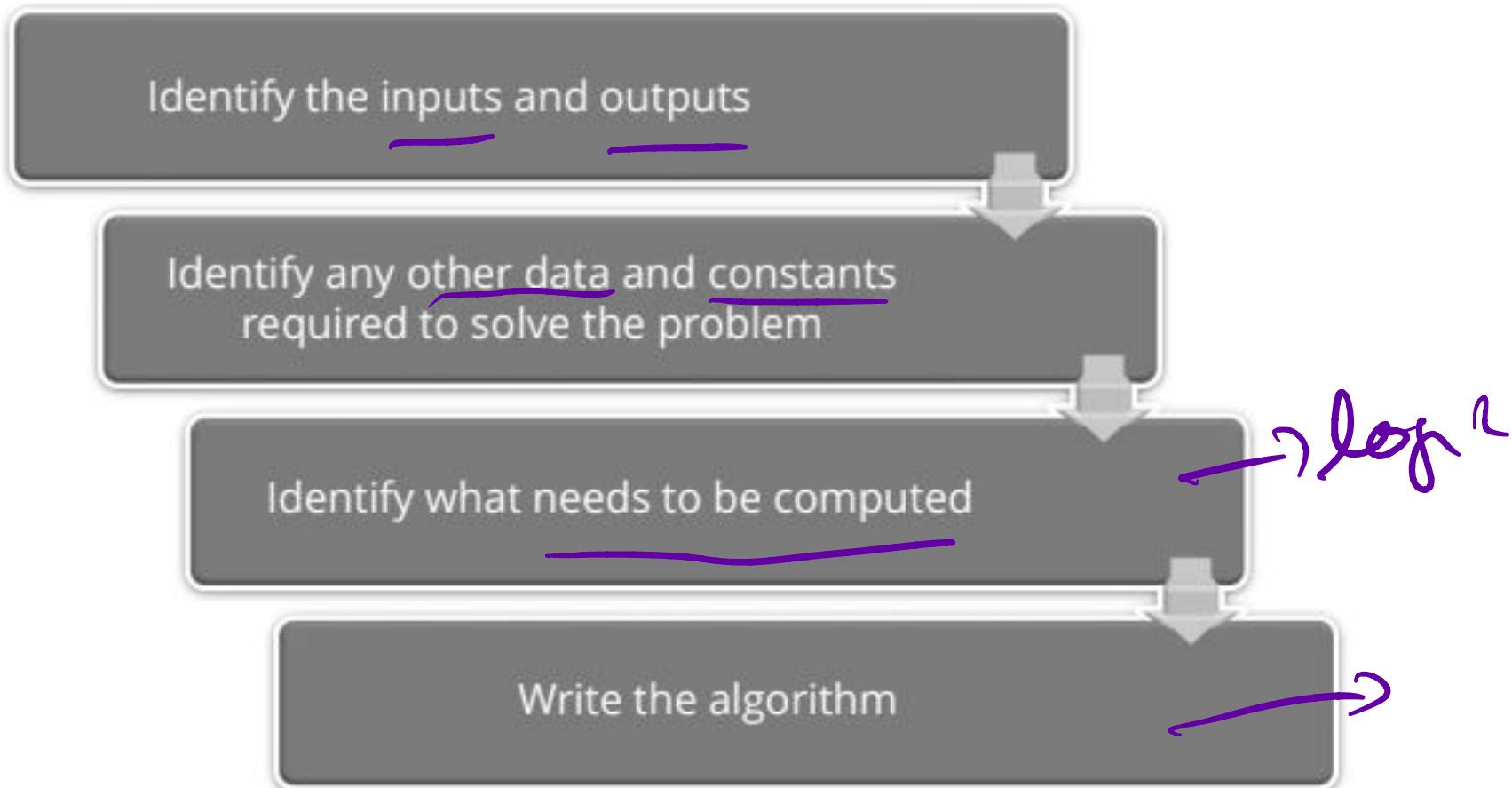
# Properties of good Algorithms

- 1) Precision ✓
- 2) Uniqueness ✓
- 3) Finiteness ✓
- 4) Input ✓
- 5) Output ✓
- 6) Generality ✓

# Properties of good Algorithms

- 1) **Precision** – *Steps are precisely stated.*
- 2) **Uniqueness** – *Results of each step are uniquely defined and depend on the input and the result of preceding steps.*
- 3) **Finiteness** – *Algorithm stops after a finite number of instructions are executed.*
- 4) **Input** – *Algorithm receives input.*
- 5) **Output** – *Algorithm produces output.*
- 6) **Generality** – *Algorithm applies to a set of inputs.*

# Using Algorithm to solve Problems



# Common words used in an Algorithm

- Input : Read, Obtain, Get
- Output : Print, Write
- Compute : Compute, Calculate,  
Determine
- Initialize : Set, Initialize
- Add one : Increment
- Subtract one : Decrement

# Example

Write an algorithm to find the average of three numbers.

Problem Statement: Find the average of three numbers

- *Identify the inputs and outputs*

*Input* : Three numbers

*num1, num2, num3*

*Output* : Average of three numbers

- Identify any other data and constants required to solve the problem

The average of numbers is defined as,

*Sum of all inputs*

*Total number of inputs*

Here, other data to be computed is  
*Sum of all inputs, ie,*

$$\text{Sum} = \text{num1} + \text{num2} + \text{num3}$$

- *Identify what needs to be computed*

The average of three numbers has to be computed.

$$\text{Average} = \frac{\text{Sum}}{3}$$

- *Write the algorithm*

*Step 1 : Start* ✓

*Step 2 : Read three numbers,*

*num1,num2,num3*

*Step 3 : Calculate the sum,*

$$\text{Sum} = \underline{\text{num1}} + \underline{\text{num2}} + \underline{\text{num3}}$$

*Step 4 : Calculate the average,*

$$\text{Average} = \text{Sum}/3$$

*Step 5 : Print Average*

*Step 6 : Stop* ✓

*which can also be written as,*

*Step 1 : Start ✓*

*Step 2 : Read num1,num2,num3*

*Step 3 : Sum = num1 + num2 + num3*

*Step 4 : Average = Sum/3*

*Step 5 : Print Average*

*Step 6 : Stop*

## Question No. 1

Write an algorithm to find area of a circle.

Problem Statement : Find area of a circle



*Step 1 : Start ✓*

*Step 2 : Read radius, r*

*Step 3 : Calculate the area,*

$$Area = \underline{3.14 * r * r}$$

*Step 4 : Print Area*

*Step 5 : Stop ✓*

*which can also be written as,*

*Step 1 : Start*

*Step 2 : Read radius,  $r$*

*Step 3 :  $\text{Area} = 3.14 * r * r$*

*Step 4 : Print  $\text{Area}$*

*Step 5 : Stop*

## Question No. 2

Write an algorithm to find area of a square.

---

Problem Statement : Find area of a square

## Write the algorithm

*Step 1 : Start* ✓

*Step 2 : Read length of side,  $a$*

*Step 3 :  $Area = a * a$*  —

*Step 4 : Print  $Area$*

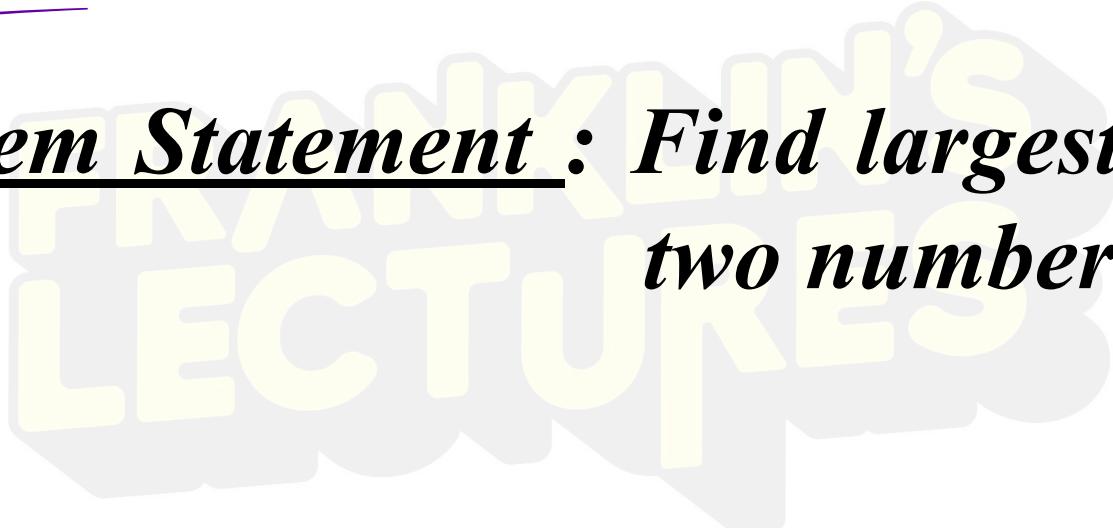
*Step 5 : Stop* ✓

### Question No. 3

Write an algorithm to find largest among two numbers.

---

Problem Statement: Find largest among two numbers

A faint watermark logo is centered in the background. It features the text "TANAKA'S" stacked above "LECTURES". Both words are written in a stylized, blocky font where each letter is composed of several smaller squares. The entire logo is a light gray color.

- Write the algorithm

Step 1 : Start ✓

Step 2 : Read num1, num2

Step 3 : If num1 > num2, then, go to  
Step 4, otherwise go to Step 5.

Step 4 : Largest = num1 go to Step 6

Step 5 : Largest = num2

Step 6 : Print Largest

Step 7 : Stop ✓

## Question No. 4

Write an algorithm to check whether the given number is odd or even.

Problem Statement: Check whether the given number is odd or even

- *Write the algorithm*

*Step 1 : Start* ✓

*Step 2 : Read number, n* ✓

*Step 3 : Remainder = n % 2*

*Step 4 : If Remainder == 0, then, go to  
Step 5, otherwise go to Step 6.*

*Step 5 : Print “Even Number” go to  
Step 7.*

*Step 6 : Print “Odd Number”*

*Step 7 : Stop* ✓

remainder  
always  
done

# Flow Charts

- A Flow Chart depicts pictorially the sequence in which instructions are carried out in an algorithm.

# Symbols used in Flow Charts

## Symbol



Rectangle with rounded ends

## Use

Start or end of the program or flowchart



Rectangles

Computational steps or processing function of a program

$\log_2$

# Symbols used in Flow Charts(Contd...)

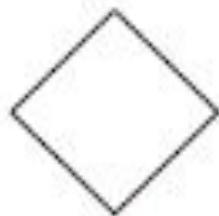
## Symbol

## Use



Parallelograms

Input entry or output display operation



Diamond shaped boxes

A decision making and branching operation that has two alternatives



Used to join different parts of a flow chart.

# Symbols used in Flow Charts(Contd...)

## Symbol

## Use

Connector



Indicate the direction to be followed in a flow chart. Every line in a flow chart must have an arrow on it.

Flow Lines



For representing Loop in a flow chart

Loop



For representing subroutine in a flow chart

Subroutine

# Example

Give the algorithm and draw flow chart to find the average of three numbers.

Problem Statement : Find the average of three numbers

## Algorithm

Step 1 : Start ✓

Step 2 : Read num1,num2,num3

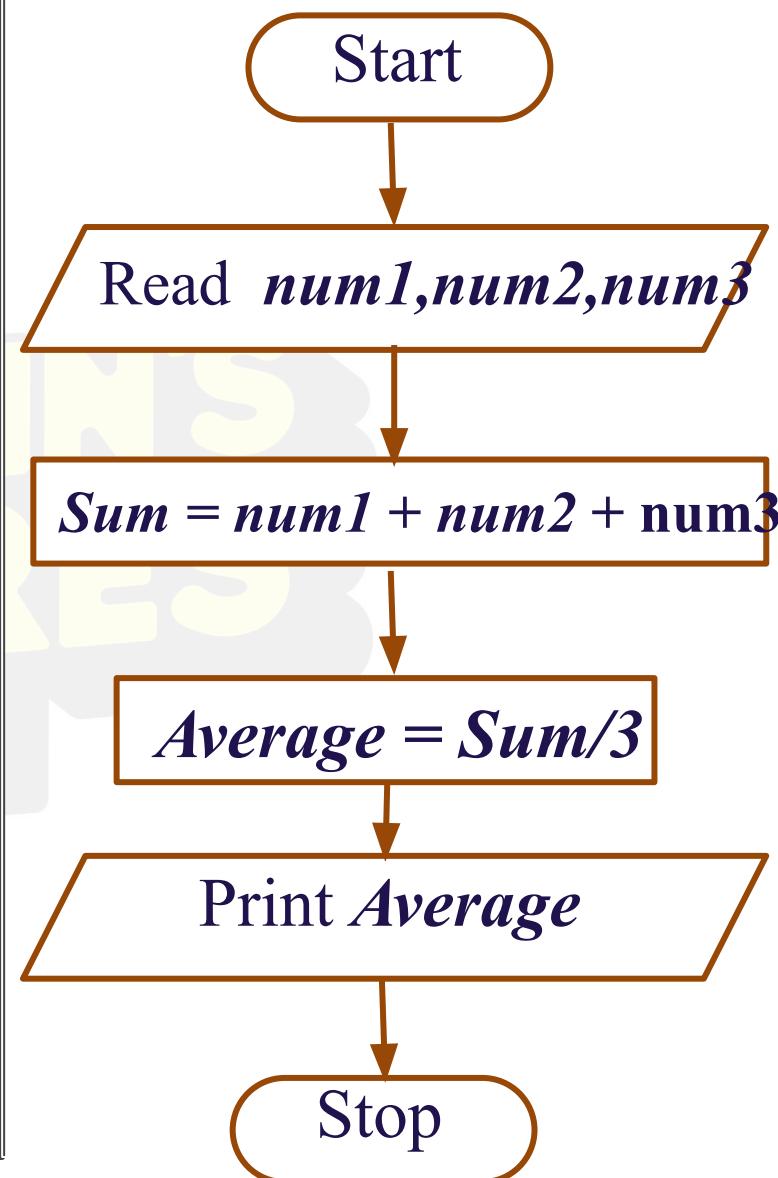
Step 3 : Sum = num1 + num2 + num3

Step 4 : Average = Sum/3

Step 5 : Print Average

Step 6 : Stop ✓

## Flow Chart



## Question No. 1

Design an algorithm and flow chart to find area of a circle.

---

Problem Statement: Find area of a circle

## Algorithm

*Step 1* : Start

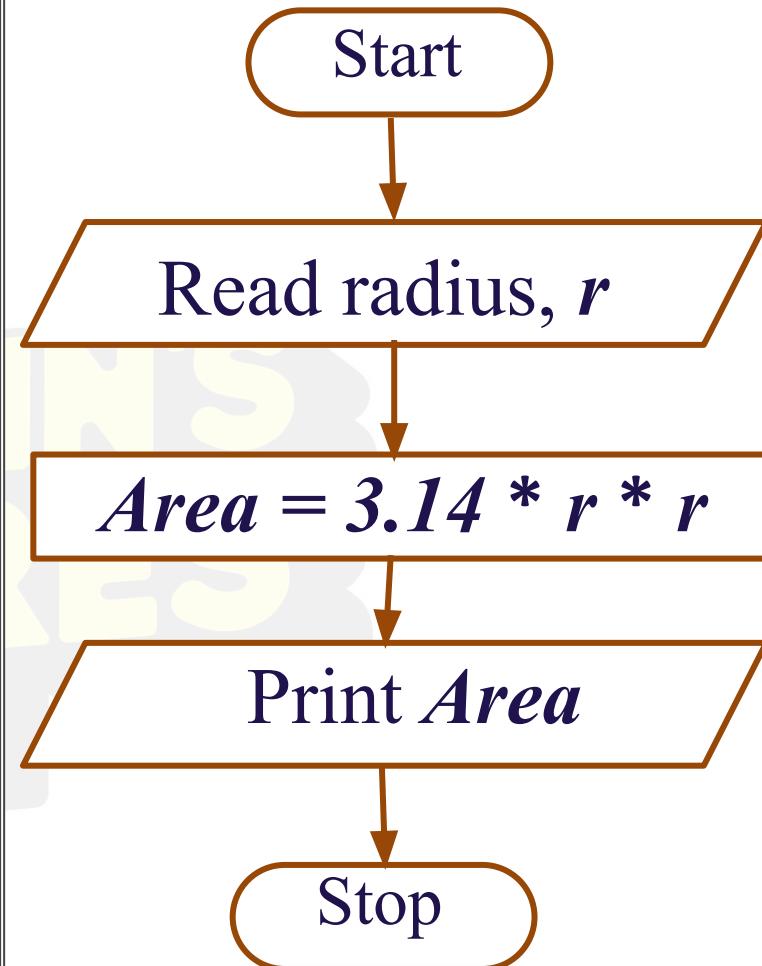
*Step 2* : Read radius,  $r$

*Step 3* :  $Area = 3.14 * r * r$

*Step 4* : Print  $Area$

*Step 5* : Stop

## Flow Chart



## Question No. 2

Write an algorithm and draw flow chart to find area of a square.

Problem Statement: Find area of a square

## Algorithm

*Step 1 : Start*

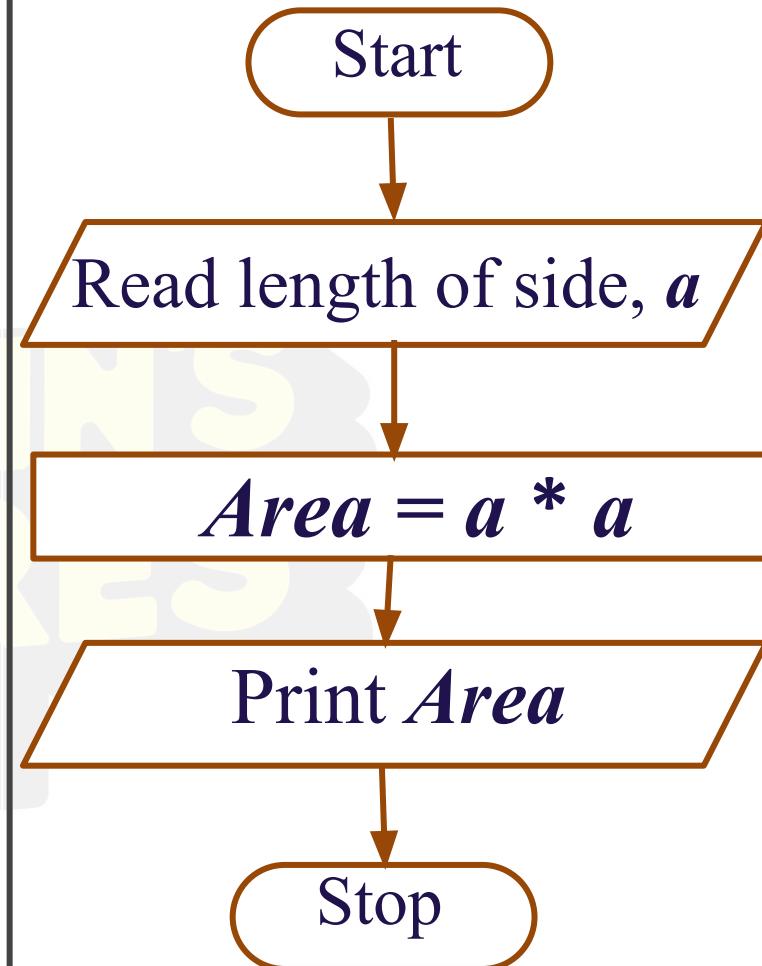
*Step 2 : Read length of side,  $\underline{a}$*

*Step 3 :  $Area = a * a$*

*Step 4 : Print  $Area$*

*Step 5 : Stop*

## Flow Chart



## Question No. 3

Write an algorithm and draw the flow chart to find largest among two numbers.

Problem Statement: Find largest among two numbers

# Algorithm

*Step 1* : Start

*Step 2* : Read *num1, num2*

*Step 3* : If  $\underline{\textit{num1} > \textit{num2}}$ , then, go to *Step 4*,  
otherwise go to *Step 5*.

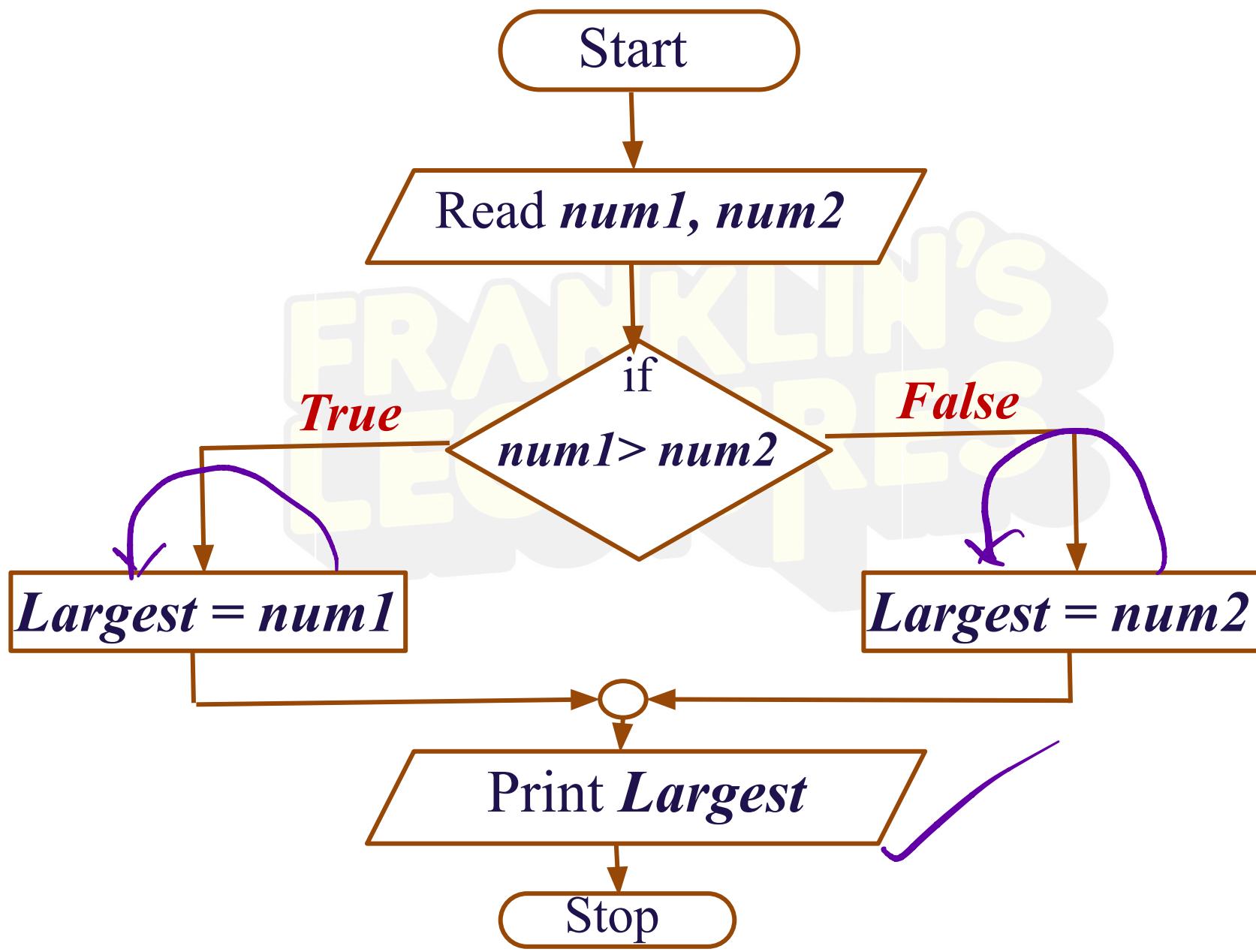
*Step 4* : *Largest* = *num1* go to *Step 6*

*Step 5* : *Largest* = *num2*

*Step 6* : Print *Largest*

*Step 7* : Stop

# Flow Chart



## Question No. 4

Write an algorithm and draw the flow chart to check whether the given number is odd or even.

Problem Statement : Check whether the given number is odd or even

## Algorithm

*Step 1 : Start*

*Step 2 : Read number,  $n$*

*Step 3 :  $Remainder = n \% 2$*

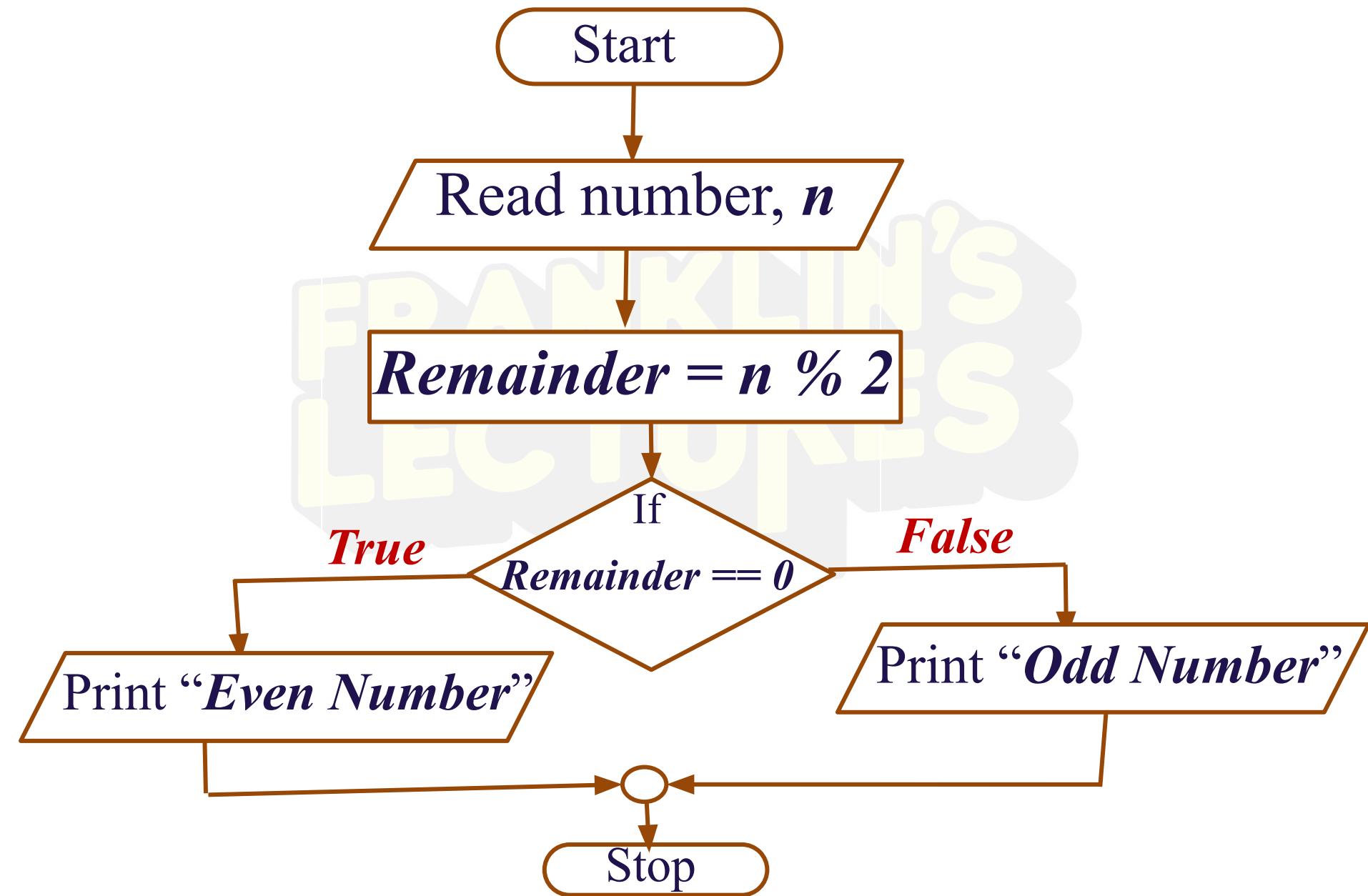
*Step 4 : If  $Remainder == 0$ , then, go to Step 5,  
otherwise go to Step 6.*

*Step 5 : Print “Even Number” go to Step 7.*

*Step 6 : Print “Odd Number”*

*Step 7 : Stop*

# Flow Chart



# Pseudocode

- Implementation of an algorithm in the form of annotations and informative text written in plain English.
- It has no syntax.
- Can't be compiled or interpreted by the computer.

# Pseudocode (*Contd...*)

- Pseudocode is a simple way of writing programming code in English.
- It uses short phrases to write code for programs before you actually create it in a specific language.
- Pseudocode makes creating programs easier.

# Why use pseudocode at all?

- 1) Better readability
- 2) Ease up code construction
- 3) A good middle point between flow chart and code
- 4) Act as start point for documentation
- 5) Easier bug detection and fixing

# The main constructs of pseudocode

- The core of pseudocode is the ability to represent *six programming constructs*, always written in uppercase.
- These constructs, also called *keywords*, are used to describe the control flow of the algorithm.

# The main constructs of pseudocode

- 1) SEQUENCE
- 2) WHILE
- 3) REPEAT - UNTIL
- 4) FOR
- 5) IF - THEN - ELSE
- 6) CASE

# The main constructs of pseudocode

- 1) **SEQUENCE** *represents linear task sequentially performed one after the other.*
- 2) **WHILE** *a loop with a condition at its beginning.*
- 3) **REPEAT - UNTIL** *a loop with a condition at the bottom.*
- 4) **FOR** *another way of looping.*
- 5) **IF - THEN - ELSE** *a conditional statement changing the flow of the algorithm.*
- 6) **CASE** *the generalization form of IF - THEN - ELSE.*

## SEQUENCE

**Input:** READ, OBTAIN, GET

**Output:** PRINT, DISPLAY, SHOW

**Compute:** COMPUTE,  
CALCULATE, DETERMINE

**Initialize:** SET, INIT

**Add:** INCREMENT, BUMP

**Sub:** DECREMENT

## FOR

**FOR** iteration bounds

sequence

**ENDFOR**

## IF-THEN-ELSE

**IF** condition **THEN**

sequence 1

**ELSE**

sequence 2

**ENDIF**

## WHILE

**WHILE** condition

sequence

**ENDWHILE**

## REPEAT-UNTIL

**REPEAT**

sequence

**UNTIL** condition

## CASE

**CASE** expression **OF**

condition 1: sequence 1

condition 2: sequence 2

...

condition n: sequence n

**OTHERS:**

default sequence

**ENDCASE**

# Example

Give the algorithm, draw flow chart and pseudocode to find the average of three numbers.

Problem Statement: Find the average of three numbers

## Algorithm

*Step 1 : Start*

*Step 2 : Read num1,num2,num3*

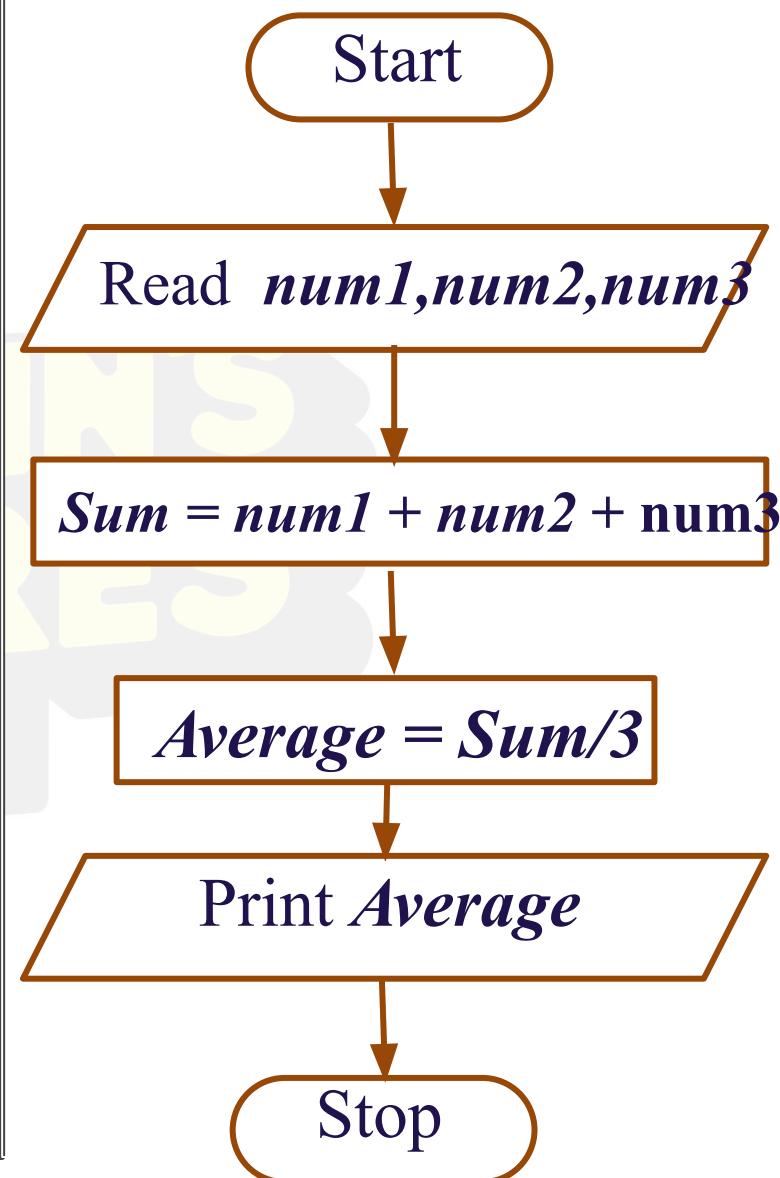
*Step 3 : Sum = num1 + num2 + num3*

*Step 4 : Average = Sum/3*

*Step 5 : Print Average*

*Step 6 : Stop*

## Flow Chart



## Pseudocode

READ  $num1, num2, num3$

$Sum = num1 + num2 + num3$

$Average = Sum/3$

PRINT  $Average$

## Question No. 1

Design an algorithm, draw flow chart and pseudocode to find area of a circle.

Problem Statement: Find area of a circle

## Algorithm

*Step 1 : Start*

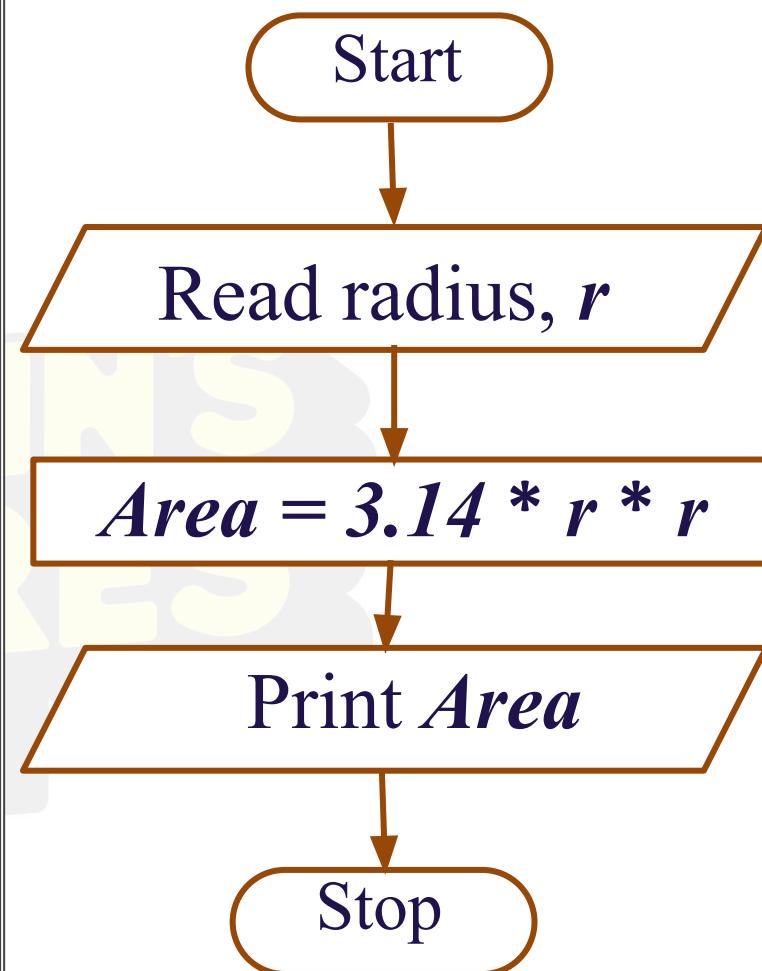
*Step 2 : Read radius,  $r$*

*Step 3 :  $Area = 3.14 * r * r$*

*Step 4 : Print  $Area$*

*Step 5 : Stop*

## Flow Chart



# Pseudocode

READ  $r$

$Area = 3.14 * r * r$

PRINT  $Area$



## Question No. 2

Write an algorithm, draw flow chart and pseudocode to find area of a square.

---

Problem Statement: Find area of a square

## Algorithm

*Step 1* : Start

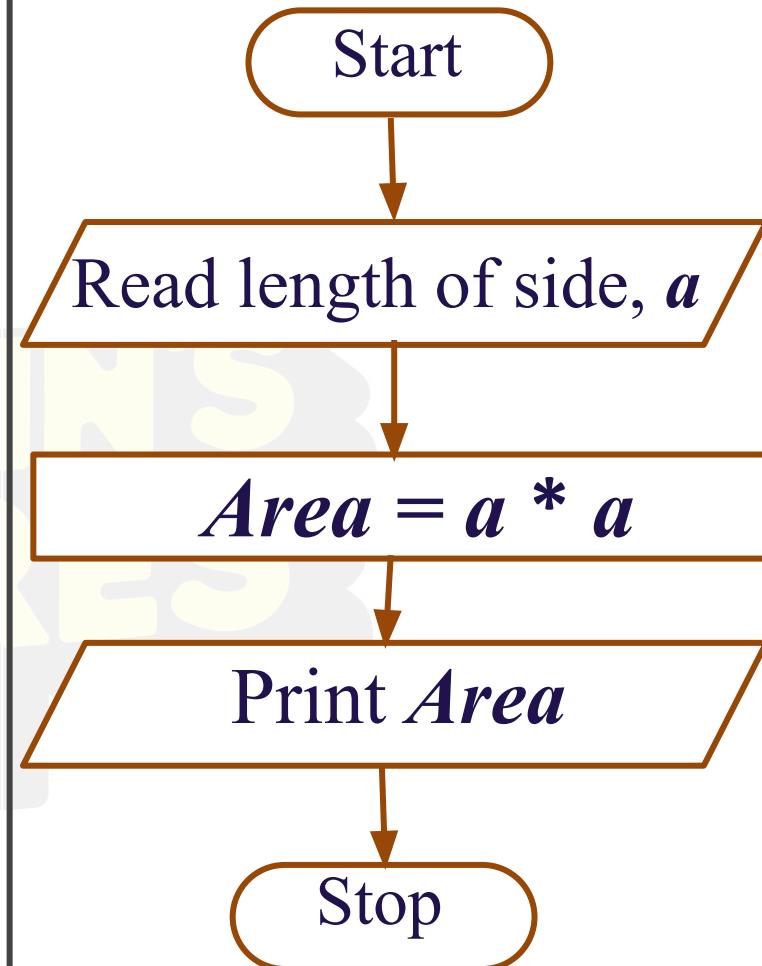
*Step 2* : Read length of side,  $a$

*Step 3* :  $Area = a * a$

*Step 4* : Print  $Area$

*Step 5* : Stop

## Flow Chart



# Pseudocode

READ  $a$

$$Area = a * a$$

PRINT  $Area$



## Question No. 3

Write an algorithm, draw flow chart and pseudocode to find largest among two numbers.

Problem Statement : Find largest among two numbers

# Algorithm

*Step 1* : Start

*Step 2* : Read *num1, num2*

*Step 3* : If *num1 > num2*, then, go to *Step 4*,  
otherwise go to *Step 5*.

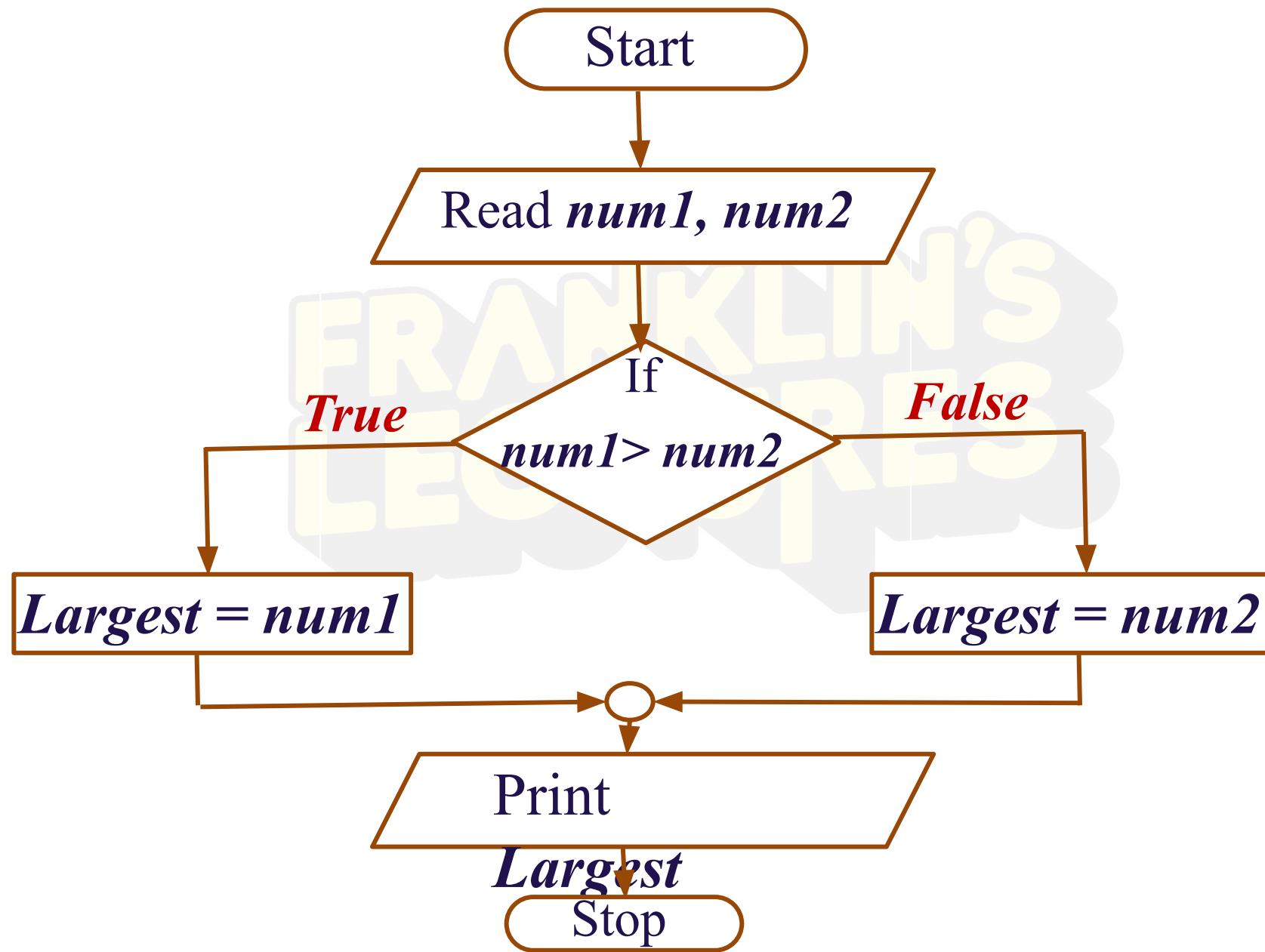
*Step 4* : *Largest = num1* go to *Step 6*

*Step 5* : *Largest = num2*

*Step 6* : Print *Largest*

*Step 7* : Stop

# Flow Chart



# Pseudocode

READ *num1, num2*

IF *num1 > num2*

THEN

*Largest = num1*

ELSE

*Largest = num2*

ENDIF

PRINT *Largest*

## Question No. 4

Write an algorithm, draw flow chart and pseudocode to check whether the given number is odd or even.

Problem Statement : Check whether the given number is odd or even

## Algorithm

*Step 1* : Start

*Step 2* : Read number,  $n$

*Step 3* :  $\text{Remainder} = n \% 2$

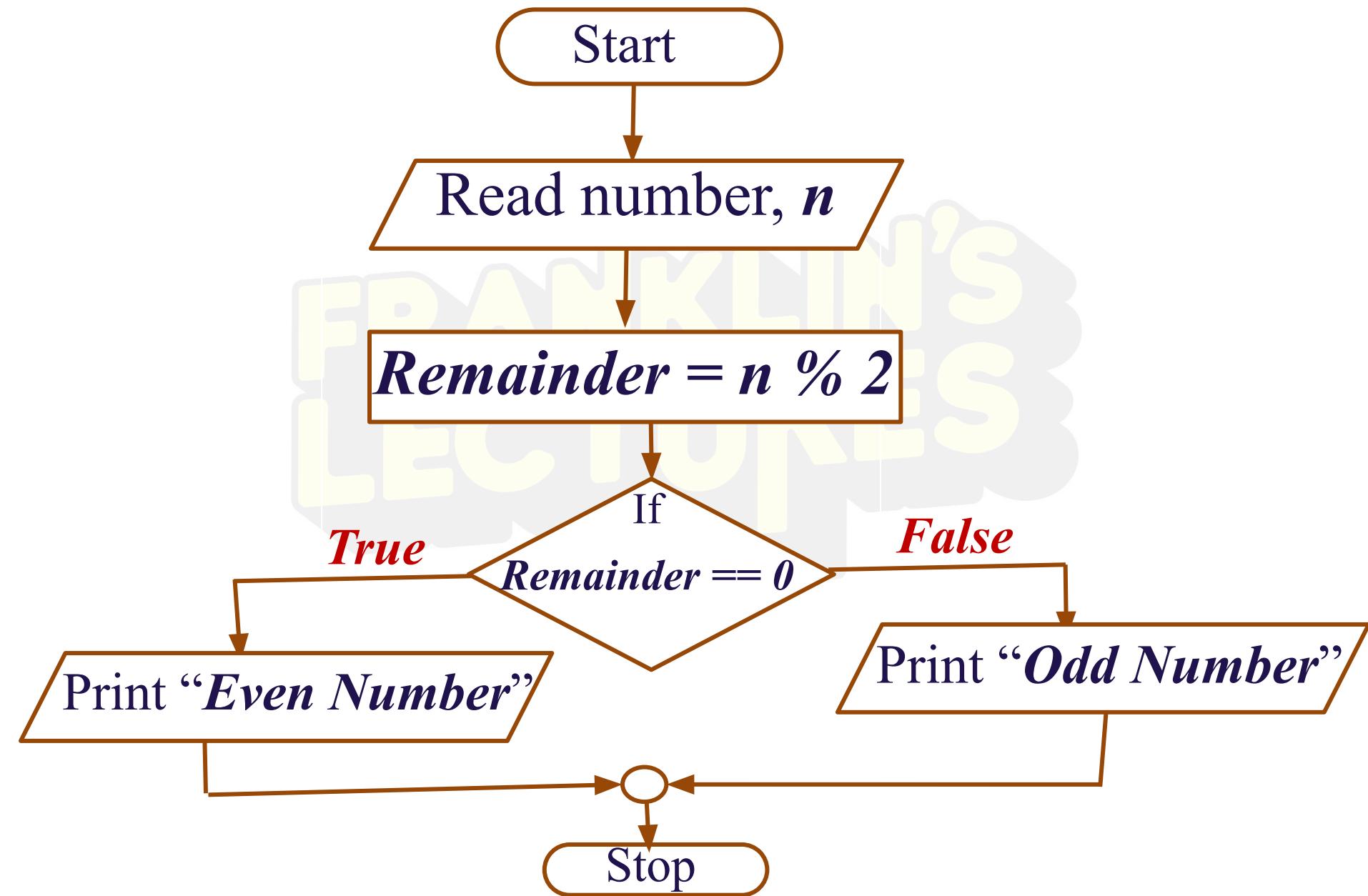
*Step 4* : If  $\text{Remainder} == 0$ , then, go to *Step 5*,  
otherwise go to *Step 6*.

*Step 5* : Print “*Even Number*” go to *Step 7*.

*Step 6* : Print “*Odd Number*”

*Step 7* : Stop

# Flow Chart



# Pseudocode

READ *n*

*Remainder* = *n % 2*

IF *Remainder* == 0

THEN

    PRINT “*Even Number*”

ELSE

    PRINT “*Odd Number*”

ENDIF

# Difference b/w Algorithm, Flow Chart and pseudo Code

S. No	Algorithm	Flow Chart	Pseudo Code
1	Sequence of instruction to solve the Particular Problem	Pictorial Representation of algorithm	Sequence of instruction to solve the Particular Problem
2	It's pure English Language, it has no rules to write	It has symbol to Represent the instruction	It's a English language But it has, some set of rules
3	It's not a tool for document purpose	Flow chart are tools document and represent algorithm	Pseudo code are tools document represent and Algorithm
4	Sequence of instruction to solve the Particular Problem	Pictorial representation of algorithm using standard symbols.	Rules of structured design & programming.

# Algorithm to find the factorial of a number

Step 1: Start ✓

Step 2: Read the number, n

Step 3: Initialize i=1 and fact=1

Step 4: Calculate fact=fact\*i

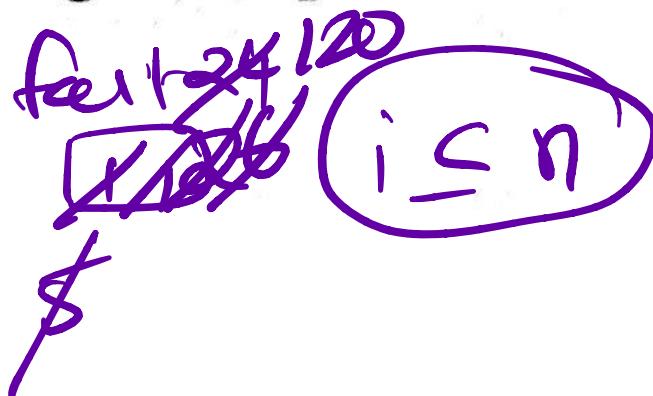
Step 5: Set i=i+1

Step 6: Check whether  $i \leq n$ , If yes goto step 4 otherwise goto  
step 7

Step 7: Print the factorial, fact

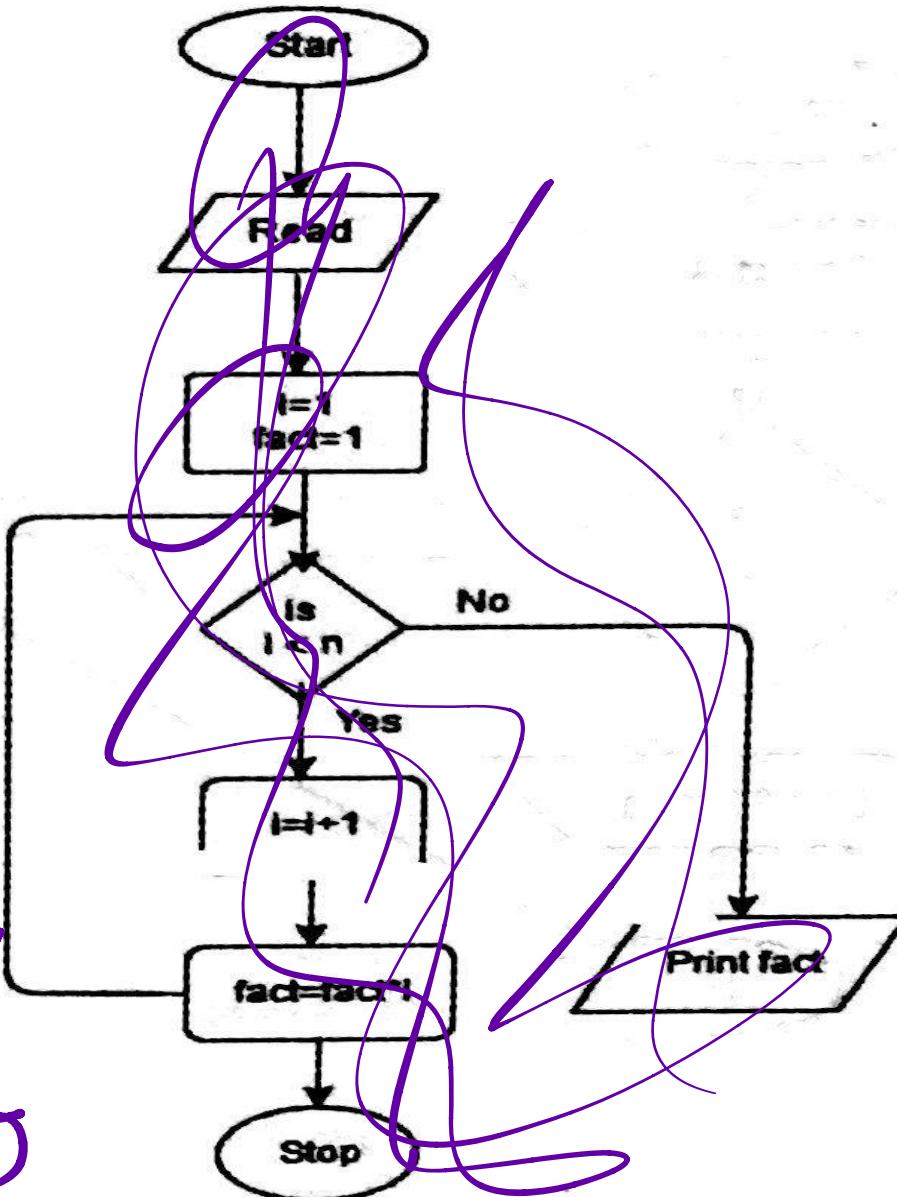
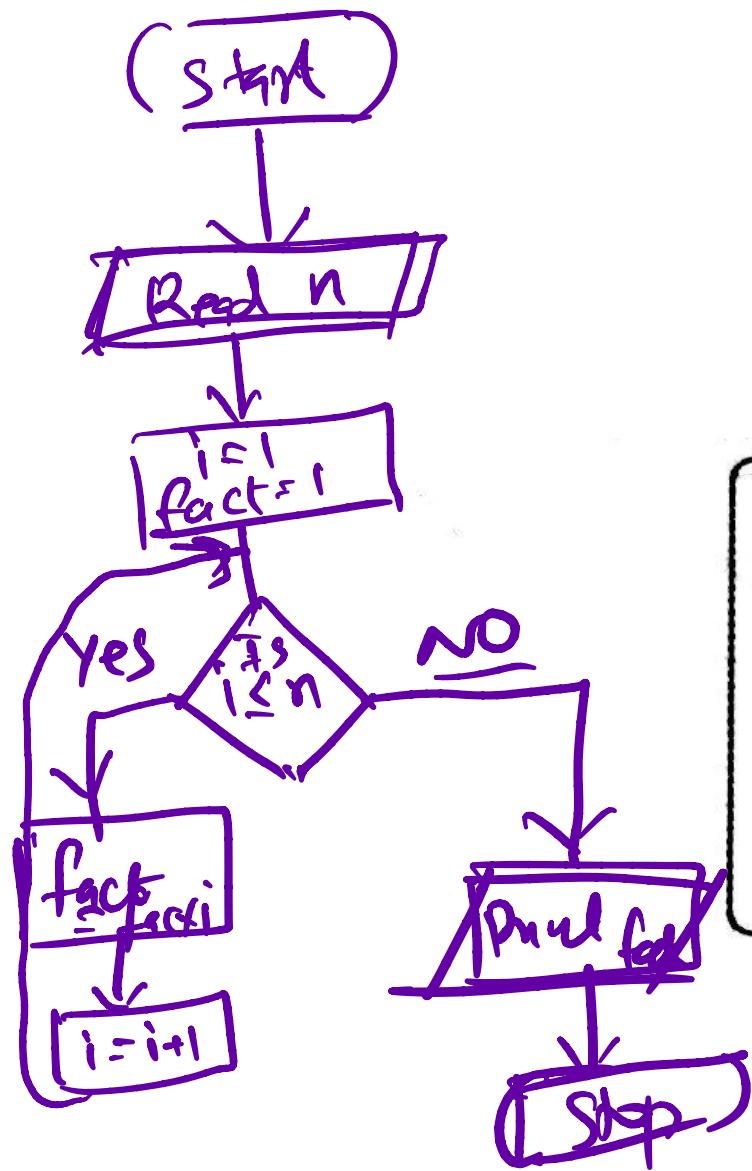
Step 8: Stop

1 2 3 4



$$5! = \\ 5 \times 4 \times 3 \times 2 \times 1$$

# Flowchart to find the factorial of a number



# Algorithm for sum of n natural numbers

Step 1 : START

Step 2 : Read limit of numbers , n

Step 3 : Assign sum=0 and i=0

Step 4 : Repeat until  $i \leq n$

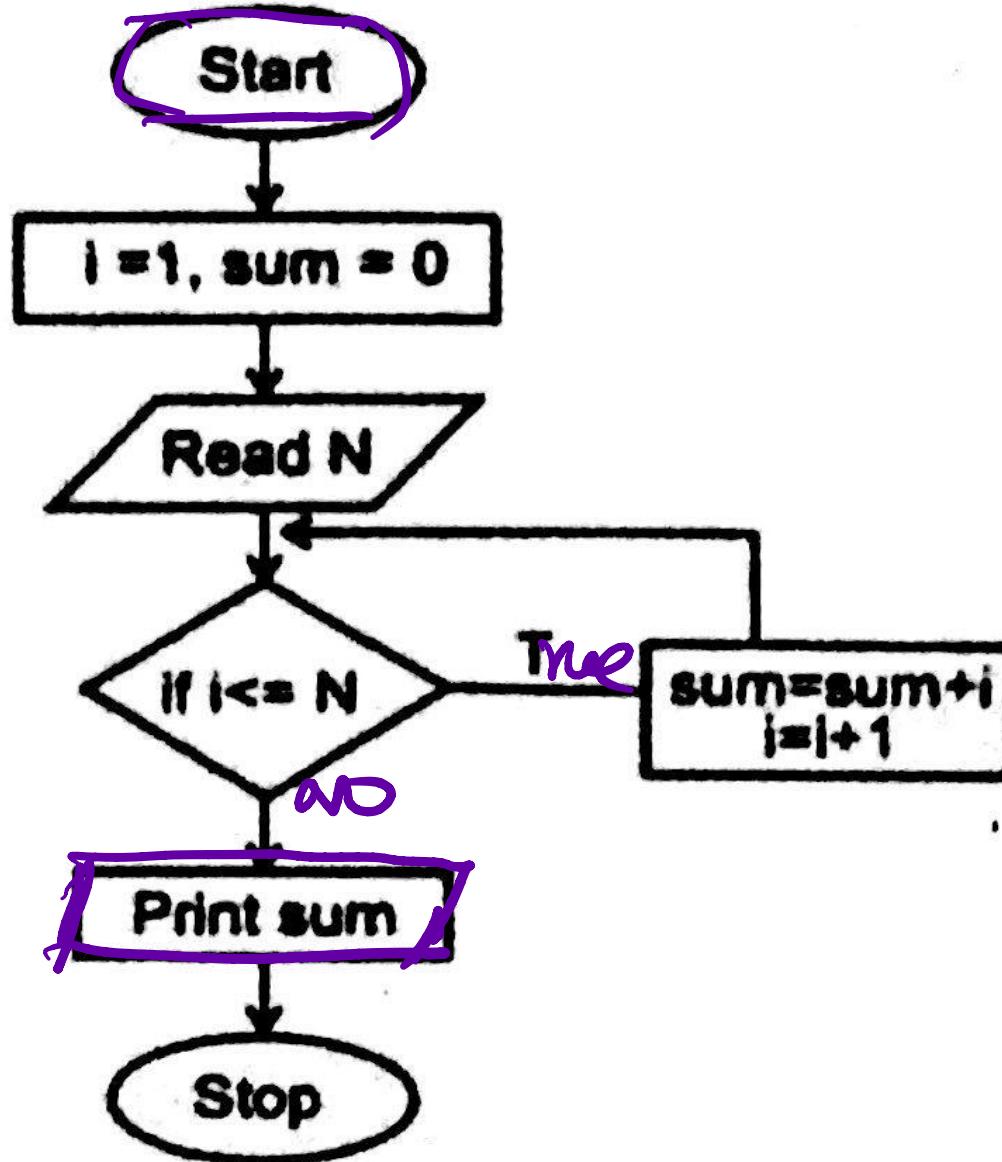
    Step 4.1 : Calculate  $sum = sum + i$

    Step 4.2 : Increment i by 1

Step 5 : Print sum

Step 6 : STOP

# Flowchart to find the sum of n natural numbers



**Ques 41) Write an algorithm and draw a flowchart for finding maximum number out of any given three numbers.**

**Or**

**Write an algorithm to find the largest of three numbers.**

**(2021 [03])**

**Ans: Algorithm for Finding Maximum Number Out of Any Given Three Numbers**

**Step 1: START**

**Step 2: Read a, b, c**

**Step 3:** If  $a > b$

If  $a > c$

Print a;

Else print c

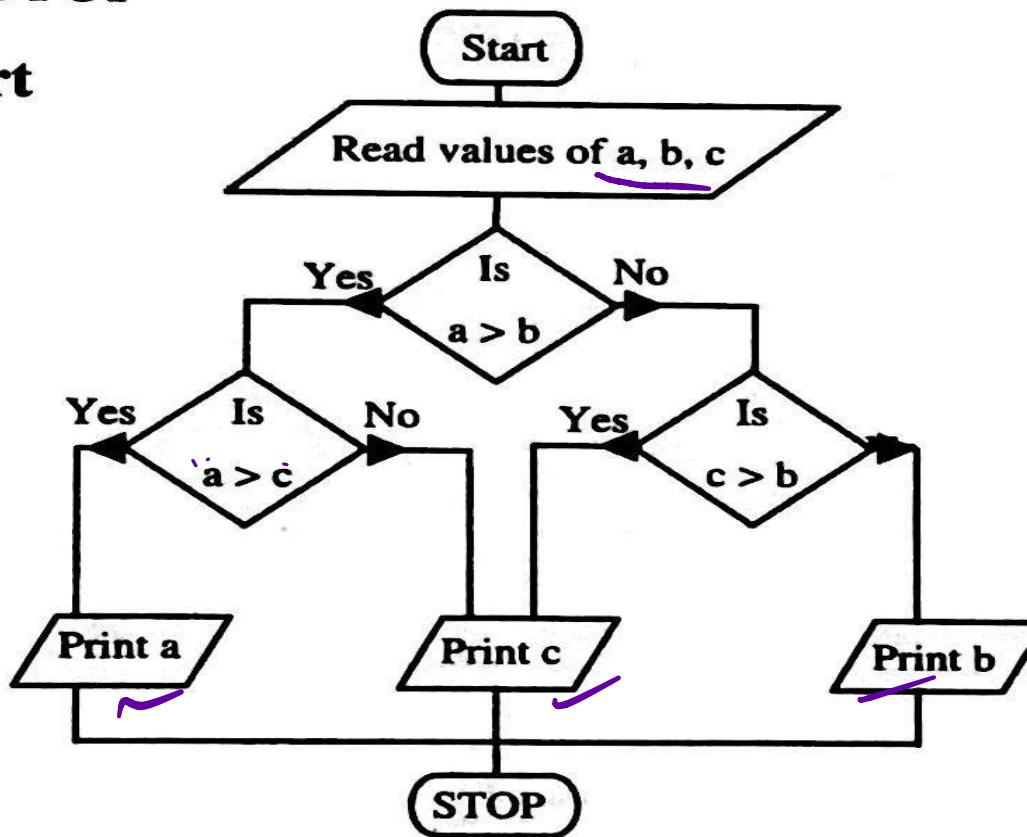
Else if  $c > b$

Print c

Else print b

**Step 4:** STOP

**Flowchart**



**Ques 53) Write an algorithm to find sum of digits of a number. (2021 [07])**

**Or**

**Write algorithm and Draw a flowchart to find the sum of digits of an integer.**

**Ans: Algorithm for Sum of digits**

**Step 1: start**

**Step 2: read N**

**Step 3: initialize the SUM =0**

**Step 4: if N<0 goto Step 7**

**Step 5: if N!=0 goto Step 6 else goto step 7**

**Step 6: store N%10 value in REM**

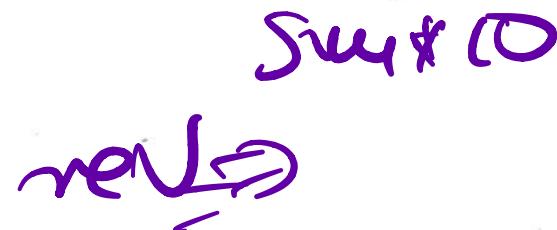
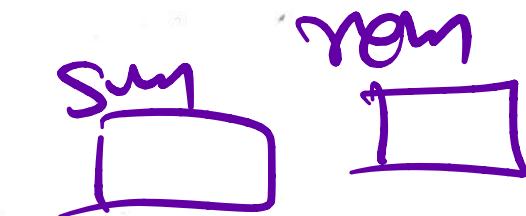
**Add REM value to SUM**

**Assign N/10 value to N**

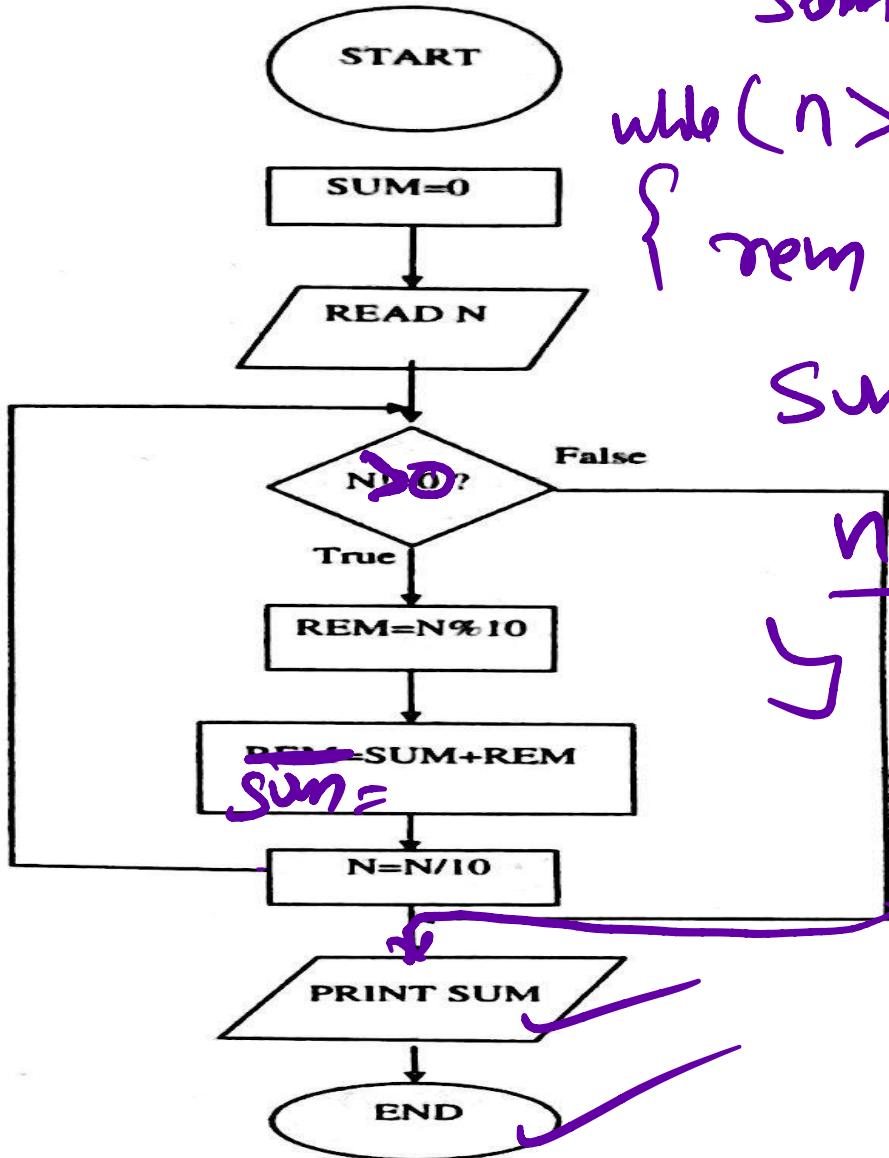
**Goto Step 5**

**Step 7: print the output**

**Step 8: stop**



## Flowchart



Sum: 0    12  $\textcircled{B}$   
while ( $n > 0$ )  
{    rem =  $n \% 10$

$$\text{Sum} = \text{Sum} + \text{rem}$$

$$n = \underline{\underline{n / 10}}$$

Flowchart for Sum of Digits of an Integer

**Ques 55) Using an algorithm and flowchart check whether the given number is palindrome or not?**

**Ans: Check Palindrome**

If a word, phrase, number, or other sequence of symbols or elements whose meaning is same in both directions either forward or reverse direction then that word or phrase is known as palindrome.

**For example, consider the string 'Malayalam' this string is same in both the direction.**

**Algorithm**

**Step 1: START**

**Step 2: Read n**

**Step 3: a = 0, b = n**

**Step 4: While (n > 0)**

**Begin**

$$r = \underline{n \% 10}$$

$$n = \underline{n / 10}$$

$$a = a * 10 + r$$

**End**

**Step 5: If (b = a)**

**Print, 'Number is Palindrome'**

**else**

**Print, 'Number is Not Palindrome'**

**Step 6: STOP**

**Start**

**Read n**

**Set  $\gamma = 0, dup = n, rev = 0$**

**while (n > 0)**

$$\gamma = \underline{n \% 10}$$

$$rev = \underline{rev * 10 + \gamma};$$

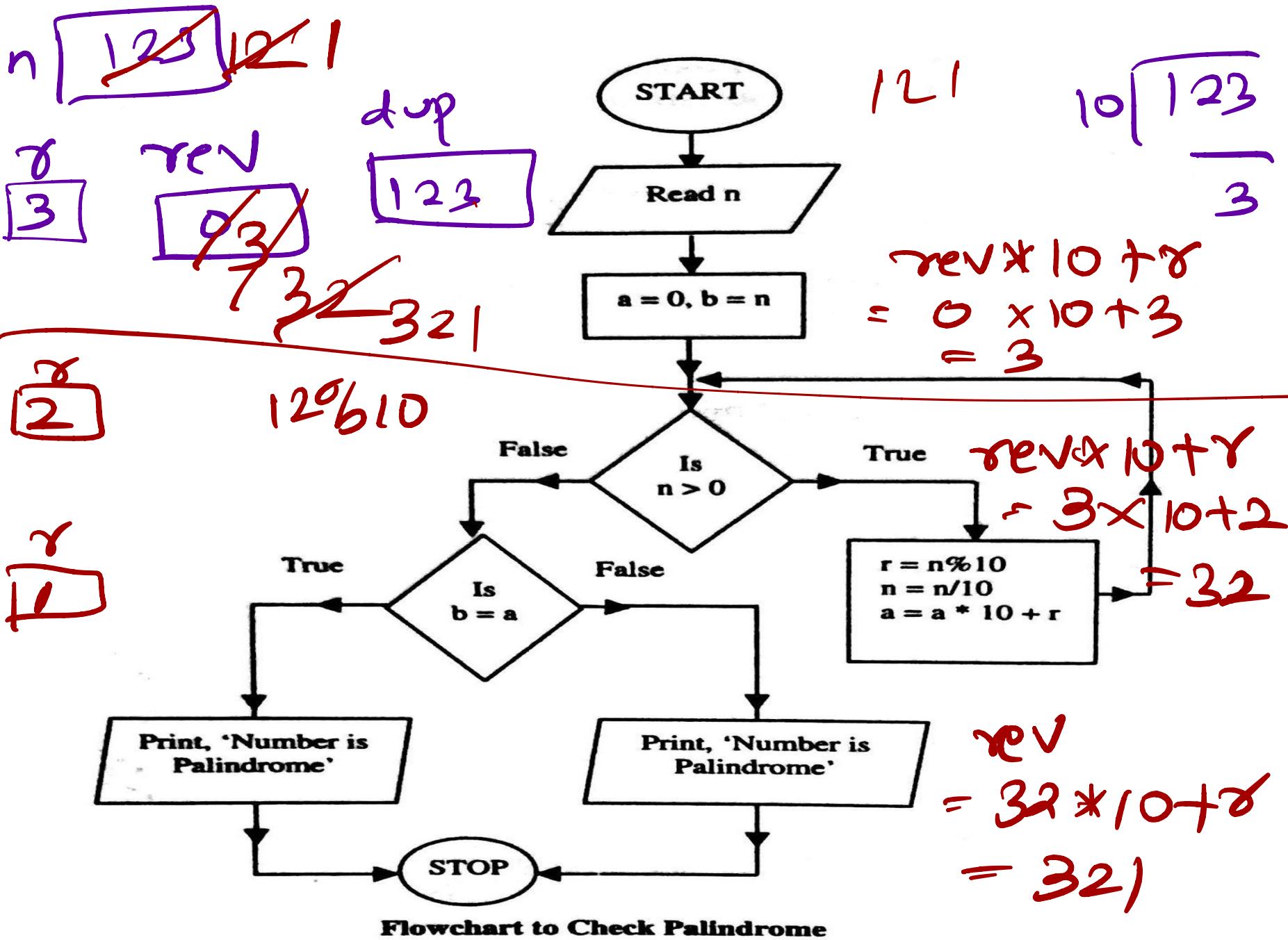
$$n = \underline{n / 10}$$

**if (rev == dup)**

**Print Palindrome**

**else**

**not Palindrome**



**Ques 47) Write an algorithm and draw the flowchart to swap two numbers using third variable.**

**Or**

**Write algorithm and draw flowchart to perform swapping of two numbers. (2021 [08])**

**Ans: Algorithm to Swap Two Numbers Using Third Variable**

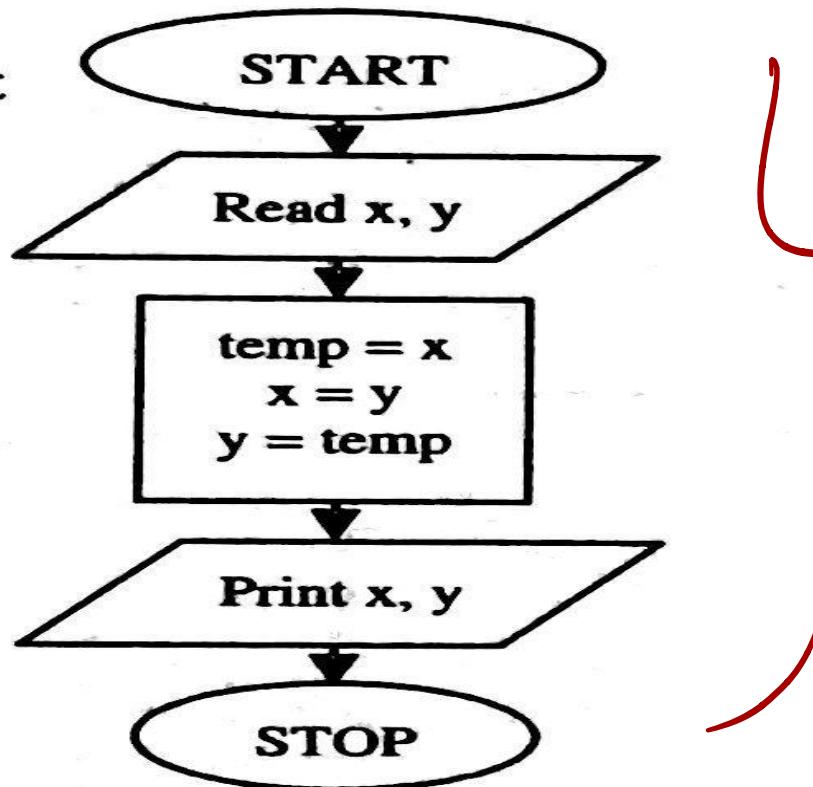
Swapping is the process in which one can interchange the value of two variables with each other. To perform swapping one needs to use third variable to store the values temporary. Let take the two variables x and y, and one temporary variable temp.

- 1) Firstly save the value of variable x to the temp variable.
- 2) Then save the value of y to x.
- 3) In the last step, save the value x from the temp variable to y variable.

**Steps of algorithm are as follows:**

- Step 1: START**
- Step 2: Read x, y**
- Step 3: temp = x**
- Step 4:  $x = y$**
- Step 5:  $y = temp$**
- Step 6: Print x, y**
- Step 7: STOP**

**Flowchart**



# Linear Search

- A *linear search* or *sequential search* is a method for finding an element within a list.
- It sequentially checks each element of the list until a match is found or the whole list has been searched.
- It is the simplest searching algorithm that searches for an element in a list in sequential order.

## *Pseudo code of Linear Search*

Begin

~~Read Search element, e~~

Read n → limit

While ( $i \leq n$ )

    Read num

    if ( $num == e$ )

        Read search element  
        e.

        Print 'Element Found'

    End if

End while

# Algorithm:

Step 1: START

Step 2: Read the element to be checked, val

Step 3: Initialize pos = -1 & flag = 0

Step 4: Initialize i = 0

Step 5: Repeat until i < n

Step 5.1: Check whether a[i] == val, if true, go to step 5.1.1  
otherwise go to step 5.2

Step 5.1.1: set pos = i

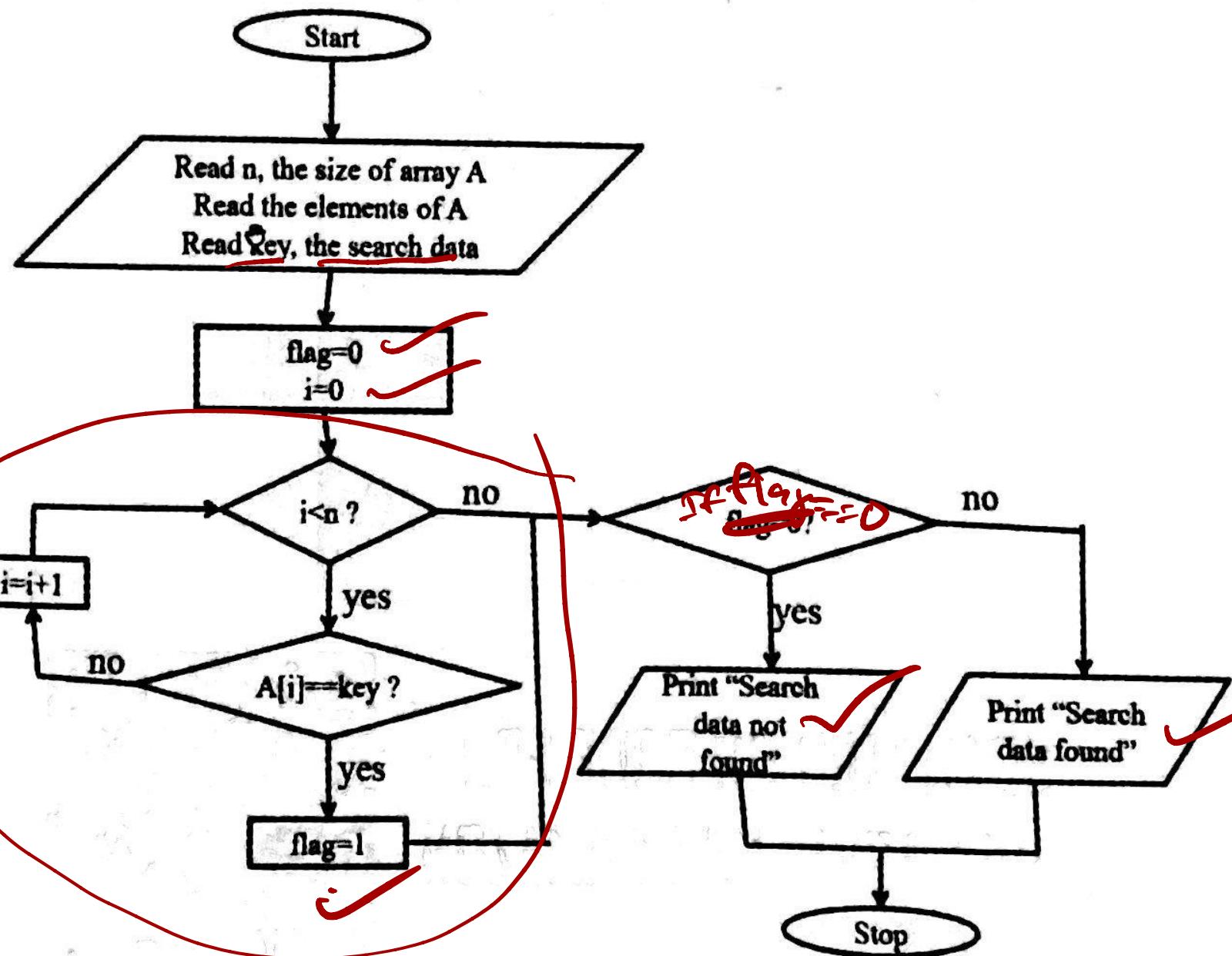
Step 5.1.2: Set flag = 1 and go to Step 6

Step 5.2: Increment i by 1

Step 6: Check whether flag=0, if true then print "Element is not present in the array", otherwise print "The element is found at position, pos"

Step 7: STOP

# Flowchart:



# Bubble Sort

- *Bubble sort*, also referred to as *Comparison sort*, is a simple sorting algorithm.
- It repeatedly goes through the list, compares adjacent elements and swaps them if they are in the wrong order.
- Bubble sort is the easiest sorting algorithm to implement.
- It is an in-place sorting algorithm.

# How Bubble Sort works?

- *Bubble sort uses multiple passes (scans) through an array.*
- *In each pass, bubble sort compares the adjacent elements of the array.*
- *It then swaps the two elements if they are in the wrong order.*
- *In each pass, bubble sort places the next largest element to its proper position.*
- *In short, it bubbles down the largest element to its correct position.*

## Pseudo Code for Bubble Sort:

```
Begin
    for j=1 to n-1
        for i=1 to n-1
            if list[i] > list[i+1]
                end if
            end for
        end for
    end
```

for (i = 0; i < n; i++)  
{  
 for (j = 0; j < n-1-i; j++)  
 {  
 if a(j) > a(j+1)  
 swap (list[i], list[i+1])  
 }  
}

Swap

Step 1: START

Step 2: Read the number of elements, n ✓

Step 3: Read the array of elements

Step 4: Initialize i=1 and j=1.

Step 5: Check whether i< n, if true go to step 6 otherwise go to step.

Step 6: Compare a[j] with a[j+1], if in order go to Step 7 otherwise go

to Step 6.1

Step 6.1: temp=a[j]

Step 6.2: a[j]=a[j+1]

Step 6.3: a[j+1]= temp

Step 7: Increment j by 1

$a[j] > a[j+1]$

Step 8: Check whether  $j \leq n-1$  if true go to Step 6 otherwise goto  
step 9.

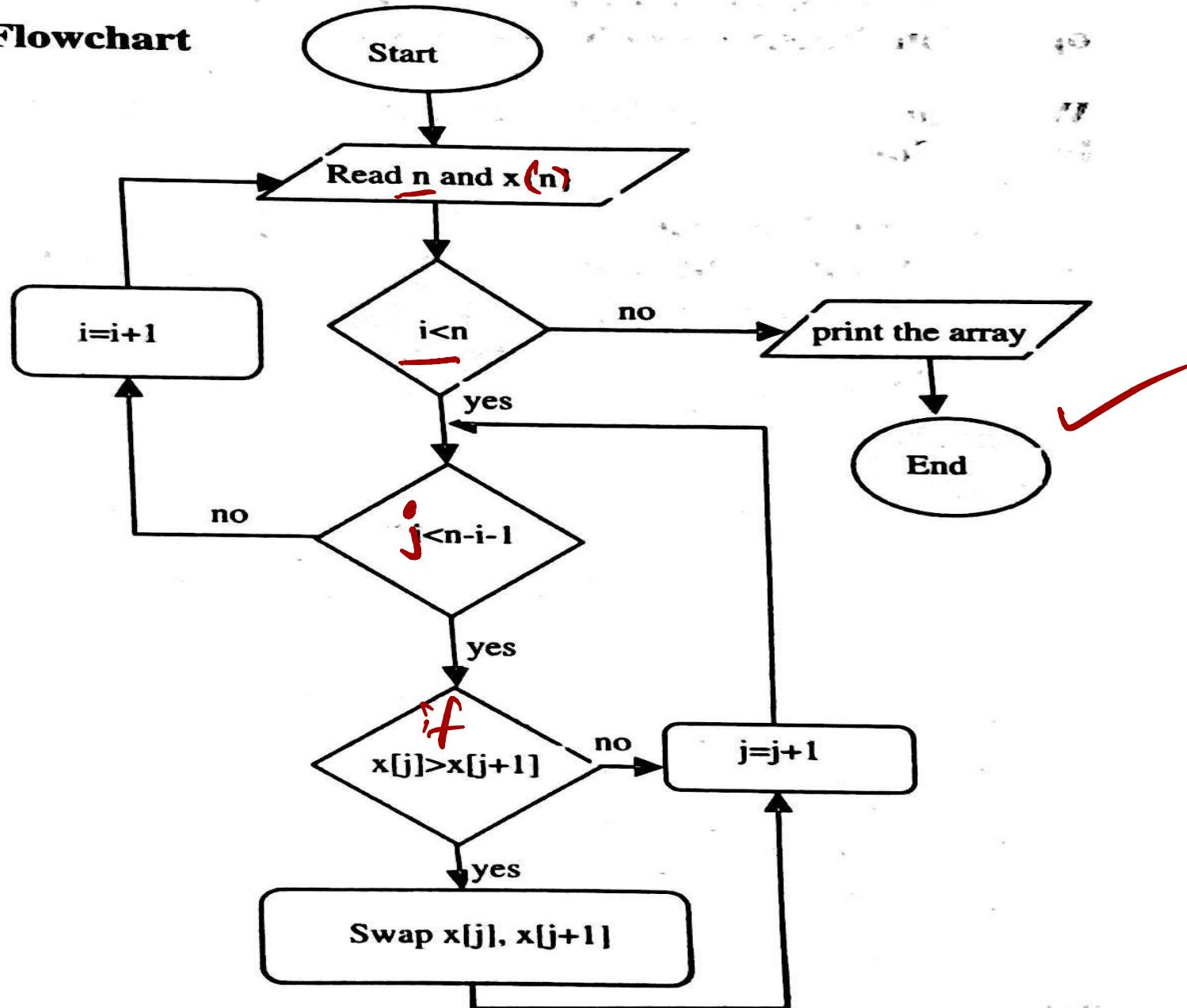
$n-1-j$

Step 9: Increment i by 1 and set j=1

Step 10: Display the sorted array, a

Step 11: STOP

## Flowchart



# **Programming in C**

## Module 2

## History of C :-

BCPL (Basic combined programming language) was developed by martin Richard. BCPL lead to B, it was developed by Ken Thompson. BCPL and B are type less programming languages. B lead to C, it was developed by Dennis Ritchie in 1972, at AT&T Bell laboratory on UNIX machine.

## Structure of C

```
#include<stdio.h> // include statements
void main() // main function
{
    // processing statements;
}
```

Any c program has 3 types of statements.

1. Include statements
2. Main function
3. Processing statements

Include statements :

- ❖ These are the first statements.
- ❖ These are always begins with # symbol.
- ❖ These are also called as pre processor directives.
- ❖ It will execute at compile time.
- ❖ These gives direction to compiler.
- ❖ It copy the content of header file into our program.

## 2. Main function :-

- ❖ It is first executable statement.
- ❖ It is entry point.
- ❖ It is mandatory.
- ❖ It can not be duplicated.
- ❖ It is user defined function.
- ❖ It can be write in 6 forms.

a. main()

b. int main()

c. void main()

d. main(void)

e. int main(void)

f. void main(void)

### 3. Processing statements :-

These are the statements those tells “what to do?”, “when to do?” and “how to do?” in order to solve the problem

#### Statement Terminator ( ; )

- ❖ An instruction given to computer is called as a statement.
- ❖ Every statement must be terminated with semi colon.
- ❖ It is called as statement terminator.

## Open Delimiter ( { ):-

It is used to indicate the beginning of function or method or block or user defined reference.

## Closing Delimiter ( } ):-

It is used to indicate the ending of function or method or block or user defined reference.

## Comment :-

- ❖ Comment is a message.
- ❖ It improves the clarity of program.
- ❖ It gives extra meaning to program.
- ❖ Comments do not compile.
- ❖ Comments do not convert to machine language.
- ❖ Comments do not execute.
- ❖ It is just for user.
- ❖ It is 2 types.
  1. Single line comment

A comment which fit in one line, then it is called single line comment. It is also called as line comment. it always begins with “//”.

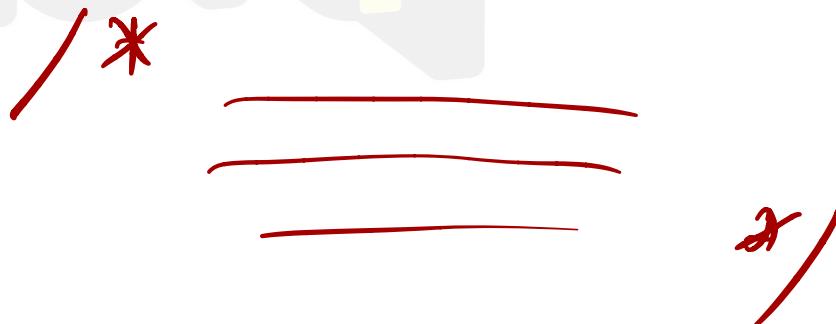
Example : //This is single line comment.

## 2. Multi line comment

A comment which does not fit in one line.

It present in multiple lines. Then it is called multi line comment. It is also called as block comment.  
It always enclosed with in /\* ..... \*/

Example : /\*This is example to Multi line  
comment. It wont execute\*/



/\*  
  \_\_\_\_\_  
  \_\_\_\_\_  
  \_\_\_\_\_  
  )  
      )

## C character set

1. English Alphabet(a...z,A....Z)
2. Digits(0 to 9)
3. Special characters ( +, -, \*, /, ., , )
4. White spaces(blank space,new line,tab etc)

## C Tokens

The smallest individual unit in a program as c tokens.

C tokens are 6 types

1. Keywords
2. Identifiers
3. Constants
4. Strings
5. Special character
6. operators

## Data types :-

- ❖ Data types are the keywords.
- ❖ Those are used to describe the data type.
- ❖ Data types are used to specify the size variables.
- ❖ C language provides 4 types of data types

Sno	Data types	Size	Examples
1	char	1 byte	'a', 'b', 'c', '+', '-', '5', '6'
2	int	2 bytes in 32 bit OS 4 bytes in 64 bit OS	1,2,3,4,....
3	float	4 bytes	1.100000, 1.2, 1.3, 1.111
4	double	8 bytes	1.1000000000, 1.2, 1.3, 1.111

Note : Float maintains 6 precisions.

Double maintains 10 precisions. Example :

float r=100/3=33.333333

double r=100/3=33.3333333333

Modifiers :- Modifier is a keyword which is used to modify the meaning of data type.

Example : Long, short, signed , unsigned.

Example statement :

Long int

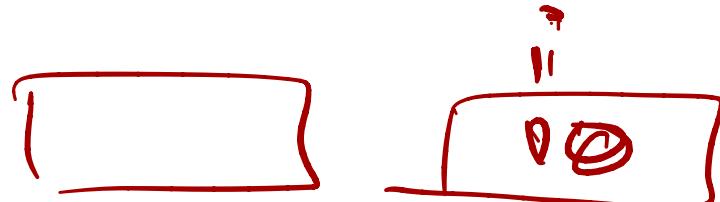
Short int

Signed int

Unsigned int

## Variable :

- ❖ It is a container.
- ❖ It can store the data.
- ❖ It can store only one value at a time.
- ❖ It must be identified by a specific name.
- ❖ These always created inside the memory.
- ❖ It can vary its value, so it is called variable.



Declaration or creation of a variable : It can be created using data type and variable name.

Syntax : **DataType** **VariableName;**

Example : int rno;  
char sname[100];  
float avgmarks;

*eg. int x;*



## Scope rules of variables :-

- ❖ Variable must begin with alphabet or underscore.
- ❖ It does not have any space.
- ❖ It does not allow special characters except underscore.
- ❖ It should not be a keyword.

Valid variable names	Invalid variable names
No1	1no
No2	2no
_no1	Odd no
Odd_no	Odd<no
	Odd-no
	Long
	Short
	Int
	char

rs.

case  
- sensitive

A

a

## Constants :-

- ❖ It is also a variable but it retains its value until program termination.
- ❖ We cannot modify the value of constant.
- ❖ If we try to change then it will produce error.
- ❖ For naming conversion constants is always represented in upper case.  
It can be created in 2 ways.

a. Create using const keyword

Example : const float PI=3.14

b. Create using #define

Example : #define PI 3.14

PI  
3.14

## Identifier :-

The names allocated to variables or constants or functions are called as identifiers.

Example 1: Int rno;

Here rno is an identifier of a variable.

Example 2: #define PI 3.14

Here PI is an identifier of a constant.

Example 3: Void main()

```
{  
}
```

Here main is identifier of a function.

# Keyword :-

- ❖ It is a reserved word.
- ❖ It is a pre defined identifier.
- ❖ We cannot alter the meaning of a keyword.
- ❖ C supports 32 keywords.

Keywords in C Programming			
<u>auto</u>	<u>break</u>	<u>case</u>	<u>char</u>
<u>const</u>	<u>continue</u>	<u>default</u>	<u>do</u>
<u>double</u>	<u>else</u>	<u>enum</u>	<u>extern</u>
<u>float</u>	<u>for</u>	<u>goto</u>	<u>if</u>

Keywords in C Programming			
<u>int</u>	<u>long</u>	<u>register</u>	<u>return</u>
<u>short</u>	<u>signed</u>	<u>sizeof</u>	<u>static</u>
<u>struct</u>	<u>switch</u>	<u>typedef</u>	<u>union</u>
<u>unsigned</u>	<u>void</u>	<u>volatile</u>	<u>while</u>

## Format specifiers :-

- ❖ It is a special character.
- ❖ It is used to specify the format of data.
- ❖ It always begins with % symbol.
- ❖ It can be used with input and output.

Data type	Format specifier
Char	%c
Int	%i, %d
Float	%f
Double	%lf
String	%s
Long int	%ld
Unsigned int	%u
Octal	%o
Hexa decimal	%x, %X
binary	%b

## Escape sequences :-

- ❖ These are the special characters.
- ❖ It always begins with \ symbol.
- ❖ It must use only with output.
- ❖ It is used to escape the sequence of output.

\n

### ~~Example :~~

\t → it is used to move the cursor to a tab distance forward.

\n → it is used to move the cursor to the beginning of next line.

\b → it is used to move the cursor to a char backward.

\r → it is used to move the cursor to the beginning of same line.

\\" → it is used to print \ in output

## Input and output functions :-

C language provides following input and output

Data type	Input / read	Output / print
char	getchar() getche() getch()	putchar()
string	gets()	puts()
char, int, float, double, string, long int, .....	scanf()	printf()

### Getchar() :-

- ❖ It is used to read a char.
- ❖ It wait for enter key.
- ❖ It read only first char of input.
- ❖ It is a input function.

### getche()

- ❖ It is used to read a char.
- ❖ It does not wait for enter key.
- ❖ It reads the data with echo/display.
- ❖ It is an input function.

### getch() :-

- ❖ It is used to read a char.
- ❖ It does not wait for enter key.
- ❖ It read the data without echo/display.
- ❖ It is an input function.

### puts () :-

- ❖ It is used to print a string.
- ❖ It is an output function.

### Putchar () :-

- ❖ It is used to print a char.
- ❖ It is an output function.

### gets() :-

- ❖ It is used to read a string.
- ❖ It stands for get string.
- ❖ It allow space in a string.
- ❖ It is an input function.

apple

Scanf() :-

- ❖ It can read any type of data.
- ❖ It can read multiple values.
- ❖ It stands for scan formatted.
- ❖ It is a general input function.

Syntax :

scanf("Format specifiers",&variableName);



*Address*

Printf() :-

- ❖ It stands for print formatted.
- ❖ It is used to print any type of data.
- ❖ It can print multiple values.
- ❖ It is general output function.

Syntax :

printf(“Format specifier”,variableList);

Types of operators :-

C provides 9 types of operators.

1. Arithmetic operators.
2. Logical operators
3. Relational operators
4. Bitwise operators
5. Assignment operators
6. Increment or decrement operators.
7. Compound operators
8. Conditional operators.
9. Special operators

## 1. Arithmetic operators

The operators those are used to evaluate the arithmetic expression are called as arithmetic operators.

Examples :

+, -, \*, /, %, mod → it gives remainder )

$$\begin{array}{r} 1 \\ 2 \overline{) 5} \\ \underline{-4} \\ 1 \end{array}$$

$$a = 5 \% 3$$

$\%$  → remainder after division

$\boxed{2}$

WAP to read a number. Find the last digit of given number

```
#include<stdio.h>
void main()
{
    int no,rem;
    printf("Enter a number : ");
    scanf("%d",&no);
    rem=no%10;  
    printf("Last digit = %d",rem);
}
```

123  
n%10

WAP to read a number, remove the last digit of given number.

```
#include<stdio.h>
void main()
{
    int no;
    printf("enter a number : ");
    scanf("%d",&no);
    no=no/10;
    printf("Number after remove the
lastdigit=%d",no);
}
```

$$\begin{array}{r} \cancel{1} \cancel{2} \\ 10 ) \overline{1} \cancel{2} \\ 1 \quad 2 \\ \hline 3 \end{array}$$

$$n = \underline{n/10}$$

## 2. Logical operators :

- ❖ The operators those works on Boolean values( True and False ).
- ❖ These gives Boolean value( True and False ) as a result.

❖ These conditions Example

Truth table of And		
Exp1	Exp2	Res
T	T	T
T	F	F
F	T	F
F	F	F

Truth table of OR		
Exp1	Exp2	Res
T	T	T
T	F	T
F	T	T
F	F	F

Truth table of Not	
Exp1	Res
T	F
F	T

&

||

!

### 3. Relational operators :-

- ❖ The operators those are used to check the relationship between 2 numbers.
- ❖ It always gives true or false as result.

Example :

, <=, >=, ==, !=

$x == 5$

$= -$

$x != 5$

$>$

$\geq$

$<$

$\leq$

#### 4. Bitwise operators :-

- ❖ These works only on numbers.
- ❖ It works on bits of operand.

Examples :

~~<< → Left shift operator~~

~~>> → right shift operator~~

~~& → bitwise and operator~~

~~| → bitwise or operator~~

~~^ → bitwise Exclusive OR operator~~

~~~ → bitwise negation operator~~

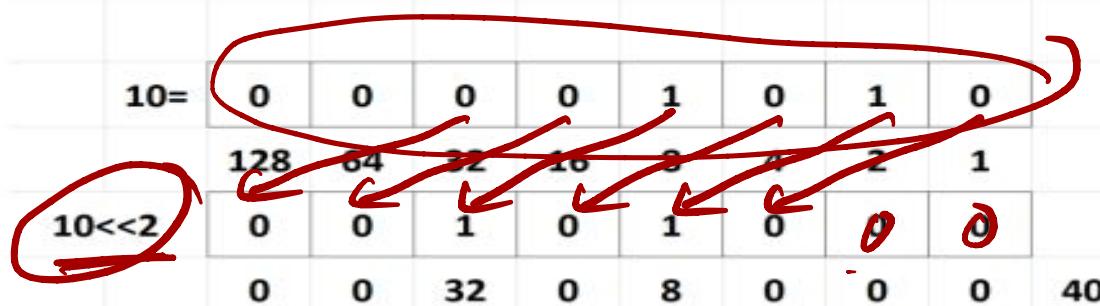
$$\begin{array}{r} 1010 \\ 0110 \\ \hline 0 \end{array}$$

$$\begin{array}{r} 01010 \\ \swarrow \searrow \swarrow \searrow \\ 10100 \end{array}$$

### << ( Left Shift Operator ) :-

**It shifts all the bits operands towards left side by given number of times.**

$$10 << 2 = ?$$

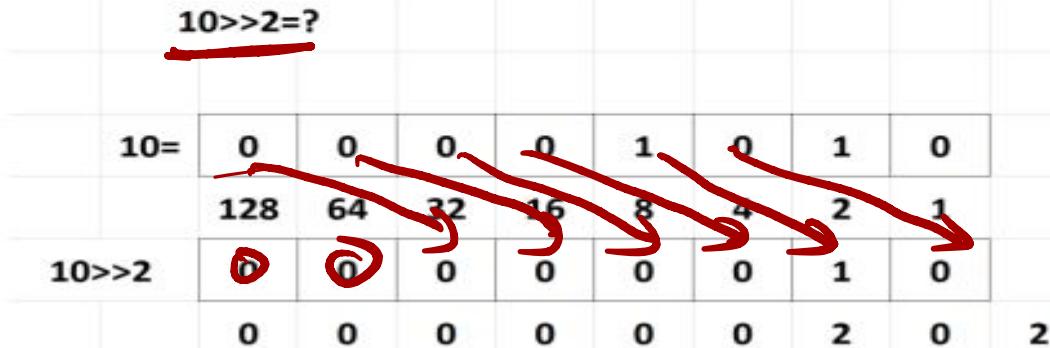


10<<2

**>> ( Right shift operator ) :-**

**It shifts all the bits of operand towards right side by given number of times.**

$10 >> 2 = ?$



## & (Bitwise and operator) :-

- ❖ It works on 2 operands.
- ❖ It tests 2 bits of 2 operands.
- ❖ It gives 1 when both the bits are 1.
- ❖ Otherwise it gives 0.

| 10&7=? |     |    |    |    |   |   |   |   |
|--------|-----|----|----|----|---|---|---|---|
| 10=    | 0   | 0  | 0  | 0  | 1 | 0 | 1 | 0 |
| 7=     | 0   | 0  | 0  | 0  | 0 | 1 | 1 | 1 |
|        | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
| 10&7=  | 0   | 0  | 0  | 0  | 0 | 1 | 0 |   |
|        | 0   | 0  | 0  | 0  | 0 | 0 | 2 | 0 |

## | (Bitwise OR operator ) :-

- ❖ It works on 2 operands
- ❖ It test 2 bits of 2 operands.
- ❖ It gives 1 when either one bit is one.
- ❖ Otherwise it gives 0.

10|7=?

10= 0 0 0 0 1 0 1 0

1 1 1

7= 0 0 0 0 0 1 1 1

128 64 32 16 8 4 2 1

10|7= 0 0 0 0 1 1 1

4 2 1

0 0 0 0 8 4 2 1 15

## $^$ (Bitwise Exclusive OR operator) :-

- ❖ It works on 2 bits.
- ❖ It test 2 bits of 2 operands.
- ❖ It gives 0 when both the bits are same.
- ❖ Otherwise it gives 1.

| $10 \wedge 7 = ?$ |                 |    |    |    |   |   |   |   |    |
|-------------------|-----------------|----|----|----|---|---|---|---|----|
| 10=               | 0 0 0 0 1 0 1 0 |    |    |    |   |   |   |   |    |
| 7=                | 0 0 0 0 0 1 1 1 |    |    |    |   |   |   |   |    |
|                   | 128             | 64 | 32 | 16 | 8 | 4 | 2 | 1 |    |
| $10 \wedge 7 =$   | 0 0 0 0 1 1 0 1 |    |    |    |   |   |   |   |    |
|                   |                 |    |    |    | 8 | 4 | 0 | 1 | 13 |

## $\sim$ (Bitwise Negation operator) :-

- ❖ It works on single operand.
- ❖ It performs one's complement.

| $\sim 10 = ?$ |      |    |    |    |   |   |   |     |
|---------------|------|----|----|----|---|---|---|-----|
| 10 =          | 0    | 0  | 0  | 0  | 1 | 0 | 1 | 0   |
|               |      |    |    |    |   |   |   |     |
| MSB           | 128  | 64 | 32 | 16 | 8 | 4 | 2 | LSB |
| $\sim 10 =$   | 1    | 1  | 1  | 1  | 0 | 1 | 0 | 1   |
|               |      |    |    |    |   |   |   |     |
| SB            | -128 | 64 | 32 | 16 | 0 | 4 | 0 | -11 |

## 5. Assignment operators :-

It is used to assign the result to a variable.

It gives the result of right hand side expression to left hand side variable.

Example:

$R=10$

$R=5+6$



$$x = 10$$

$$c = \underline{a + b}$$

## 6. Compound operator :-

- ❖ It is a combination of assignment operator and arithmetic operator.
- ❖ It is also called as short hand operator.
- ❖ Example :

$+=, -=, *=, /=, \%=$

Example :  $A=10$

$A=A+5 \rightarrow A+=5$

$A=A-5 \rightarrow A-=5$

$A=A*2 \rightarrow A*=2$

$A=A/2 \rightarrow A/=2$

$A=A\%2 \rightarrow A\%=2$

$a += 5$

$\underline{a = a + 5}$

## 7. Increment or Decrement operators :-

❖ The operators those are used to increase or decrease the value of operands.

❖ Examples :

❖ ++, --, -

Postfix

i = 10;

x = i++;

;

~~10~~  
++

10

Prefix

i = 10;

x = ++i;

;

~~10~~  
++

11

x

**++ :-**

- ❖ It is an increment operator.
- ❖ It increases the value of operand by 1.
- ❖ It can be used both the sides of operand.
- ❖ If you use ++ before the operand then it is called pre increment . ❖ If you use ++ after the operand then it is called post increment.

Example 1 : Int  
a=10; a++;

Output : a=11

Example 2 : Int a=10 ++a;

Output : a=11

Example3 : Int a=10,b; b=++a;

Output : a=11, b=11

-- :-

- ❖ It is decrement operator.
- ❖ It is used to decrease the value by 1.
- ❖ It can be used both the sides of operands.
- ❖ If you use -- before the operand then it is called pre decrement.
- ❖ If you use -- after the operand then it is called post decrement.

Example 1 : A=10 --a;

Output : a=9

Example 2 : A=10 a--;

output : a=9

example 3: a=10,b; b=a--;

output : b=10 a=9

- :

- ❖ It is called both increment and decrement operator.
- ❖ It can increase or decrease the value of operand.

Example 1: Int a=10;

a=-1\*a → -1\*10 → -10

Example 2 : Int a=-10;

a= -1\*a → -1 \* -10 → +10

## 8. Conditional operators

- ❖ The operators those works on given condition.
- ❖ It is also called ternary operator.
- ❖ It works on more than 2 operands.

Syntax : Condition ? value if true : value if false;

Example : Big =  $a > b ? a : b$

Big =  $(a > b) ? a : b$        $0 \leq x \leq 5$

$a$        $\leftarrow$  True  
                False

$x = (x > 5) ? 1 : 0$

WAP to find big number between 2 numbers

```
#include<stdio.h>
void main()
{
    int a,b,big;
    printf("Enter 2 numbers : ");
    scanf("%d%d",&a,&b);
    big=(a>b)?a:b;
    printf("Big number=%d",big);
}
```

WAP to find big number between 3 numbers

```
#include<stdio.h>
void main()
{
    int a,b,c,big;
    printf("Enter 3 numbers : ");
    scanf("%d%d%d",&a,&b,&c);
    big= (a>b?a:b) > c ? (a>b?a:b) : c;
    printf("Big number=%d",big);
}
```

$a > b ? a : b$

$\Leftarrow$   $\Rightarrow$

$a > c ? a : c$

$\leftarrow$  true

false

## 9. Special operators :-

C language provides 2 types of special operators.

- a. Sizeof()
- b. Comma( , )

Sizeof() :-

- ❖ It gives the size of operand.

int x ;

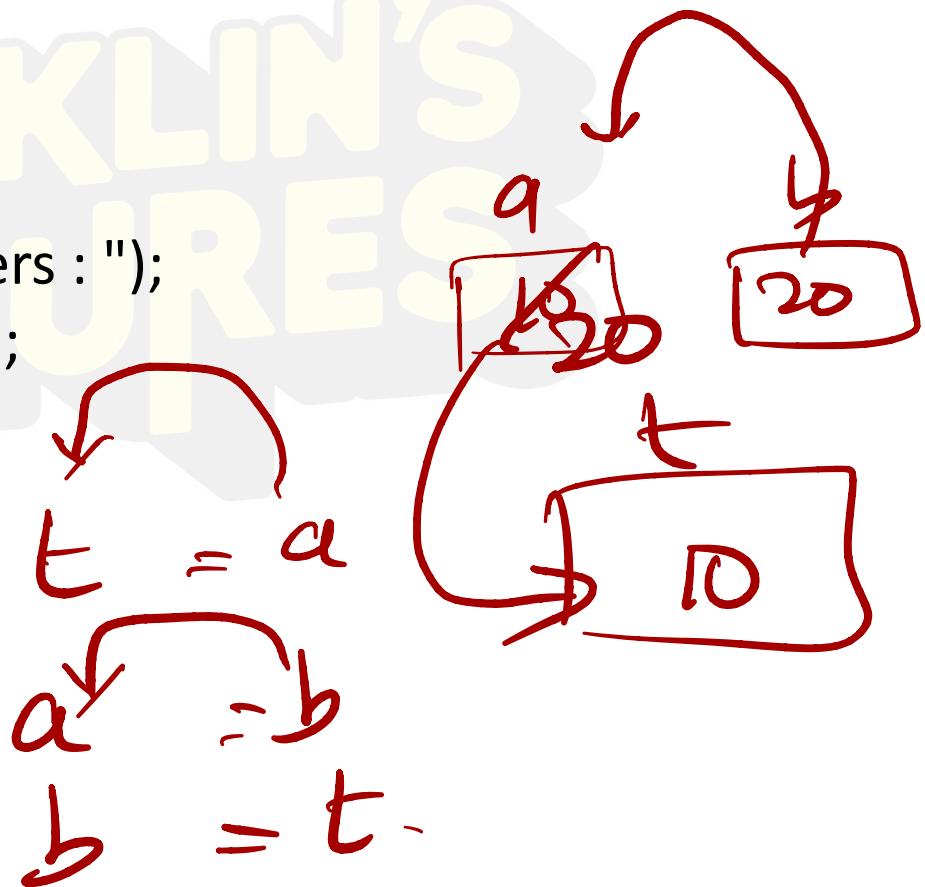
Sizeof(x)

64-Bit OS

→ 4 Byte.

WAP to read 2 numbers. Swap the numbers.

```
#include<stdio.h>
void main()
{
    int a,b,t;
    printf("Enter 2 numbers : ");
    scanf("%d%d",&a,&b);
    t=a;
    a=b;
    b=t;
    printf("a=%d",a);
    printf("\nb=%d",b);
}
```



WAP to swap 2 numbers without using third variable.

```
#include<stdio.h>
void main()
{
    int a,b;
    printf("Enter 2 numbers : ");
    scanf("%d%d",&a,&b);
    a=a+b;
    b=a-b;
    a=a-b;
    printf("a=%d",a);
    printf("\nb=%d",b);
}
```

~~a  
10  
30  
20~~

~~b  
20  
10~~

## Control Statements

- ❖ The statements those are used to control the flow of execution.

### Simple If :-

- ❖ It is a control statement.
- ❖ It is a conditional statement.
- ❖ It is a selective statement
- ❖ It is a branching statement.
- ❖ It works based on given condition.
- ❖ Syntax : if( condition )  
{ body if true }

if ( condition )

{  
=

if ( condition )

=  
else  
,  
=

WAP to read a lower case char. Convert it to upper case.

```
#include<stdio.h>
void main()
{
    char ch;
    printf("Enter a lower case char : ");
    scanf("%c",&ch);
    ch=ch-32;
    printf("Upper case char=%c",ch);
}
```

char

ch

Ch = ch = 32

WAP to read a upper case char, convert it to lower case.

```
#include<stdio.h>
void main()
{
char ch;
printf("Enter a upper case char : ");
scanf(" %c",&ch);
ch=ch+32;
printf("Lower case char=%c",ch);
}
```

## If – else statement

❖ Syntax :

```
if( condition )  
{ body if true }
```

Else

```
{ body if false }
```

WAP to read a number. Print whether it is odd or even.

```
#include<stdio.h>
void main()
{
int no;
printf("Enter a number : ");
scanf("%d",&no);
if( no%2==0 )
    printf("Even number ");
else
    printf("Odd number");
}
```

$$1 \text{ } A \text{ } n \text{ } b_2 = 0$$

WAP to read a char. Print whether it is upper case or not.

```
#include<stdio.h>
void main()
{
    char ch;
    printf("Enter a char : ");
    scanf(" %c",&ch);
    if( ch>='A' && ch<='Z' )
        printf("Upper case ");
    else
        printf("Not a Upper case");
}
```

WAP to read a char. Print whether it is digit or not.

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

```
void main()
```

```
{
```

```
char ch;
```

```
printf("Enter a digit : ");
```

```
scanf(" %c",&ch);
```

```
if( ch>='0' && ch<='9' )
```

```
    printf("Digit ");
```

```
else
```

```
    printf("Not a Digit");
```

```
}
```

WAP to read a character. Print whether it is vowel or consonant.

```
#include<stdio.h>
void main()
{
    char ch;
    printf("Enter a char : ");
    scanf(" %c",&ch);
    if( ch=='a' || ch=='e' || ch=='i' || ch=='o' || ch=='u'
        || ch=='A'
        || ch=='E' || ch=='I' || ch=='O'|| ch=='U')
        printf("Vowel");
    else
        printf("Consonant");
```

If else ladder / Stair case if / if else – if else –if

Syntax : if( condition )

{ body }

else if( condition )

{

} .....

body;

Else

{

body;

}



~~Iml~~

WAP to read marks in 3 subjects. Print the grade.

```
#include<stdio.h>
void main()
{
    int avg;
    printf("Enter avg marks : ");
    scanf("%d",&avg);
    if( avg>=60 )
        printf("I Class");
    else if( avg>=50 )
        printf("II Class");
    else
        printf("III Class");
}
```

70

56

Switch :

- ❖ It is a control statement.
- ❖ It is a decision making statement.
- ❖ It is a conditional statement.
- ❖ It is multi way selection statement.
- ❖ It is a branching statement.
- ❖ It is used when we need to select a single choice from group of choices.

Syntax : switch ( expression )

```
{  
    case constant1: body;  
    break;  
    case constant2: body;  
    break;  
    .....  
    case constantN : body; break;  
    default: body;  
}
```

- Note : ♦ Break is optional.
- ♦ Body is optional.
- ♦ Cases also optional.
- ♦ Default also optional.
- ♦ Order of case is user choice.
- ♦ Place of default is user choice.
- ♦ Case should be a constant/constant expression.
- ♦ Cases should not be duplicated.
- ♦ Switch allows to pass only char or only integer as input.

## WAP to read a single digit number. Spell it.

```
#include<stdio.h>
void main()
{
    int no;
    printf("enter a single digit number : ");
    scanf("%d",&no);
    switch( no )
    {
        case 0: printf("Zero"); break;
        case 1: printf("One"); break;
        case 2: printf("Two"); break;
        case 3: printf("Three"); break;
        case 4: printf("Four"); break;
        case 5: printf("Five"); break;
        case 6: printf("Six"); break;
        case 7: printf("Seven"); break;
        case 8: printf("Eight"); break;
        case 9: printf("Nine"); break;
        default: printf("Invalid input");
    }
}
```

WAP to read 2 numbers, 1 operator. Find the result  
based on operator.

```
#include<stdio.h>
void main()
{
int a,b,r;
char op;
printf("Enter 2 numbers, 1 operator : ");
scanf("%d%d%c",&a,&b,&op);
switch( op )
{
case '+': r=a+b; break;
case '-': r=a-b; break;
case '*': r=a*b; break;
case '/': r=a/b; break;
default : printf("Invalid input");
}
printf("Result=%d",r);
}
```

WAP to read a character. Print whether it is vowel  
or consonant.

```
#include<stdio.h>
void main()
{
    char ch;
    printf("Enter a char : ");
    scanf(" %c",&ch);
    switch( ch )
    {
        case 'a': printf("Vowel"); break;
        case 'e': printf("Vowel"); break;
        case 'I': printf("Vowel"); break;
        case 'o': printf('Vowel'); break;
        case 'u': printf("Vowel"); break;
        case 'A': printf("Vowel"); break;
        case 'E': printf("Vowel"), break;
        case 'I': printf("Vowel"); break;
        case 'O': printf("Vowel"); break;
        case 'U': printf("Vowel"); break;
        default : printf("Consonants");
    }
}
```

WAP to read a single digit number. Print whether it  
is odd or even using switch case.

```
#include<stdio.h>
void main()
{
int no;
printf("Enter a single digit number : ");
scanf("%d",&no);
switch( no)
{
case 1:
case 3:
case 5:
case 7:
case 9: printf("Odd number"); break;
case 0:
case 2:
case 4:
case 6:
case 8: printf("Even number"); break;
default : printf("Invalid input");
}
```

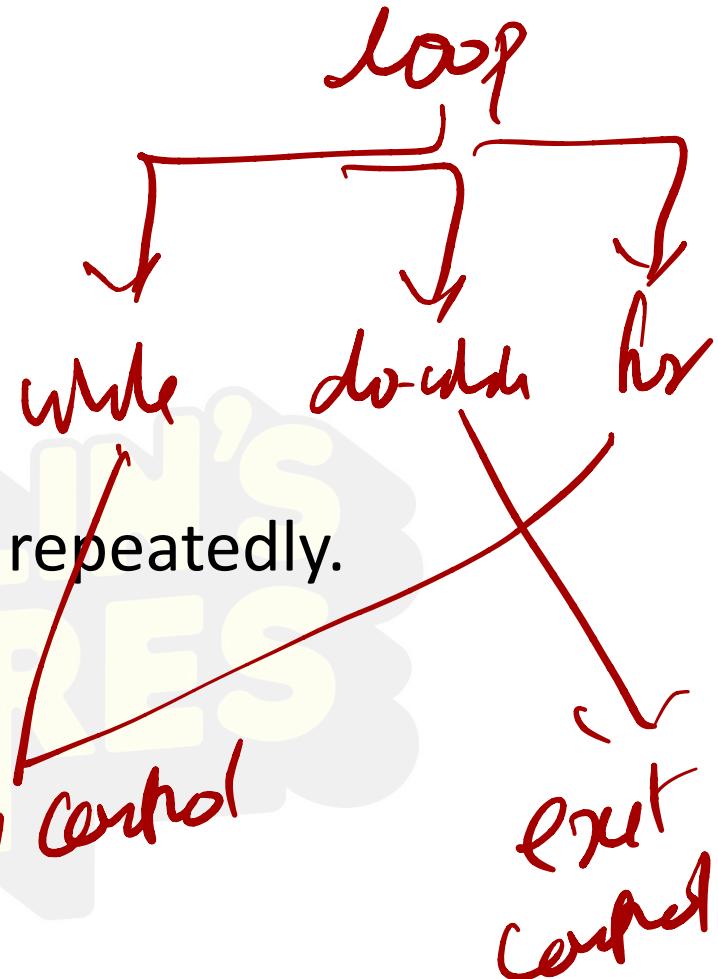
While :-

- ❖ It is a control statement
- ❖ It is a conditional statement.
- ❖ It is an iterative tool.
- ❖ It is a pre test loop.
- ❖ It is used to execute the code repeatedly.

Syntax : while( condition )

```
{  
Body;  
}
```

*Entry control*



WAP to print natural numbers from 1 to 100.

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

```
{
```

```
int no=1;
```

```
while( no<=100 )
```

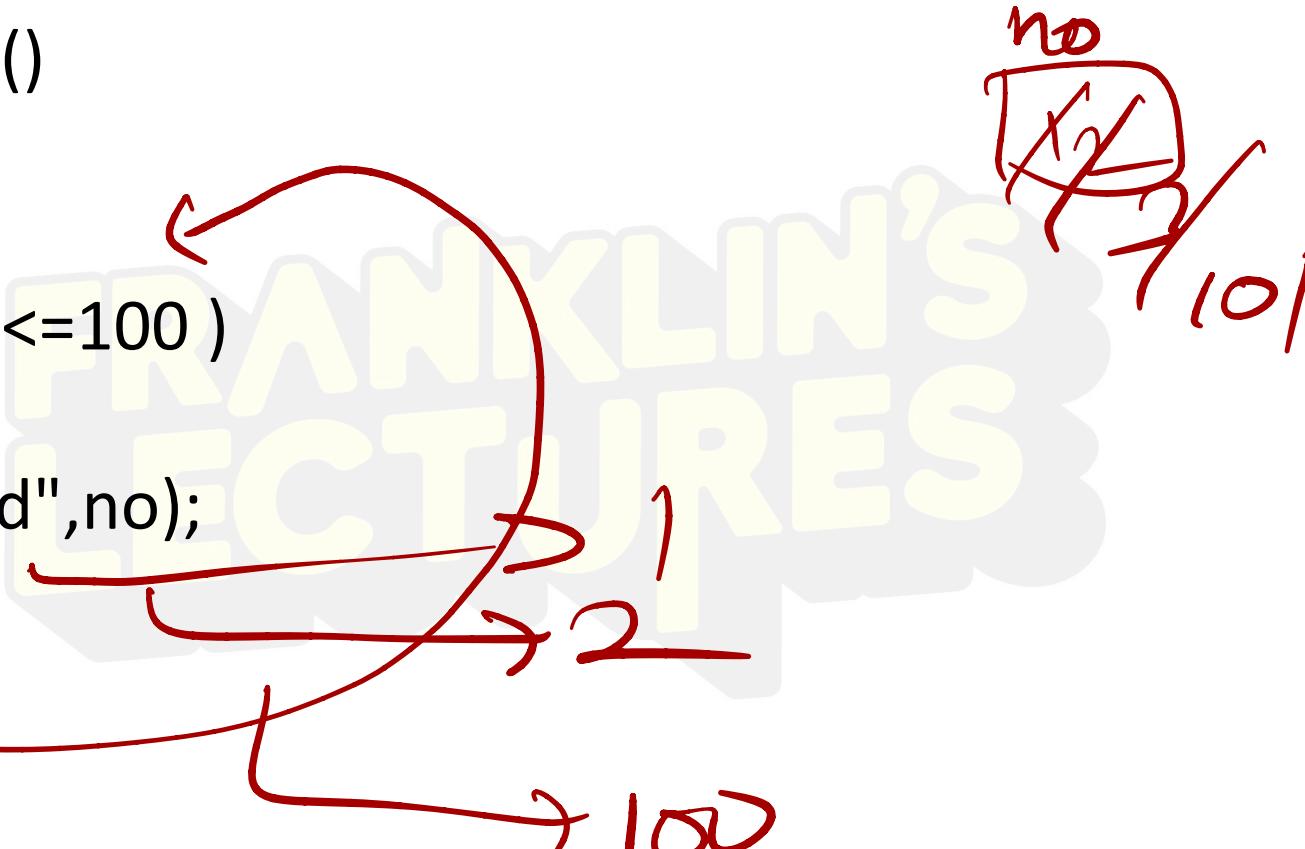
```
{
```

```
printf("%d",no);
```

```
no++;
```

```
}
```

```
}
```



WAP to print odd numbers from 1 to 100.

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

```
void main()
```

```
{
```

```
int no=1;
```

```
while( no<=100 )
```

```
{
```

```
printf("%d",no);
```

```
no+=2;
```

```
}
```

```
}
```

1, 3, 5, 7  
no = no + 2

99

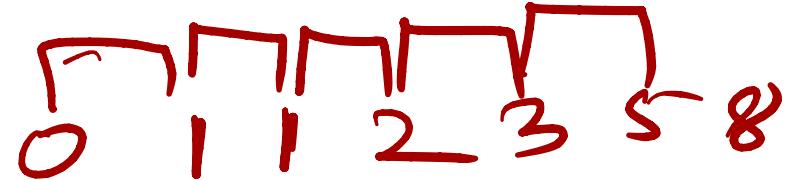
WAP to print natural number those are divisible by  
5 from 1 to 100.

```
#include<stdio.h>
void main()
{
    int no=1;
    while( no<=100 )
    {
        if( no%5==0 )
            printf("%d",no);
        no++;
    }
}
```

if ( $n \% 5 == 0$ )

WAP to print Fibonacci series.

```
#include<stdio.h>
void main()
{
    int a=0,b=1,c,i=3;
    printf("%5d%5d",a,b);
    while( i<=10 )
    {
        c=a+b;
        printf("%d",c);
        i++;
        a=b;
        b=c;
    }
}
```



WAP to read a number. Print whether it is  
palindrome or not

```
#include<stdio.h>
void main()
{
    int no,rem,rev=0,dup;
    printf("Enter a number : ");
    scanf("%d",&no);
    dup=no;
    while((no!=0))
    {
        rem=no%10;
        rev=rev*10+rem;
        no=no/10;
    }
    if( rev==dup )
        printf("Palindrome ");
    else
        printf("Not a palindrome");
}
```

$$dup = n$$

$rev = 0$

$while(n > 0)$

{

$r = no \% 10;$

$rev = rev * 10 + r;$

$n = n / 10;$

}

$if(dup == rev)$

    ↳ Palindrome?

WAP to read a number. Print whether it is  
Armstrong number or not.

```
#include<stdio.h>
void main()
{
int no,rem,sum=0,dup;
printf("Enter a number : ");
scanf("%d",&no);
dup=no;n>0
while(no!=0)
{
rem=no%10;
sum=sum+rem*rem*rem;
no=no/10;
}
if( sum==dup)
printf("Armstrong number ");
else
printf("Not an armstrong number");
}
```

153

$$1^3 + 5^3 + 3^3 = 153$$

1234

$$1^4 + 2^4 + 3^4 + 4^4 = 1234$$

1234

Lecture 1

WAP to read a number. Find first and last digits of a number.

```
#include<stdio.h>
void main()
{
    int no,fd,id;
    printf("Enter a number : ");
    scanf("%d",&no);
    id=no%10;
    while( no!=0 )
    {
        fd=no%10;
        no=no/10;
    }
    printf("First digit=%d",fd);
    printf("\nLast digit=%d",id);
}
```

For :-

- ❖ It is a control statement.
- ❖ It is a conditional statement.
- ❖ It is an iterative statement.
- ❖ It is a pre test loop.
- ❖ It can carry initialization, condition and inc/dec in one line.

Syntax :

```
for( initialization ; condition ; inc/dec )  
{  
    Body;  
}
```

for(  $i = 1; i \leq (0); i++$  )  
{ print("Id", i); }  
y

Note :

- ❖ If there are more than one initialization then we have to separate them by comma.
- ❖ If there are multiple increments or decrements then we have to separate them by comma.
- ❖ If there are multiple conditions then we have to combine them by logical operators.

Example 1 : `for( i=1 , j=5; i<=5 && j>=1 ; i++ , j--)`

{ Body;

}

- ❖ If there is no initialization then we can omit that part.
- ❖ If there is not condition then we can omit that part.
- ❖ If there is no increment or decrements then we can omit that part.

Example : `For( ; ; ) { }`

- ❖ If there is no body then we have to terminate with semi colon.

Example : `for( i=1;i<=10;i++) ;`

WAP to read a number. Print whether it is prime number or not

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

```
void main()
```

```
{
```

```
int no,c=0,i;
```

```
printf("Enter a number : ");
```

```
scanf("%d",&no);
```

```
for( i=1;i<=no;i++ )
```

```
{
```

```
if( no%i==0 )
```

```
c++;
```

```
}
```

```
if( c==2 )
```

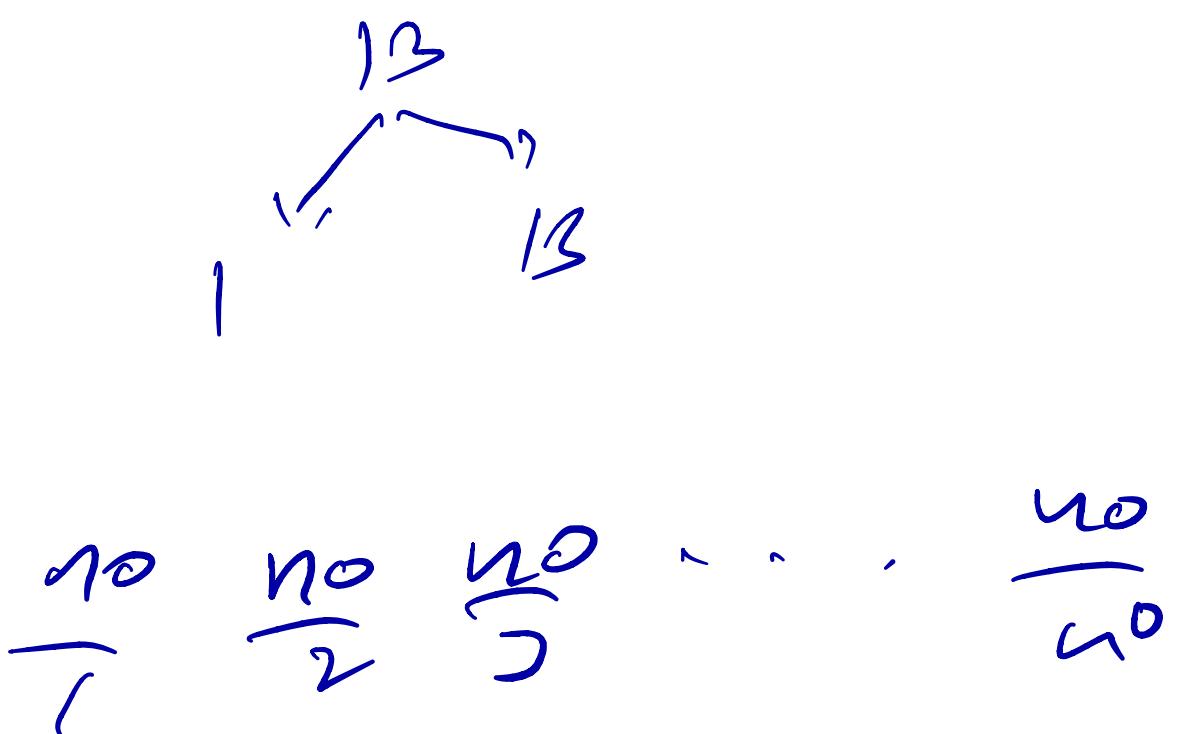
```
printf("Prime number ");
```

```
else
```

```
printf("Not a prime number");
```

```
}
```

$$c=0$$



WAP to read a number. Find the factorial value.

```
#include<stdio.h>
void main()
{
    int no,i;
    long int f;
    printf("Enter a number : ");
    scanf("%d",&no);
    for( i=1; f=1; i<=no; i++ )
        f=f*i;
    printf("Factorial=%ld",f);
}
```

$f = 1$

$f = f \times i$

$1 \times 2 \times 3 \dots \times no$

$\text{fct}(i) = \{ i \leq n ; i+f \}$

$f = f \times i$

WAP to read a number. Print whether it is perfect number or not.

```
#include<stdio.h>
void main()
{
    int no,i,sum=0;
    printf("enter a number : ");
    scanf("%d",&no);
    for( i=1; i<no;i++ )
    {
        if( no%i==0 )
            sum=sum+i;
    }
    if( sum==no )
        printf("Perfect number");
    else
        printf("Not a perfect number");
}
```

$$\text{if } (\text{Sum} = n)$$

Sum = 0;  
for ( i=1; i<n; i++ )  
 $\sum \text{ if}(n \% i == 0)$   
Sum = Sum + i;

WAP to print following pattern.

```
#include<stdio.h>
void main()
{
int i,j;
for( i=1;i<=5;i++)
{
```

```
    for( j=1;j<=i;j++)
        printf("* ");
    printf("\n");
}
```

element  
print

now

1 1 1 1 1  
\*  
\* \*  
\* \* \*  
\* \* \* \*  
\* \* \* \* \*

1 1 1 1 1

1

1 2

1 2 3

WAP to print following pattern.

```
#include<stdio.h>
void main()
{
    int i,j;
    for( i=1;i<=5;i++ )
    {
        for( j=1;j<=i;j++ )
            printf("%d ",j);
        printf("\n");
    }
}
```

- |   |   |   |   |   |
|---|---|---|---|---|
| 1 |   |   |   |   |
| 1 | 2 |   |   |   |
| 1 | 2 | 3 |   |   |
| 1 | 2 | 3 | 4 |   |
| 1 | 2 | 3 | 4 | 5 |

## Do-While :-

- ❖ It is a control statement.
- ❖ It is a conditional statement.
- ❖ It is an iterative statement.
- ❖ It is used to execute the code repeated.
- ❖ It is a post test loop.
- ❖ The body will execute at least once.
- ❖ It is used when we don't know exact count of iteration.
- ❖ Syntax : do

```
{  
body;  
}while( condition );
```

do  
=   
while (condition);

WAP to find sum of elements entered by user.

```
#include<stdio.h>
void main()
{
int no,sum=0;
char ch;
do
{
printf("\nEnter a number : ");
scanf("%d",&no);
sum=sum+no;
printf("\nDo want to add one more? ");
scanf(" %c",&ch);
}while( ch!='n');

printf("Sum=%d",sum);
}
```

1  
g  
2  
2

Sum = 3

**Output :**

Enter a number : 1  
Do want to add one more? y  
Enter a number : 2  
Do want to add one more? y  
Enter a number : 3  
Do want to add one more? y  
Enter a number : 4  
Do want to add one more? y  
Enter a number : 5  
Do want to add one more? n  
Sum=15

Break :-

- ❖ It is a control statement.
- ❖ It is an un conditional statement.
- ❖ It is used to stop the loop at middle.
- ❖ It can be used only with switch, while, for, do-while.

Example :

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

```
void main()
```

```
{
```

```
int no;
```

```
for(no=1;no<=100;no++)
```

```
{
```

```
printf("%d",no);
```

```
if( no==10 )
```

```
break;
```

```
}
```

```
}
```

Output :- 1 2 3 4 5 6 7 8 9 10

Continue :-

- ❖ It is a control statement.
- ❖ It is a decision making statement.
- ❖ It is a un conditional statement.
- ❖ It is used to skip the rest of the code of current iteration.
- ❖ It can be used only with while, for, do-while.
- ❖ Example : #include <stdio.h>

**void main()**

```
{  
int no;  
for(no=1;no<=100;no++)  
{  
if( no%2==0 )  
    continue;  
printf("%d",no);  
}  
}
```

Output : 1 3 5 7 9 .... 99

Goto :-

- ❖ It is a control statement.
- ❖ It is a un conditional statement.
- ❖ It is a decision making statement.
- ❖ It is used to move the control from one location to another location.
- ❖ It move the cursor to the specified lable.
- ❖ Label is a name which always ends with ":" symbol.

Example : #include

```
void main()
{
    printf("\none");
    goto ss;
    printf("\ntwo");
    printf("\nthree");
    ss: printf("\nfour");
    printf("\nfive");
}
```

Output : none four five

none

goto

label name',

skip

label name :

There are two types of type qualifier in c

**Size qualifier:** short, long

**Sign qualifier:** signed, unsigned

When the qualifier `unsigned` is used the number is always positive, and when `signed` is used number may be positive or negative. If the sign qualifier is not mentioned, then by default sign qualifier is assumed. The range of values for signed data types is less than that of unsigned data type. Because in signed type, the left most bit is used to represent sign, while in unsigned type this bit is also used to represent the value. The size and range of the different data types on a 16 bit machine is given below:

| Basic data type | Data type with type qualifier | Size (byte) | Range                     |
|-----------------|-------------------------------|-------------|---------------------------|
| char            | char or signed char           | 1           | -128 to 127               |
|                 | Unsigned char                 | 1           | 0 to 255                  |
| int             | int or signed int             | 2           | -32768 to 32767           |
|                 | unsigned int                  | 2           | 0 to 65535                |
|                 | short int or signed short int | 1           | -128 to 127               |
|                 | unsigned short int            | 1           | 0 to 255                  |
|                 | long int or signed long int   | 4           | -2147483648 to 2147483647 |
|                 | unsigned long int             | 4           | 0 to 4294967295           |
| float           | float                         | 4           | -3.4E-38 to 3.4E+38       |
| double          | double                        | 8           | 1.7E-308 to 1.7E+308      |
|                 | Long double                   | 10          | 3.4E-4932 to 1.1E+4932    |

Two operator characteristics determine how operands group with operators: precedence and associativity.

Precedence is the priority for grouping different types of operators with their operands.

Associativity is the left-to-right or right-to-left order for grouping operands to operators that have the same precedence.

An operator's precedence is meaningful only if other operators with higher or lower precedence are present.

Expressions with higher-precedence operators are evaluated first.  
The grouping of operands can be forced by using parentheses.

$c = a + b;$   
 $\uparrow \overbrace{J}^R \text{ to L}$

$* + -$

## Precedence and associativity of operators

| Operators     | Description     | Precedence level | Associativity |
|---------------|-----------------|------------------|---------------|
| 0             | function call   | 1                | left to right |
| []            | array subscript |                  |               |
| $\rightarrow$ | arrow operator  |                  |               |
| .             | dot operator    |                  |               |
| +             | unary plus      | 2                | right to left |
| -             | unary minus     |                  |               |
| ++            | increment       |                  |               |
| --            | decrement       |                  |               |
| !             | logical not     |                  |               |
| $\sim$        | 1's complement  |                  |               |
| *             | indirection     |                  |               |
| &             | address         |                  |               |
| (data type)   | type cast       |                  |               |
| <b>sizeof</b> | size in byte    |                  |               |
| *             | multiplication  | 3                | left to right |
| /             | division        |                  |               |
| %             | modulus         |                  |               |
| +             | addition        | 4                | left to right |

|               |                       |    |               |
|---------------|-----------------------|----|---------------|
| -             | subtraction           |    |               |
| <<            | left shift            | 5  | left to right |
| >>            | right shift           |    |               |
| <=            | less than equal to    | 6  | left to right |
| >=            | greater than equal to |    |               |
| <             | less than             |    |               |
| >             | greater than          |    |               |
| ==            | equal to              | 7  | left to right |
| !=            | not equal to          |    |               |
| &             | bitwise AND           | 8  | left to right |
| ^             | bitwise XOR           | 9  | left to right |
|               | bitwise OR            | 10 | left to right |
| &&            | logical AND           | 11 |               |
|               | logical OR            | 12 |               |
| ?:            | conditional operator  | 13 |               |
| =, *=, /=, %= | assignment operator   | 14 | right to left |
| &=, ^=, <<=   |                       |    |               |
| >>=           |                       |    |               |
| ,             | comma operator        | 15 |               |

# **Programming in C**

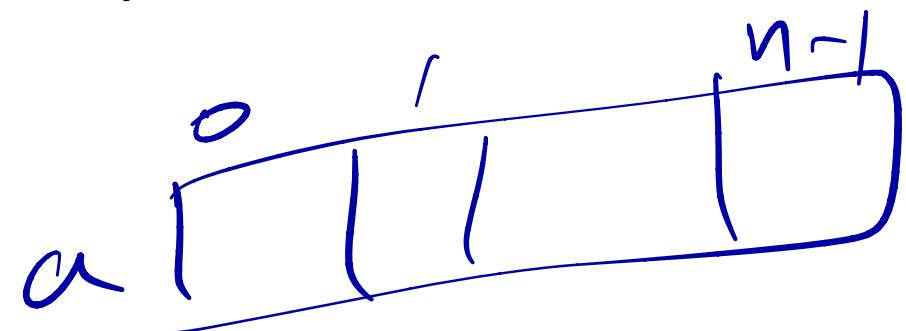
## Module 3

## Array

It is a group of elements.

- ❖ The elements should be same data type.
- ❖ The elements must be continuously in memory.
- ❖ All the elements can be identified by a common name.

It is defined as the collection of similar type of data items stored at contiguous memory locations.



Declaration / Creation of array :-

Array declaration is as same as normal variable declaration only the difference is it always ends with [size] .

Syntax :

Datatype ArrayName[ size ];

Example :

int a[100];

char ch[10];

float rate[50];

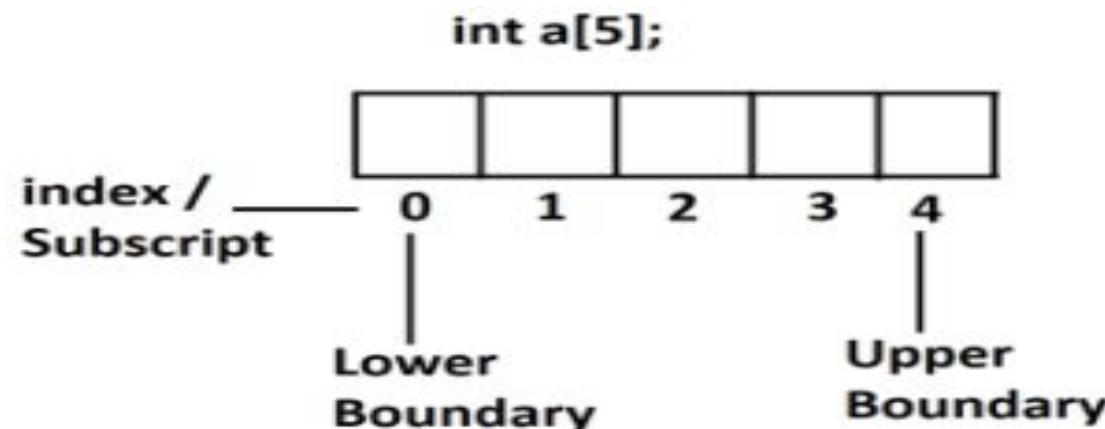
int a(100)

## Index :-

To identify the elements of an array we have to use a +ve integer it is called index.

It is also called as sub script.

It always begins with 0. It is called lower boundary. It always ends with size-1. It is called upper boundary.



### **Accessing Array elements :**

**We can access the array elements with the help of arrayName and index.**

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

|   |    |    |    |    |    |
|---|----|----|----|----|----|
| a | 10 | 20 | 30 | 40 | 50 |
|   | 0  | 1  | 2  | 3  | 4  |

**syntax :**

**arrayName[index]**

**Examples**

**a[0]=10**

**a[2]=30**

**a[4]=50**

**a[10]=NA**

Storing array elements :-

We can store array elements in 3 ways.

Way 1 :-

We can store array elements by initialization process.

Example :

```
int a[5]={10,20,30,40,50};
```

|   |    |    |    |    |    |
|---|----|----|----|----|----|
| a | 10 | 20 | 30 | 40 | 50 |
|   | 0  | 1  | 2  | 3  | 4  |

Way2 :-

We can store array elements by assigning individual elements.

```
int a[5];
a[0]=10;
a[2]=20;
a[4]=50;
```

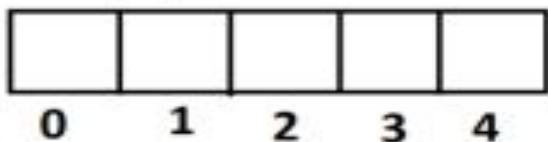
|   |    |   |    |   |    |
|---|----|---|----|---|----|
| a | 10 |   | 20 |   | 50 |
|   | 0  | 1 | 2  | 3 | 4  |

### Way3 :-

We can store array elements using input operation :

Syntax:

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



```
printf("Enter %d elements",size );
for( i=0;i<size;i++ )
scanf("%d",&arrayName[i])
```

### Printing array elements :-

```
printf("Array is")
for( i=0;i<size;i++)
printf("%5d",arrayName[i])
```

WAP to read N elements into an array. Print the same on screen.

```
#include<stdio.h>
void main()
{
    int a[100],n,i;
    //Read array size
    printf("Enter array size : ");
    scanf("%d",&n);

    //Read elements into array.
    printf("Enter %d elements",n);
    for( i=0;i<n;i++ )
        scanf("%d",&a[i]);

    //print elements of array.
    printf("Array is\n");
    for( i=0;i<n;i++ )
        printf("%d",a[i]);
}
```

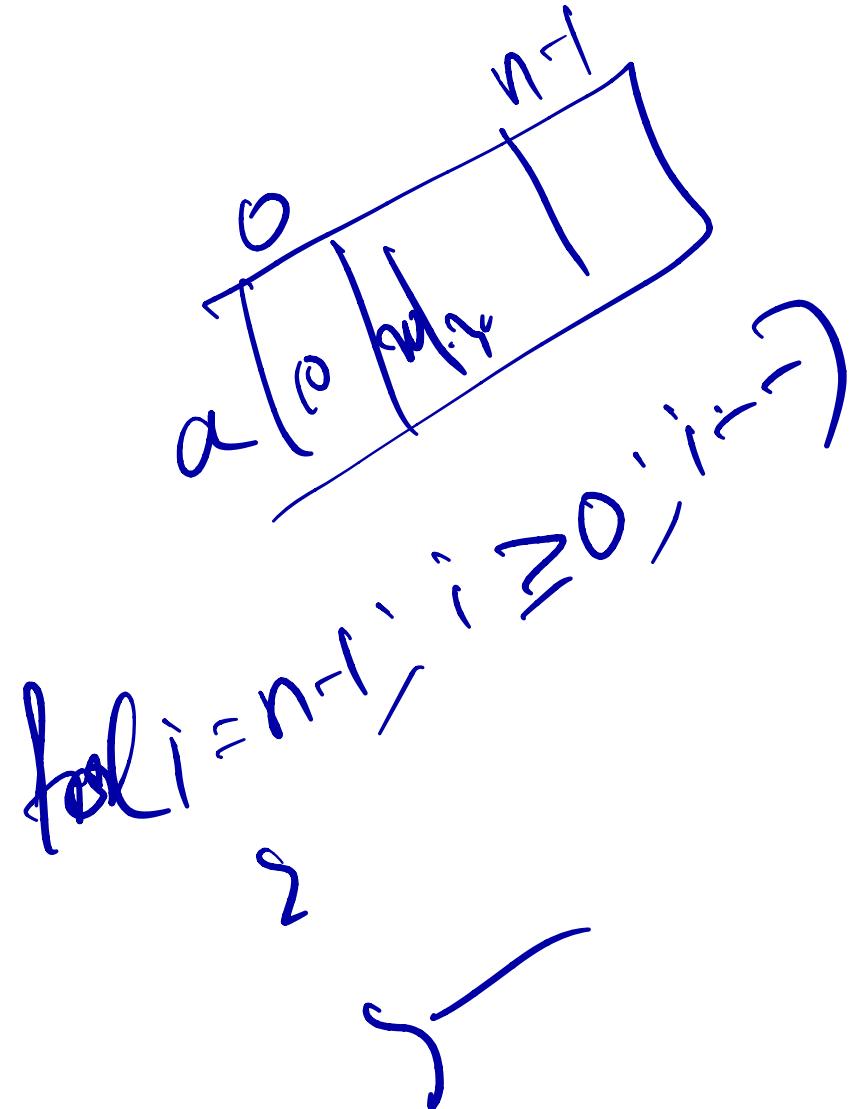
int a[100];

a

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

WAP to read n elements into an array. Print them in reverse.

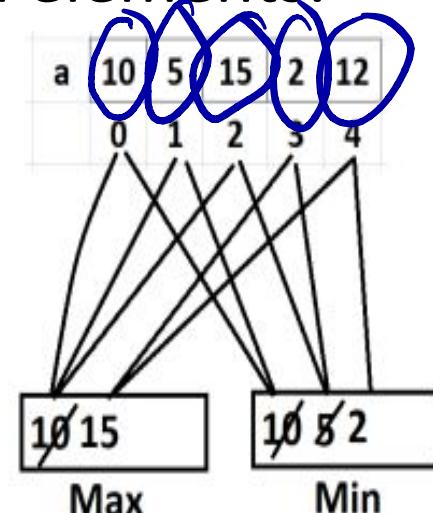
```
#include<stdio.h>
void main()
{
    int a[100],n,i;
    // read array size.
    printf("Enter array size : ");
    scanf("%d",&n);
    // read elements into array.
    printf("Enter %d elements",n);
    for( i=0;i<n;i++ )
        scanf("%d",&a[i]);
    // print elements in reverse.
    printf("Array elements in reverse \n");
    for( i=n-1;i>=0;i-- )
        printf("%5d",a[i]);
}
```



WAP to read n elements into an array. Find max and min elements.

```
#include<stdio.h>
void main()
{
    int a[100],n,i,max,min;
    //read array size.
    printf("Enter array size : ");
    scanf("%d",&n);
    // Read array elements
    printf("Enter %d elements",n);
    for( i=0;i<n;i++)
        scanf("%d",&a[i]);

    //find max and min
    max=min=a[0];
    for( i=0;i<n;i++)
    {
        if( a[i] > max)
            max=a[i];
        if( a[i] < min )
            min=a[i];
    }
    printf("Max=%d",max);
    printf("\nMin=%d",min);
}
```



10 5 15 2 12

10 5 Max

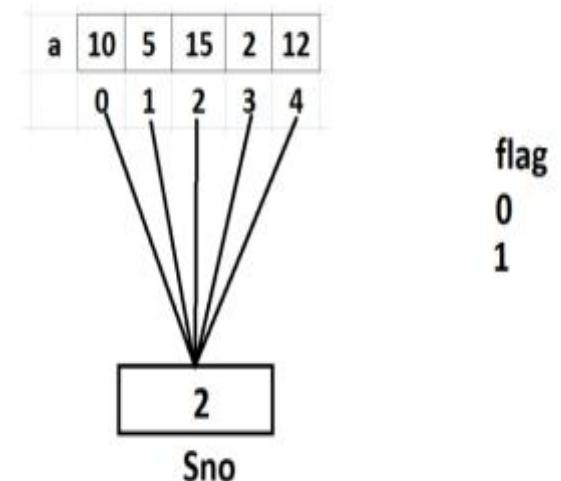
2 Min

WAP to read n elements into an array, search elements. Print whether the search element is in array or not.

```
#include<stdio.h>
void main()
{
    int a[100],n,i,sno,flag=0;
    // read array size
    printf("Enter array size : ");
    scanf("%d",&n);
    // read array elements
    printf("Enter %d elements",n);
    for( i=0;i<n;i++ )
        scanf("%d",&a[i] );
    // read search number
    printf("Enter search number : ");
    scanf("%d",&sno);

    // Search operation
    for( i=0;i<n;i++ )
    {
        if( a[i]==sno )
            flag=1;
    }
    if( flag==1 )
        printf("Element found");
    else
        printf("Element not found");
}
```

*Search Element*



## 2-D Arrays

It is a group of one dimensional arrays of same size and same data type.

int a [1 2 3] size=3

0 1 2

int b [4 5 6] size=3

0 1 2

int c [7 8 9] size=3

0 1 2

|   |   |   |   |   |
|---|---|---|---|---|
| a | 0 | 1 | 2 | 3 |
|   | 1 | 4 | 5 | 6 |
|   | 2 | 7 | 8 | 9 |

3x3

Declaration / creation of 2-D array.

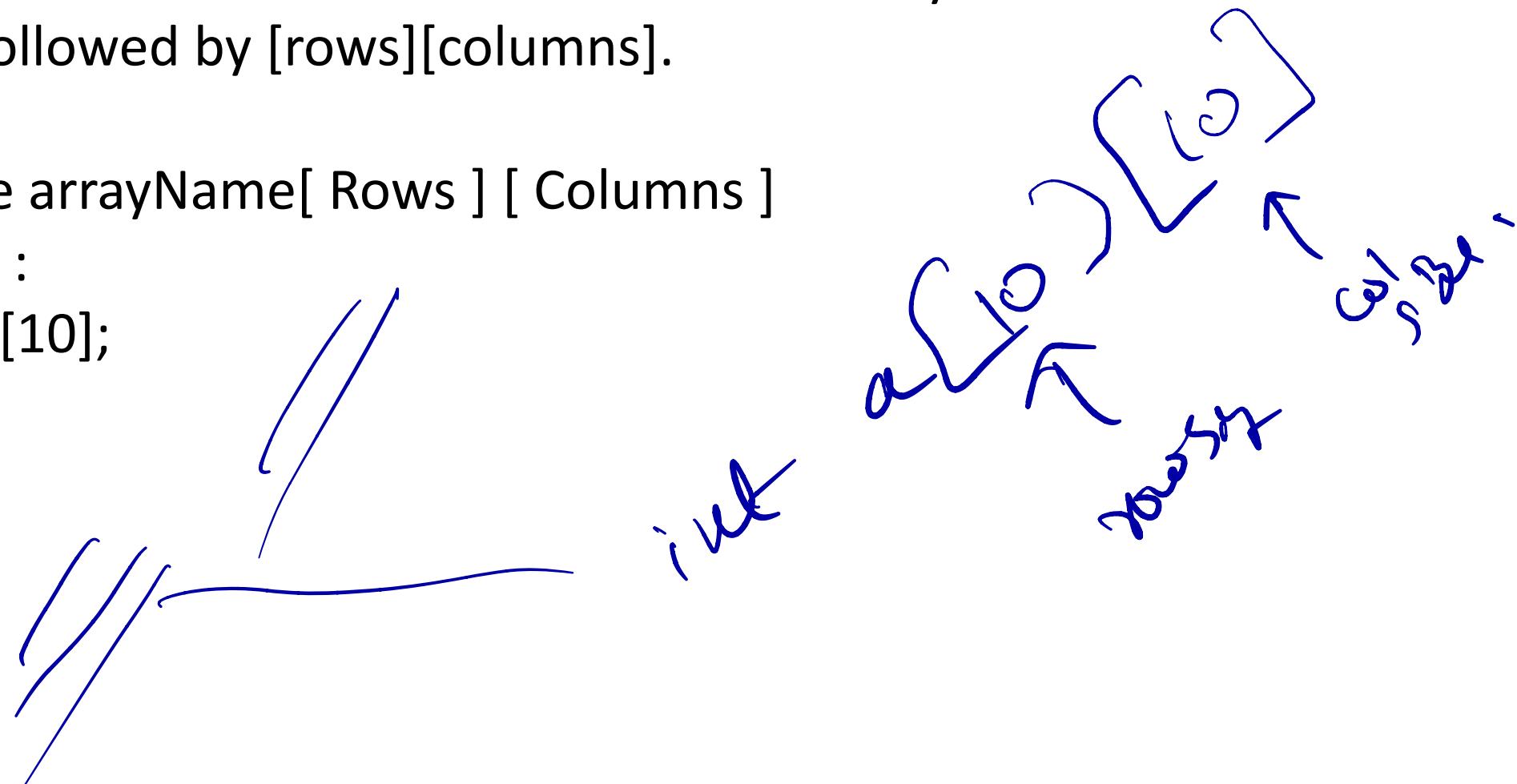
It is as same as normal variable declaration only the difference is it always followed by [rows][columns].

Syntax :

Datatype arrayName[ Rows ] [ Columns ]

Example :

int a[10][10];



### Accessing Elements of 2-D array.

Array elements can access using array Name, Row Index, Column Index.

|     |   | 0 | 1 | 2 |
|-----|---|---|---|---|
| a   | 0 | 1 | 2 | 3 |
|     | 1 | 4 | 5 | 6 |
|     | 2 | 7 | 8 | 9 |
| 3x3 |   |   |   |   |

Syntax:

ArrayName[Row Index][Column Index]

Examples

`a[0][0]=1`

`a[0][2]=3`

`a[1][1]=5`

`a[2][2]=9`

### Storing array elements :-

We can store array elements in 3 ways.

1. We store using initialization process.

|   |   | 0  | 1  | 2  |     |
|---|---|----|----|----|-----|
| a | 0 | 10 | 20 | 30 |     |
|   | 1 | 40 | 50 | 60 |     |
|   | 2 | 70 | 80 | 90 |     |
|   |   |    |    |    | 3x3 |

Syntax:

**datatype arrayName[rows][cols]={values separated by comma}**

Examples

```
int a[3][3]={10,20,30,40,50,60,70,80,90}
```

2. We can store by assignment of individual elements.

Example

|   |   | 0 | 1 | 2  |
|---|---|---|---|----|
| a | 0 | 5 |   | 25 |
|   | 1 |   | 7 |    |
|   | 2 |   |   | 22 |

3x3

Syntax:

**arrayName[row Index][col index]=value;**

Examples

**a[0][0]=5;**

**a[0][2]=25**

**a[1][1]=7**

**a[2][2]=22**

3.We can read array elements using for loop and scanf()

```
for( i=0;i<rows;i++)  
for( j=0;j<cols;j++)  
scanf("%d",&Arrayname[ i ][ j ] );
```

### Print array elements :-

We can print array elements using for loop and printf()

```
printf("array is \n");  
for( i=0;i<rows;i++ )  
{  
for( j=0;j<cols;j++ )  
printf("%d",arrayname[ i ][ j ] );  
printf("\n");  
}
```

1. WAP to read  $m \times n$  elements into an arrays. Print it on screen.

WAP to read  $m \times n$  elements into an array, find the transpose matrix.

## **Question No. 6**

Write a C program to display the number of occurrences of a given element in an array.

## **Program**

```
#include<stdio.h>
int main()
{
    int a[10],n,i,k,count;
    printf("Enter the size of the array\n");
    scanf("%d",&n);
    printf("Enter the elements\n");
    for(i=0;i<n;i++)
    {
        scanf("%d",&a[i]);
    }
```

```
printf("Enter the element to be searched\n");
scanf("%d",&k);
count=0;
for(i=0;i<n;i++)
{
    if(a[i]==k)
    {
        count++;
    }
}
```

```
printf("Number    of    occurrences    of    the  
element=%d\n",count);  
return 0;  
}
```

## **Output**

Enter the size of the array

5

Enter the elements

10

20

54

20

20

Enter the element to be searched

20

Number of occurrences of the element=3

## Question No. 7

Write a C program to find transpose of a matrix,

### Program

```
#include<stdio.h>
int main()
{
    int a[10][10],t[10][10],i,j,m,n;
    printf("Enter the number of rows and
           columns of matrix\n");
    scanf("%d%d",&m,&n);
```

```
printf("Enter the elements\n");
for(i=0;i<m;i++)
{
    for(j=0;j<n;j++)
    {
        scanf("%d",&a[i][j]);
    }
}
```

now size  
on size

```
printf("Matrix\n");
for(i=0;i<m;i++)
{
    for(j=0;j<n;j++)
    {
        printf("%d\t",a[i][j]);
    }
    printf("\n");
}
```

```
printf("Transpose Matrix\n");
for(i=0;i<m;i++)
{
    for(j=0;j<n;j++)
    {
        t[j][i]=a[i][j];
    }
}
```

↓  
t[i][j]

= a[i][j]

```
for(i=0;i<n;i++)
{
    for(j=0;j<m;j++)
    {
        printf("%d\t",t[i][j]);
    }
    printf("\n");
}
return 0;
}
```

CCCGC

**Output**

Enter the number of rows and columns of matrix

2

2

Enter the elements

4

5

8

10

**Matrix**

4 5

8 10

**Transpose Matrix**

4 8

5 10

## Question No. 8

Write a C program to print row sum, column sum  
and diagonal sum of a matrix.

### Program

```
#include<stdio.h>
int main()
{
    int a[10][10],i,j,m,n,rsum,csum,dsum;
    printf("Enter the number of rows and
           columns of matrix\n");
    scanf("%d%d",&m,&n);
```

```
printf("Enter the elements\n");
for(i=0;i<m;i++)
{
    for(j=0;j<n;j++)
    {
        scanf("%d",&a[i][j]);
    }
}
```

```
printf("Matrix:\n");
for(i=0;i<m;i++)
{
    for(j=0;j<n;j++)
    {
        printf("%d\t",a[i][j]);
    }
    printf("\n");
}
```

29w

```
for(i=0;i<m;i++)  
{  
    rsum=0;  
    for(j=0;j<n;j++)  
    {  
        rsum=rsum+a[i][j];  
    }  
    printf("Sum of elements of %d row =  
          %d\n",i,rsum);  
}
```

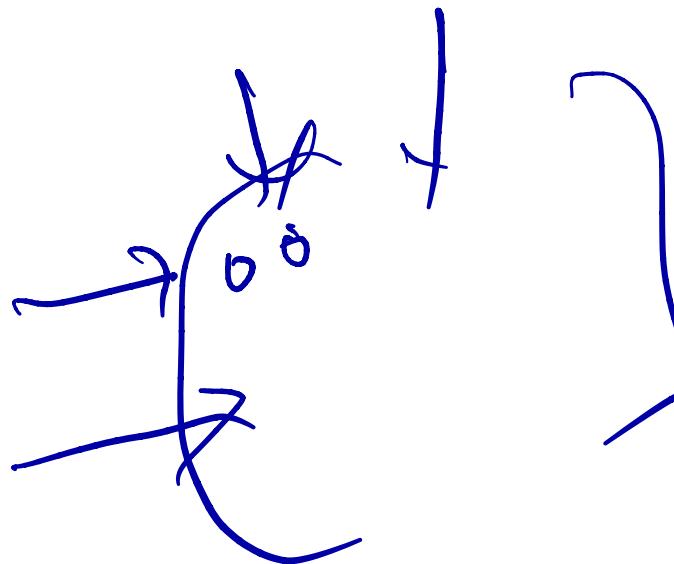
```
for(i=0;i<n;i++)
{
    csum=0;
    for(j=0;j<m;j++)
    {
        csum=csum+a[j][i];
    }
    printf("Sum of elements of %d column =
          %d\n",i,csum);
}
```

col sum

row sum

alias C

```
dsum=0;  
for(i=0;i<m;i++)  
{  
    for(j=0;j<n;j++)  
    {  
        if(i==j)  
        {  
            dsum=dsum+a[i][j];  
        }  
    }  
}
```



```
printf("Sum of diagonal elements = %d\n",
      dsum);

return 0;

}
```

### **Output 1**

Enter the number of rows and columns of matrix

3

3

**Enter the elements**

**2**

**4**

**6**

**8**

**10**

**12**

**14**

**16**

**18**

**Matrix:**

|    |    |    |
|----|----|----|
| 2  | 4  | 6  |
| 8  | 10 | 12 |
| 14 | 16 | 18 |

**Sum of elements of 0 row = 12**

**Sum of elements of 1 row = 30**

**Sum of elements of 2 row = 48**

**Sum of elements of 0 column = 24**

**Sum of elements of 1 column = 30**

**Sum of elements of 2 column = 36**

**Sum of diagonal elements = 30**

## ***Output 2***

Enter the number of rows and columns

2

3

Enter the elements

1

2

3

4

5

6

**Matrix:**

1      2      3

4      5      6

**Sum of elements of 0 row = 6**

**Sum of elements of 1 row = 15**

**Sum of elements of 0 column = 5**

**Sum of elements of 1 column = 7**

**Sum of elements of 2 column = 9**

**Sum of diagonal elements=6**

## Question No. 9

Write a C program to find product of two matrices.

### Product of two matrices

- *The product of two matrices can be computed by multiplying elements of the first row of the first matrix with the first column of the second matrix then, add all the product of elements.*
- *Continue this process until each row of the first matrix is multiplied with each column of the second matrix.*

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \quad B = \begin{pmatrix} 5 & 6 & 7 \\ 8 & 9 & 10 \end{pmatrix}$$

**2x2**

**2x3**

Multiplication of two matrixes:

$$A * B = \begin{pmatrix} 1*5 + 2*8 & 1*6 + 2*9 & 1*7 + 2*10 \\ 3*5 + 4*8 & 3*6 + 4*9 & 3*7 + 4*10 \end{pmatrix}$$

$$A * B = \begin{pmatrix} 21 & 24 & 27 \\ 47 & 54 & 61 \end{pmatrix}$$

**2x3**

**m × n**  
**p × q**  
**n = p**

**m**  
**n**

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \quad B = \begin{pmatrix} 5 & 6 & 7 \\ 8 & 9 & 10 \end{pmatrix}$$

**$m1 \times n1$**        **$m2 \times n2$**

Multiplication of two matrixes:

$$A * B = \begin{pmatrix} 1*5 + 2*8 & 1*6 + 2*9 & 1*7 + 2*10 \\ 3*5 + 4*8 & 3*6 + 4*9 & 3*7 + 4*10 \end{pmatrix}$$

$$A * B = \begin{pmatrix} 21 & 24 & 27 \\ 47 & 54 & 61 \end{pmatrix}$$

**$m1 \times n2$**

## Program

```
#include<stdio.h>
int main()
{
    int a[10][10], b[10][10], p[10][10], m1, n1,
        m2, n2, i, j, k;
    printf("Enter the number of rows and
           columns of first matrix\n");
    scanf("%d%d",&m1,&n1);
```

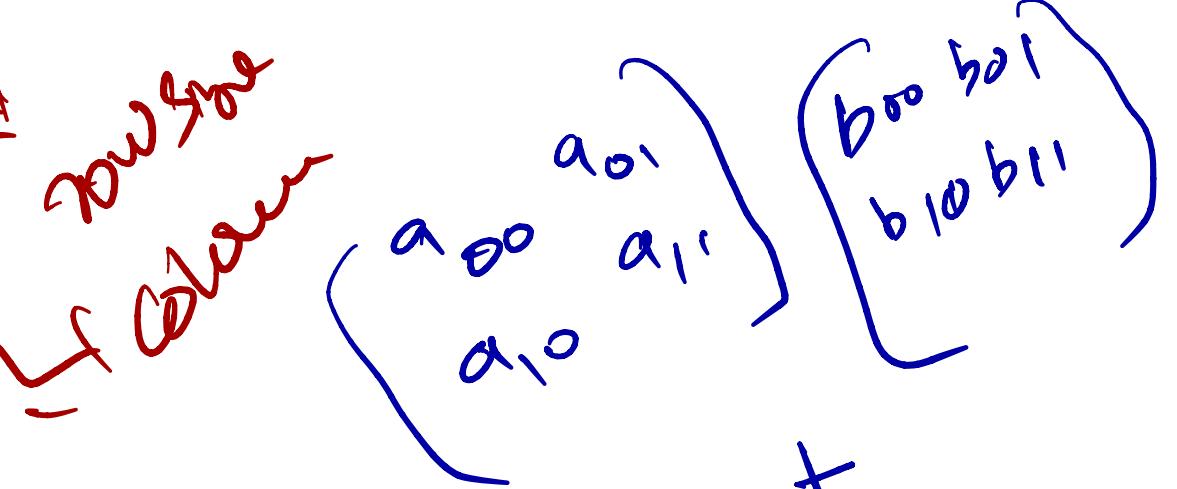
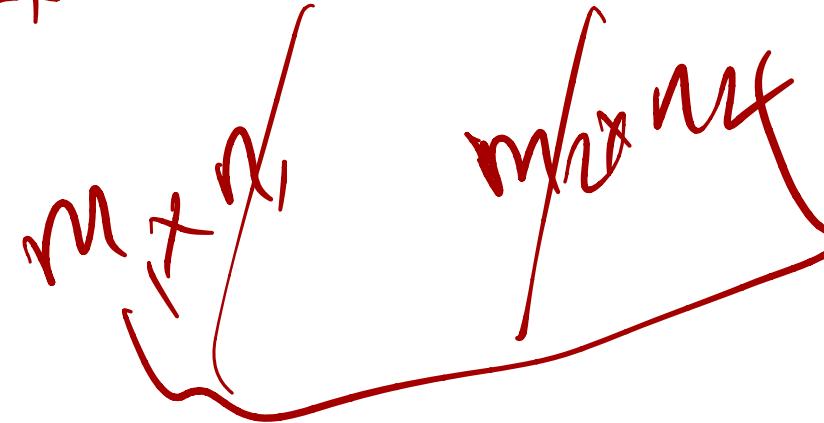
```
printf("Enter the elements\n");
for(i=0;i<m1;i++)
{
    for(j=0;j<n1;j++)
    {
        scanf("%d",&a[i][j]);
    }
}
```

```
printf("First Matrix:\n");  
for(i=0;i<m1;i++)  
{  
    for(j=0;j<n1;j++)  
    {  
        printf("%d\t",a[i][j]);  
    }  
    printf("\n");  
}
```

```
printf("Enter the number of rows and  
columns of second matrix\n");  
scanf("%d%d",&m2,&n2);  
printf("Enter the elements\n");  
for(i=0;i<m2;i++)  
{  
    for(j=0;j<n2;j++)  
    {  
        scanf("%d",&b[i][j]);  
    }  
}
```

```
printf("Second Matrix:\n");
for(i=0;i<m2;i++)
{
    for(j=0;j<n2;j++)
    {
        printf("%d\t",b[i][j]);
    }
    printf("\n");
}
```

```
for(i=0;i<m1;i++)  
{  
    for(j=0;j<n2;j++)  
    {  
        p[i][j]=0;  
        for(k=0;k<m2;k++)  
        {  
            p[i][j]=p[i][j]+(a[i][k]*b[k][j]);  
        }  
    }  
}
```



```
printf("Product:\n");
for(i=0;i<m1;i++)
{
    for(j=0;j<n2;j++)
    {
        printf("%d\t",p[i][j]);
    }
    printf("\n");
}
return 0;
}
```

## ***Output***

Enter the number of rows and columns of first matrix

2

2

Enter the elements

1

2

3

4

First Matrix:

1 2

3 4

**Enter the number of rows and columns of second matrix**

**2**

**3**

**Enter the elements**

**5**

**6**

**7**

**8**

**9**

**10**

**Second Matrix:**

**5      6      7**

**8      9      10**

**Product:**

**21      24      27**

**47      54      61**

## Question No. 10

Write a C program to find sum of two matrices.

## Add Two Matrices

Matrix 1

|   |   |   |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | 9 |

+

Matrix 2

|   |   |   |
|---|---|---|
| 0 | 2 | 4 |
| 4 | 6 | 8 |
| 6 | 8 | 9 |

Added Matrix

|    |    |    |
|----|----|----|
| 1  | 4  | 7  |
| 8  | 11 | 14 |
| 13 | 16 | 18 |

*a[i][j]*

*a[0][1]=2*

*Sum[0][1]= a[0][1]+b[0][1]=2+2=4*

*b[i][j]*

*b[0][1]=2*

*Sum[i][j]*

## ***Program***

```
#include<stdio.h>
int main()
{
    int a[10][10], b[10][10], Sum[10][10], m1,
        n1, m2,n2,i,j;
    printf("Enter number of rows and columns
           of first matrix\n");
    scanf("%d%d",&m1,&n1);
```

```
printf("Enter the elements\n");
for(i=0;i<m1;i++)
{
    for(j=0;j<n1;j++)
    {
        scanf("%d",&a[i][j]);
    }
}
```

```
printf("First Matrix:\n");
for(i=0;i<m1;i++)
{
    for(j=0;j<n1;j++)
    {
        printf("%d\t",a[i][j]);
    }
    printf("\n");
}
printf("Enter number of rows and columns
       of second matrix\n");
scanf("%d%d",&m2,&n2);
```

```
printf("Enter the elements\n");
for(i=0;i<m2;i++)
{
    for(j=0;j<n2;j++)
    {
        scanf("%d",&b[i][j]);
    }
}
```

```
printf("Second Matrix:\n");
for(i=0;i<m2;i++)
{
    for(j=0;j<n2;j++)
    {
        printf("%d\t",b[i][j]);
    }
    printf("\n");
}
```

```
if((m1==m2) && (n1==n2))
{
    printf("Sum of two matrices:\n");
    for(i=0;i<m1;i++)
    {
        for(j=0;j<n1;j++)
        {
            Sum[i][j]=a[i][j]+b[i][j];
        }
    }
}
```

```
for(i=0;i<m1;i++)
{
    for(j=0;j<n1;j++)
    {
        printf("%d\t",Sum[i][j]);
    }
    printf("\n");
}
}
```

```
else
{
    printf("Addition is not possible\n");
}
return 0;
}
```

## **Output 1**

Enter number of rows and columns of first matrix

2

2

Enter the elements

1

2

3

4

First Matrix:

1 2

3 4

**Enter number of rows and columns of second matrix**

**2**

**2**

**Enter the elements**

**5**

**6**

**0**

**1**

**Second Matrix:**

**5      6**

**0      1**

**Sum of two matrices:**

$$\begin{matrix} 6 & 8 \\ 3 & 5 \end{matrix}$$

## **Output 2**

Enter number of rows and columns of first matrix

2

2

Enter the elements

1

0

3

8

First Matrix:

1 0

3 8

**Enter number of rows and columns of second matrix**

**2**

**3**

**Enter the elements**

**10**

**5**

**0**

**6**

**8**

**2**

**Second Matrix:**

**10    5    0**

**6    8    2**

**Addition is not possible**

# Properties of Arrays

- Each element of an array is of same data type and carries the same size.
- Index value always starts with zero.
- Elements of the array are stored at contiguous memory locations where the first element is stored at the smallest memory locations.
- Elements of the array can be randomly accessed.

# Advantages of Arrays

- Random access of elements using array index.
- Use of less line of code as it creates a single array of multiple elements.
- Easy access to all the elements.
- Traversal through the array becomes easy using a single loop.
- Sorting becomes easy as it can be accomplished by writing less line of code.

# Disadvantages of Arrays

- 1) It allows us to enter only fixed number of elements into it. We cannot alter the size of the array once array is declared. Hence if we need to insert more number of records than declared then it is not possible. We should know array size at the compile time itself.
- 2) Inserting and deleting the records from the array would be costly since we add / delete the elements from the array, we need to manage memory space too.
- 3) It does not verify the indexes while compiling the array. In case there is any indexes pointed which is more than the dimension specified, then we will get run time errors rather than identifying them at compile time.

# String Processing

# String

- Strings are defined as an array of characters.
- The difference between a character array and a string is the string is terminated with a special character '\0'.

| Index           | 0   | 1   | 2   | 3   | 4   | 5    |
|-----------------|-----|-----|-----|-----|-----|------|
| Character Array | 'H' | 'E' | 'L' | 'L' | 'O' |      |
| String          | 'H' | 'E' | 'L' | 'L' | 'O' | '\0' |

# Declaration of Strings

- Basic syntax for declaring a string.

**char variable-name[size];**

- **variable-name** is any name given to the string variable and **size** is used define the length of the string, i.e. the number of characters strings will store.
- There is an extra terminating character which is the Null character ('\0') used to indicate termination of string which differs strings from normal character arrays.

# Initializing a String



A string can be initialized in different ways:

1. `char str[] = "Programming";`
2. `char str[50] = "Programming";`
3. `char str[] = {'P','r','o','g','r','a','m','m','i','n','g','\0'};`
4. `char str[12] = {'P','r','o','g','r','a','m','m','i','n','g','\0'};`

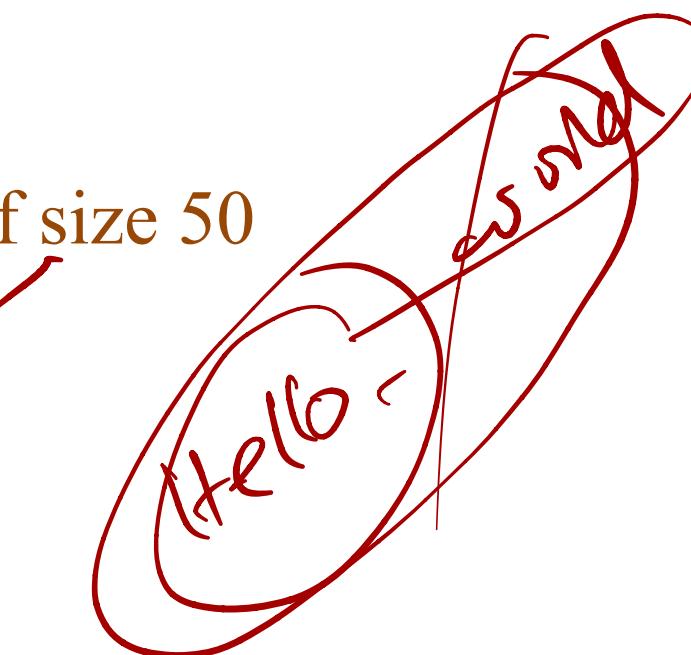
|    |
|----|
| P  |
| r  |
| o  |
| g  |
| r  |
| a  |
| m  |
| m  |
| i  |
| n  |
| g  |
| \0 |

## **Question No. 1**

Write a C program which reads and display a string.

## Method 1

```
#include<stdio.h>
int main()
{
    char a[50]; //declare character array of size 50
    //read the string
    printf("Enter the string\n");
    scanf("%s",a);
    //display the string
    printf("The string entered is:%s\n",a);
    return 0;
}
```



## ***Method 2***

```
#include<stdio.h>
int main()
{
    char a[50]; //declare character array of size 50
    int n,i;
    //read the string
    printf("Enter the string\n");
    gets(a);
    //display the string
    puts(a);
    return 0;
}
```

# String Handling Functions

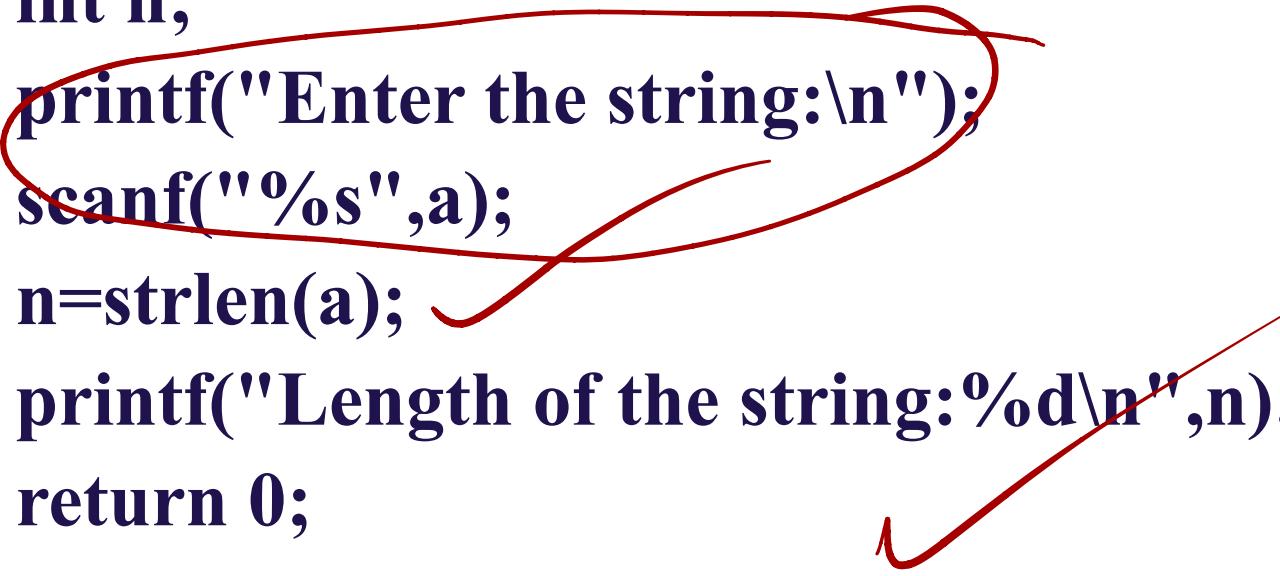
|        |                                              |               |                    |
|--------|----------------------------------------------|---------------|--------------------|
| strlen | Finds out the length of a string             | strlen(s)     | #include<string.h> |
| strcat | It appends one string at the end of another. | strcat(s1,s2) |                    |
| strcpy | Use it for copying a string into another.    | strcpy(s1,s2) |                    |
| strcmp | It compares two strings                      | strcmp(s1,s2) |                    |
| strrev | It reverses a string                         | strrev(s)     |                    |

## Question No. 2

Write a C program to find length of a string using string handling function.

## Program

```
#include<stdio.h>
#include<string.h>
int main()
{
    char a[50];
    int n;
    printf("Enter the string:\n");
    scanf("%s",a);
    n=strlen(a);
    printf("Length of the string:%d\n",n);
    return 0;
}
```



### **Question No. 3**

Write a C program to concatenate two strings using string handling function.

## Program

```
#include<stdio.h>
#include<string.h>
int main()
{
    char a[50],b[50];
    printf("Enter first string:\n");
    scanf("%s",a);
    printf("Enter second string:\n");
    scanf("%s",b);
    strcat(a,b);
    printf("Concatenated string:%s",a);
    return 0;
}
```

a = Hello  
b = world

a, b

Hello world

### **Question No. 4**

Write a C program to copy content of one string into another using string handling function.

## Program

```
#include<stdio.h>
#include<string.h>
int main()
{
    char a[50],b[50];
    printf("Enter first string:\n");
    scanf("%s",a);
    printf("Enter second string:\n");
    scanf("%s",b);
    printf("Before copying:\na: %s\nb: %s\n",a,b);
    strcpy(a,b);
    printf("After copying:\na: %s\nb: %s\n",a,b);
    return 0;
}
```

a ← Hello  
b ← World

a ← world  
b ← usslp

## **Output**

**Enter first string:**

**hello**

**Enter second string:**

**world**

**Before copying:**

**a: hello**

**b: world**

**After copying:**

**a: world**

**b: world**

### **Question No. 5**

Write a C program to compare two strings using string handling function.

## Program

```
#include<stdio.h>
#include<string.h>
int main()
{
    char a[50],b[50];
    printf("Enter first string:\n");
    scanf("%s",a);
    printf("Enter second string:\n");
    scanf("%s",b);
    if(strcmp(a,b)==0)
    {
        printf("Same\n");
    }
}
```

a Hai  
b Hai  
y

Same

```
else
{
    printf("Different\n");
}
return 0;
}
```

## **Output 1**

**Enter first string:**

**hi**

**Enter second string:**

**hi**

**Same**

## **Output 2**

**Enter first string:**

**good**

**Enter second string:**

**Good**

**Different**

## **Question No. 6**

Write a C program to print reverse of a string using string handling function.

## Program

```
#include<stdio.h>
#include<string.h>
int main()
{
    char a[10];
    printf("Enter a string\n");
    scanf("%s",a);
    strrev(a);
    printf("Reverse: %s\n",a);
    return 0;
}
```

## **Output**

Enter a string

hello

Reverse: olleh

### Question No. 7

Write a C program to check whether the given string is palindrome or not **using string handling function.**

---

## Program

```
#include<stdio.h>
#include<string.h>
int main()
{
    char a[10],b[10];
    printf("Enter a string\n");
    scanf("%s",a);
    strcpy(b,a);
    strrev(a);
```



```
if(strcmp(b,a)==0)
{
    printf("Palindrome\n");
}
else
{
    printf("Not Palindrome\n");
}
return 0;
}
```

## **Output 1**

**Enter a string**

**malayalam**

**Palindrome**

## **Output 2**

**Enter a string**

**good**

**Not Palindrome**

### **Question No. 8**

Write a C program to count number of vowels in a given string.

## Program

```
#include<stdio.h>
int main()
{
    char a[10];
    int i,count;
    printf("Enter a string\n");
    scanf("%s",a);
    count=0;
```

```
for(i=0;a[i]!='\0';i++)
{
    if(a[i]=='a'||a[i]=='e'||a[i]=='i'||a[i]=='o'||a[i]=='u'||  

        a[i]=='A'||a[i]=='E'||a[i]=='I'||a[i]=='O'||a[i]=='U')
    {
        count++;
    }
}
printf("Number of vowels=%d",count);
return 0;
}
```

## **Output**

Enter a string

education

Number of vowels=5

## Question No. 9

Write a C program to count number of spaces in a given string.

$s \in \mathbb{P}$

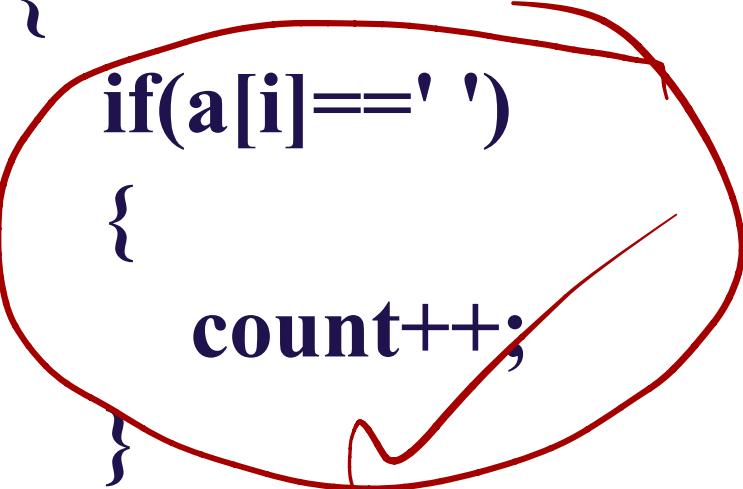
$ch \in \mathbb{C}$

$s[xt])$

## Program

```
#include<stdio.h>
int main()
{
    char a[10];
    int i,count;
    printf("Enter a string\n");
    gets(a);
    count=0;
```

```
for(i=0;a[i]!='\0';i++)
{
    if(a[i]==' ')
    {
        count++;
    }
}
printf("Number of spaces=%d",count);
return 0;
}
```



## **Output**

Enter a string

Hi Good morning

Number of spaces=2

**Question No. 10**

Write a C program to count number of ~~consonants~~ ~~vowels~~ in a given string.

## Program

```
#include<stdio.h>
int main()
{
    char a[10];
    int i,count;
    printf("Enter a string\n");
    gets(a);
    count=0;
```

```
for(i=0;a[i]!='\0';i++)
{
    if((a[i]>='a' && a[i]<='z')||(a[i]>='A' &&
        a[i]<='Z'))
    {
        count++;
    }
}
printf("Number of consonants=%d",count);
return 0;
}
```

*alphabets*

## **Output**

**Enter a string**

**Good morning**

**Number of consonants=11**

### Question No. 11

Write a C program to count number of vowels, consonants, white spaces, digits and special characters in a given string.

## Program

```
#include<stdio.h>
int main()
{
    char a[50];
    int i,v,s,sc,d,c;
    printf("Enter a string\n");
    gets(a);
    v=0;
    s=0;
    sc=0;
    d=0;
    c=0;
```

```
for(i=0;a[i]!='\0';i++)
{
    if(a[i]=='a'||a[i]=='e'||a[i]=='i'||a[i]=='o'||a[i]=='u'||  

        a[i]=='A'||a[i]=='E'||a[i]=='I'||a[i]=='O'||a[i]=='U')
    {
        v++;
    }
    else if(a[i]==' ')
    {
        s++;
    }
}
```

vowel  
space  
a-a  
A-A  
constant

```
else if((a[i]>='a' && a[i]<='z')||(a[i]>='A' &&  
       a[i]<='Z'))  
{  
    c++;  
}  
else if(a[i]>='0' && a[i]<='9')  
{  
    d++;  
}  
else  
{  
    sc++;  
}  
}
```

no st const

speed charactor

```
printf("Number of vowels=%d\n",v);
printf("Number of spaces=%d\n",s);
printf("Number of consonants=%d\n",c);
printf("Number of digits=%d\n",d);
printf("Number of special characters=%d\n",sc);
return 0;
}
```

## **Output**

**Enter a string**

**Hi, I am Nayana.Date: 02/09/2021**

**Number of vowels=8**

**Number of spaces=4**

**Number of consonants=7**

**Number of digits=8**

**Number of special characters=5**

## Question No. 12

Write a C program to find length of a string without using string handling function.



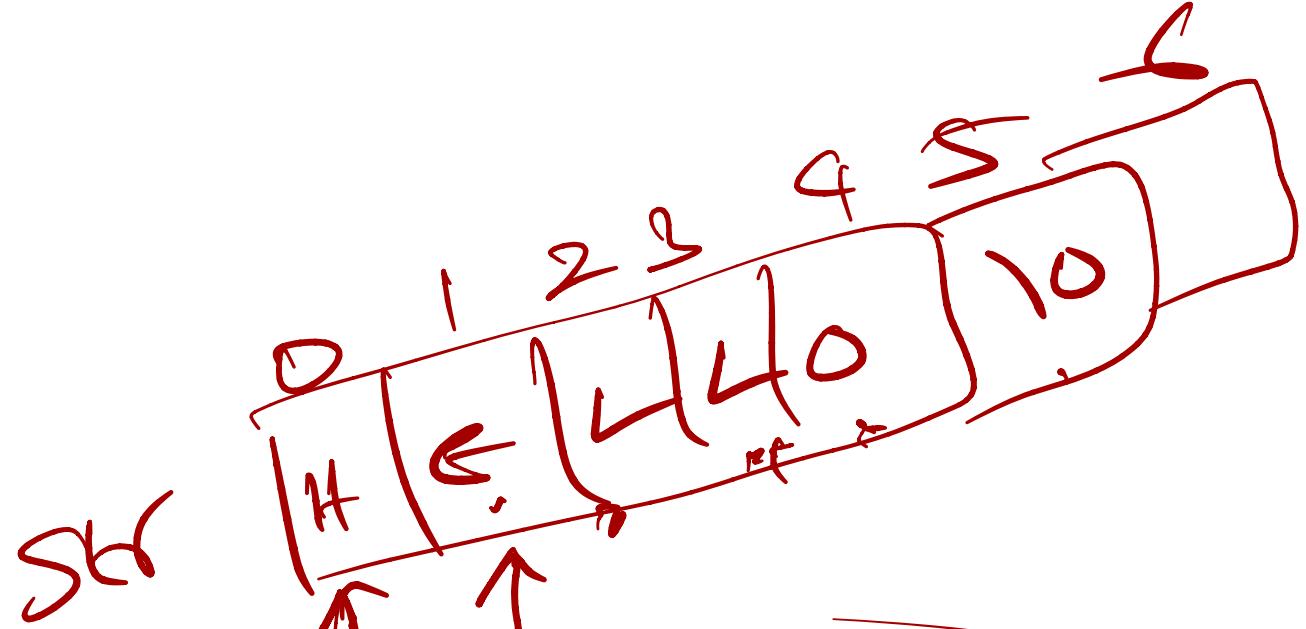
## Program

```
#include<stdio.h>
void main()
{
char str[100];
int len;
// read the string
printf("Enter a string : ");
scanf("%s",&str);
//find the length
for( len=0;str[len]!='\0';len++);
printf("Length=%d",len);
}
```

**Output :**

Enter a string : ramana

Length=6

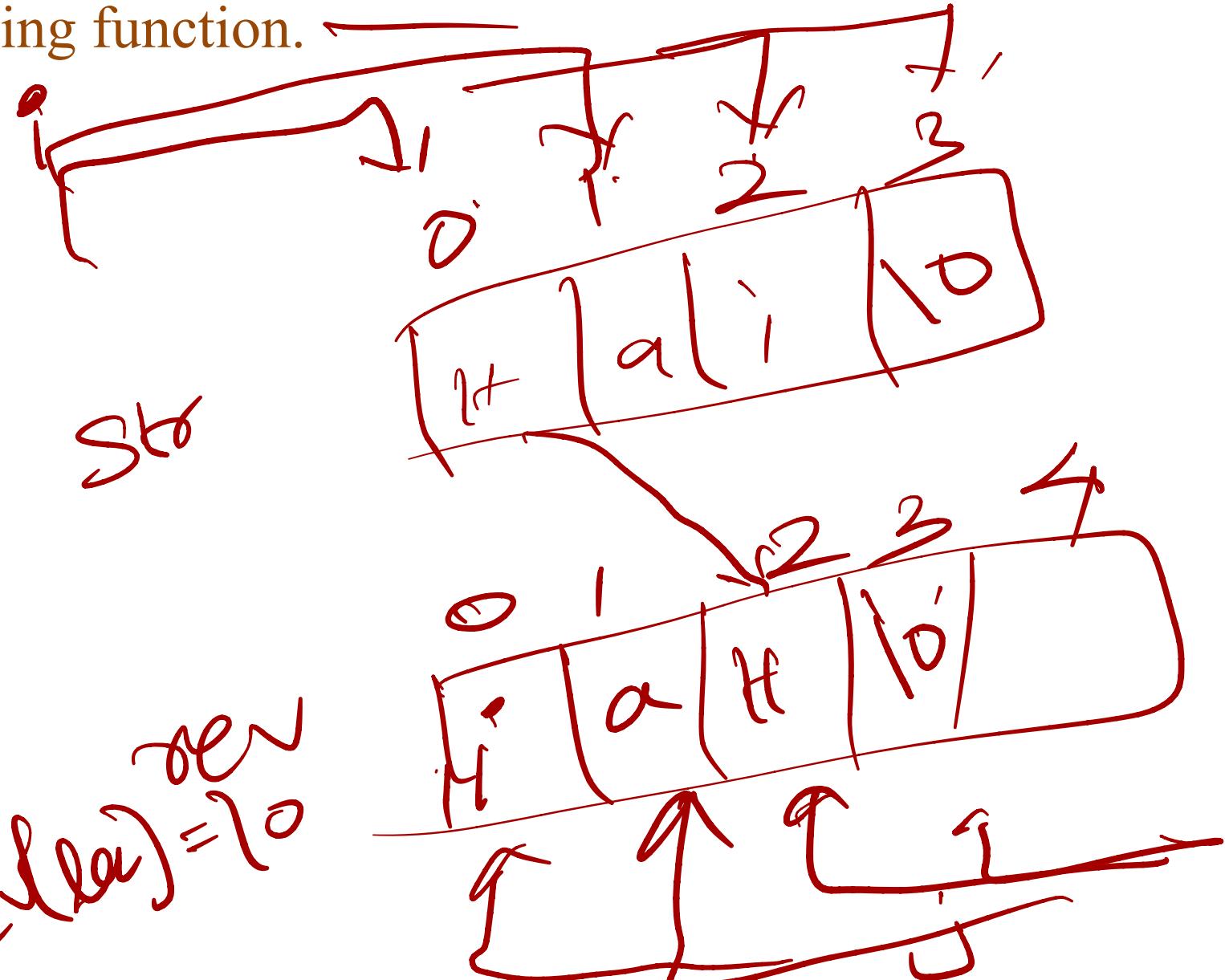


len = 6

### Question No. 13

Write a C program to print reverse of a string without using string handling function.

len = 3  
i=0  
j= len-1, j=  
arr = str[i]  
arr[i] = 0



```
#include<stdio.h>
void main()
{
char str[100],rev[100];
int i,j,len;
// read a string
printf("Enter a string : ");
scanf("%s",&str);
//reverse the string.
for( len=0;str[len]!='\0';len++);
for( i=0,j=len-1,str[i]!='\0';i++, j--)
    rev[j]=str[i];
rev[len]='\0';
printf("Reverse=%s",rev);
}
```

length

rev(len) = 1 2 3 4 5 6 7 8 9 10

### **Question No. 14**

Write a C program to check whether the given string is palindrome or not without using string handling function.

```
#include<stdio.h>
void main()
{
char str[100],rev[100];
int i,j,len,flag=1;
/// read a string
printf("Enter a string : ");
scanf("%os",&str);
//reverse the string
for( len=0;str[len]!='\0';len++);
for(i=0,j=len-1;str[i]!='\0';i++,j--)
    rev[j]=str[i];
rev[len]='\0';
for(i=0;str[i]!='\0';i++)
    if( str[i] != rev[i] )
        flag=0;
if( flag==1 )
    printf("Palindrome");
else
    printf("Not a palindrome");
}
```

. WAP to read a string. convert it to lower case.

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

```
void main()
```

```
{
```

```
char str[100];
```

```
int i;
```

```
printf("Enter a string : ");
```

```
scanf("%s",&str);
```

```
for( i=0;str[i]!='\0';i++)
```

```
{
```

```
if( str[i]>='A' && str[i]<='Z' )
```

```
str[i]=str[i]+32;
```

```
}
```

```
printf("result=%s",str);
```

```
}
```

Output1 :

Enter a string : RAMANA

result=ramana

+32  
str[i] +32  
str[i] =  
Str(C) =

### **Question No. 15**

Write a C program to concatenate two strings without using string handling function.

## Program

```
#include<stdio.h>
int main()
{
    char a[50],b[50];
    int i,n,j;
    printf("Enter first string:\n");
    scanf("%s",a);
    printf("Enter second string:\n");
    scanf("%s",b);
```

```
n=0;
```

```
for(i=0;a[i]!='\0';i++)
```

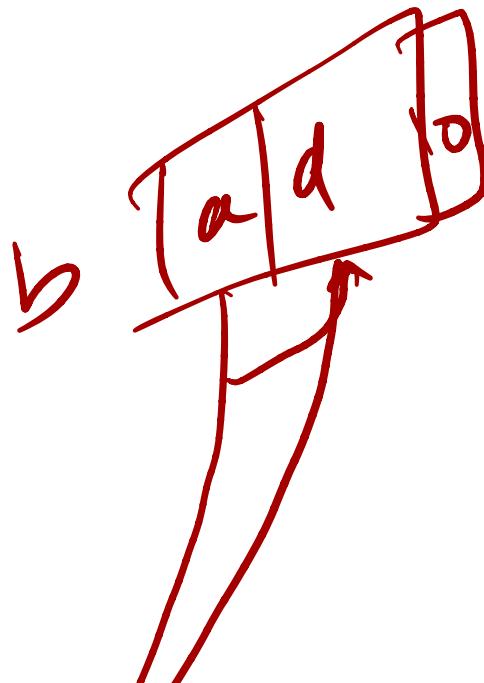
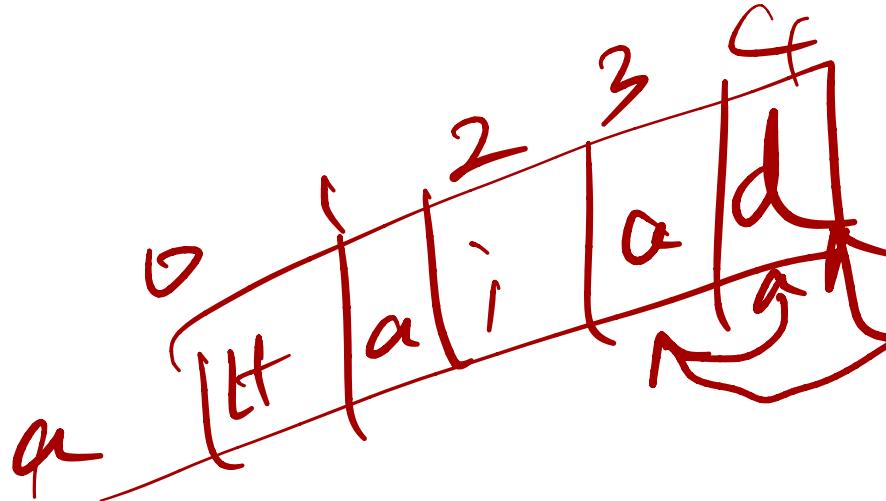
```
{
```

```
    n++;
}
```

length

```
for(j=0;b[j]!='\0';j++)  
{  
    a[n]=b[j];  
    n++;  
}  
a[n]='\0';  
printf("Concatenated string:%s\n",a);  
return 0;  
}
```

Let's



## **Output**

**Enter first string:**

**Good**

**Enter second string:**

**Morning**

**Concatenated string:GoodMorning**

### **Question No. 16**

Write a C program to copy content of one string into another without using string handling function.

## Program

```
#include<stdio.h>
int main()
{
    char a[50],b[50];
    int i;
    printf("Enter first string:\n");
    scanf("%s",a);
    printf("Enter second string:\n");
    scanf("%s",b);
    printf("Before copying:\na: %s\nb: %s\n",a,b);
    for(i=0;b[i]!='\0';i++)
    {
        a[i]=b[i];
    }
```

```
a[i]='\0';  
printf("After copying:\n a: %s\n b: %s\n",a,b);  
return 0;  
}
```

## **Output**

**Enter first string:**

**hi**

**Enter second string:**

**morning**

**Before copying:**

**a: hi**

**b: morning**

**After copying:**

**a: morning**

**b: morning**

**EST102**  
**PROGRAMMING IN C**

**MODULE IV**

# **SYLLABUS**

## **Working with functions**

Introduction to modular programming, writing functions, formal parameters, actual parameters.

Pass by Value, Recursion,

Arrays as Function Parameters, structure, union,

Storage Classes, Scope and life time of variables,

Simple programs using functions

# Introduction to Modular Programming

# Modular Programming

*Modular Programming is a software design technique that emphasizes separating the functionality of a program into independent, interchangeable modules, such that each contains everything necessary to execute only one aspect of the desired functionality.*

# Advantages of using Modular Programming

- 1) *Development can be divided*
- 2) *Readable programs*
- 3) *Programming errors are easy to detect*
- 4) *Allows reuse of codes*
- 5) *Improves manageability*
- 6) *Collaboration*

# Functions

- A *function* is a group of statements that together perform a task.
- Every C program has at least one function, main( ).
- Most programs can define additional functions.

# Types of Functions

There are two types of function in C programming :

- 1) *Standard library functions*
- 2) *User - defined functions*

# 1) Standard library functions

- *The standard library functions are built - in functions in c programming*
- *These functions are defined in header files.*

*For example,*

**printf( ), scanf( ), strlen( ), etc.**

## 2) User - defined functions

- *User can also create function as per their own needs.*
- *Such functions created by users are known as user - defined functions.*

## Advantages of User - defined functions

- 1) *The program will be easier to understand, maintain and debug.*
  - 2) *Reusable codes that can be used in other programs.*
  - 3) *A large program can be divided into smaller modules.*
-

## Parts of a User - defined function

1) Function Declaration or Prototype

2) Function Definition

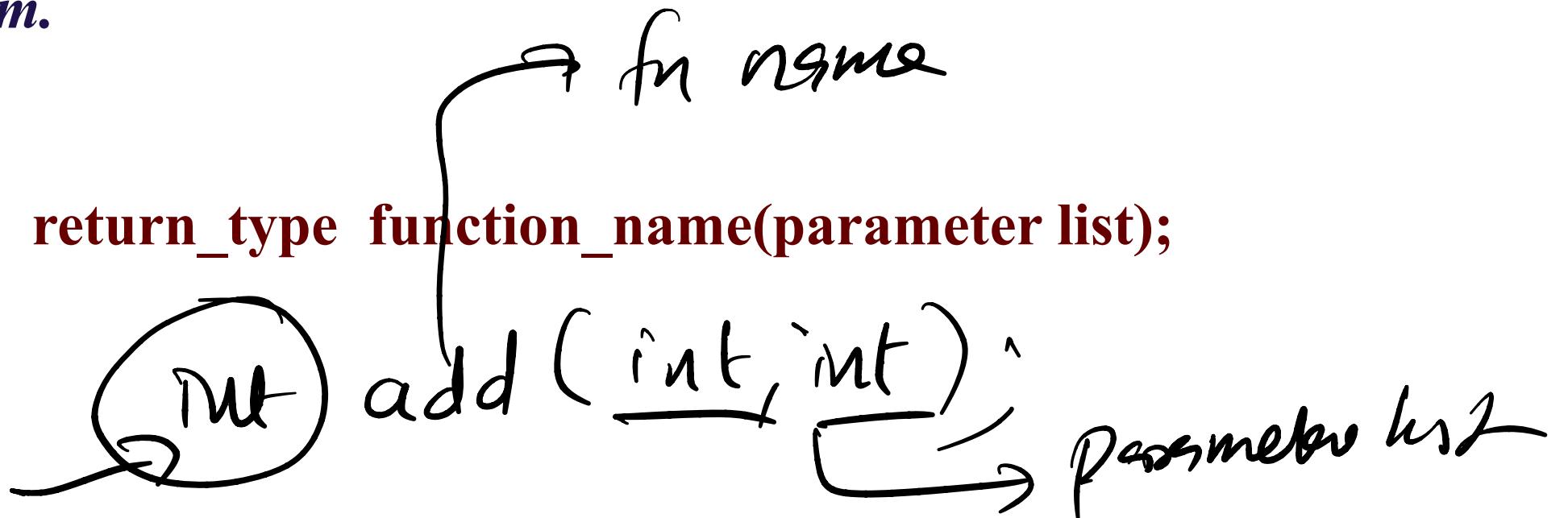
3) Function Call

## Function Declaration or Prototype

- A function prototype is simply the declaration of a function that specifies function's name, parameters and return type.
- It does not contain function body.
- It gives information to the compiler that the function may later be used in the program.

### Syntax

return\_type function\_name(parameter list);



- **Return Type**

*A function may return a value. The **return\_type** is the data type of the value the function returns. Some functions perform the desired operations without returning a value. In this case, the **return\_type** is the keyword **void**.*

- **Function Name**

*The actual name of the function.*

- **Parameter List or Arguments**

*When a function is invoked, we pass a value to the parameter. This value is referred to as actual parameter or argument.*

## Function Definition

- A function definition consists of a function header and a function body.

### Syntax

**return\_type function\_name(parameter list)**

{

**function body**

}

- Function Body

*The function body contains a collection of statements that define what the function does.*

## Function Call

- *Control of the program is transferred to the user - defined function by calling it.*
- *When a program calls a function, the program control is transferred to the called function.*
- *A called function performs a defined task.*
- *When its return statement is executed or when its closing brace is reached, it returns the program control back to the main program.*

## Syntax

**function\_name(parameter list);**



# Formal and Actual Parameters

- **Formal Parameter**

*A variable and its type as they appear in the function prototype.*

- **Actual Parameter**

*A variable corresponding to a formal parameter as they appear in the fun*

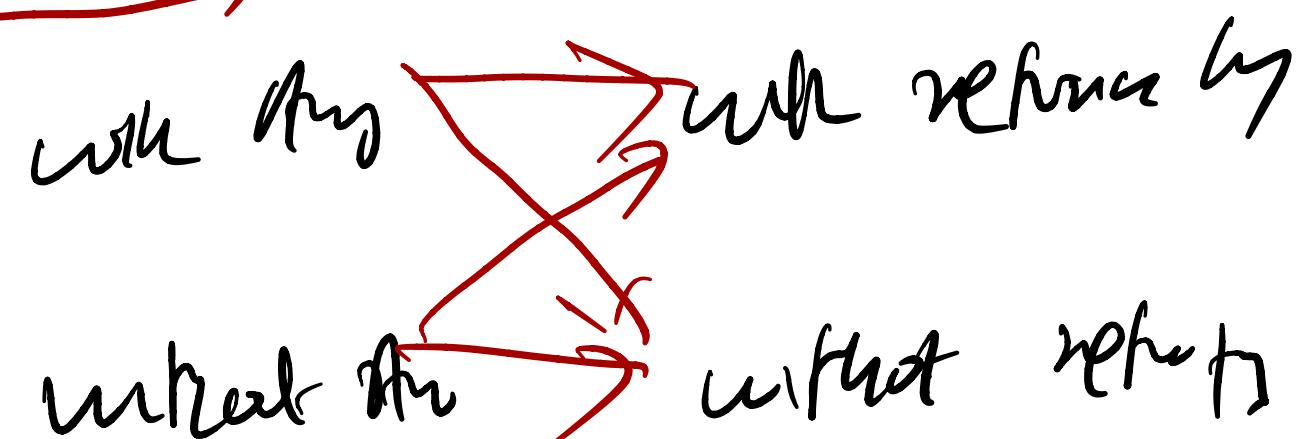
# Return Statement

- *The return statement terminates the execution of a function.*
- *Returns a value to the calling function.*
- *The program control is transferred to the calling function after the return statement.*

# Categories of Functions

Depending on whether arguments are present or not and whether a value is returned or not, functions are categorized into,

- 1) Functions without arguments and without return values
- 2) Functions without arguments and with return values
- 3) Functions with arguments and without return values
- 4) Functions with arguments and with return values



## **Program : C program to find sum of two numbers using function without arguments and without return values**

```
#include<stdio.h>
void add();
void main()
{
    add();

}

void add()
{
    int a,b,s;
    printf("Enter two numbers\n");
    scanf("%d%d",&a,&b);
    s=0;
    s=a+b;
    printf("Sum=%d\n",s);
}
```

## Program : C program to find sum of two numbers using function without arguments and with return values

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

```
int add();
```

```
int main()
```

```
{
```

```
    int c;
```

```
    c=add();
```

```
    printf("Sum=%d\n",c);
```

```
    return 0;
```

```
}
```

```
int add()
```

```
{
```

```
    int a,b,s;
```

```
    printf("Enter two numbers\n");
```

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

```
    s=0;
```

```
    s=a+b;
```

```
    return(s);
```

```
}
```

Program : C program to find sum of two numbers using function with arguments and without return values

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

```
void add(int,int);
```

```
int main()
```

```
{
```

```
    int a,b;
```

```
    printf("Enter two numbers\n");
```

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

```
    add(a,b);
```

```
    return 0;
```

```
}
```

```
void add(int p,int q)
```

```
{
```

```
    int s;
```

```
    s=0;
```

```
    s=p+q;
```

```
    printf("Sum=%d\n",s);
```

```
}
```

pass by value

a  
10

b  
20

add(10, 20)

P  
10

Q  
20

s  
30

30

## Program : C program to find sum of two numbers using function with arguments and with return values

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

```
int add(int,int);
```

```
int main()
```

```
{
```

```
    int a,b,c;
```

```
    printf("Enter two numbers\n");
```

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

```
    c=add(a,b);
```

c = add(a, b);

```
    printf("Sum=%d\n",c);
```

```
    return 0;
```

```
}
```

```
int add(int p,int q)
```

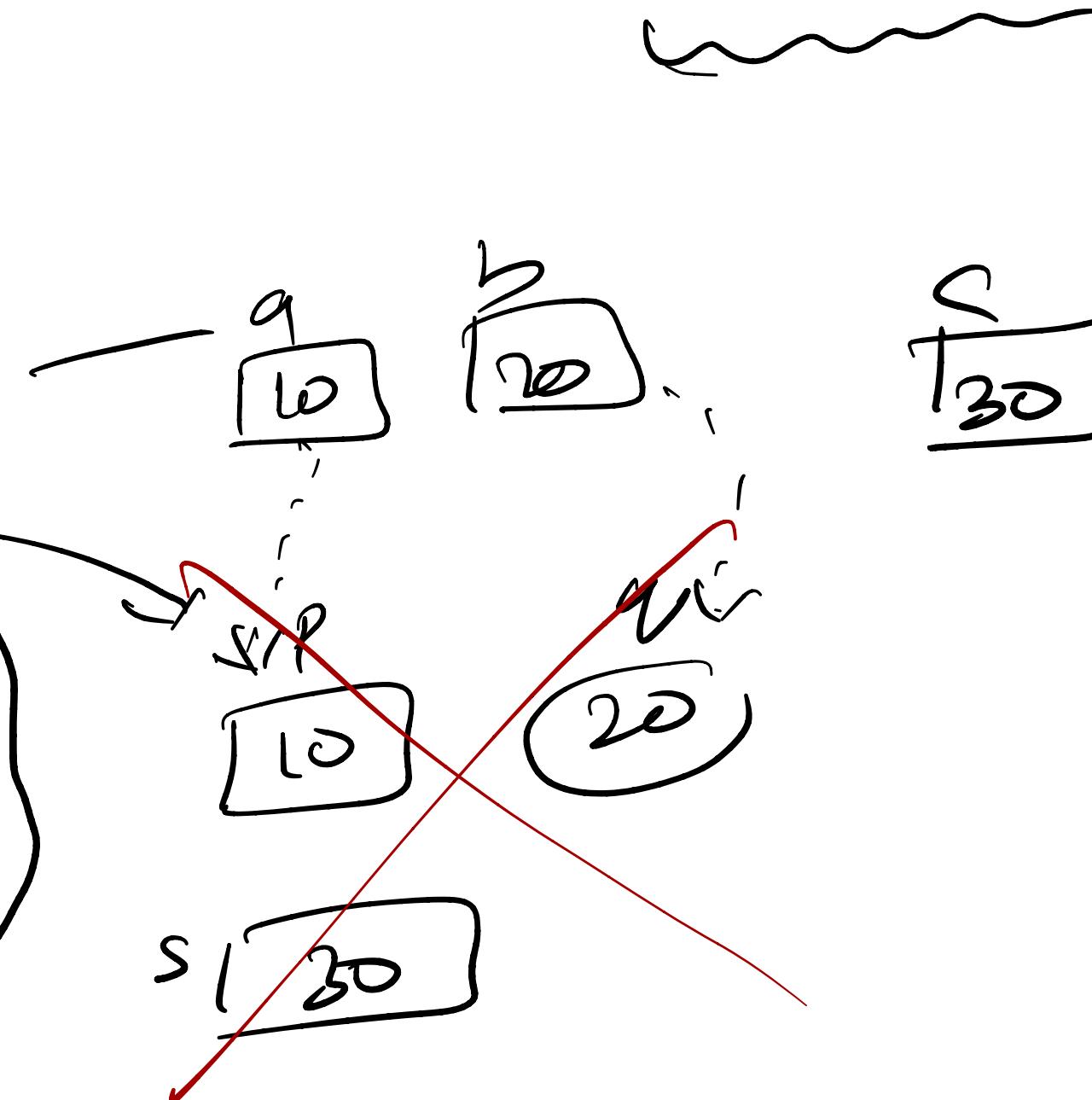
```
    int s;
```

```
    s=0;
```

```
    s=p+q;
```

```
    return s;
```

```
}
```



# Pass by Value

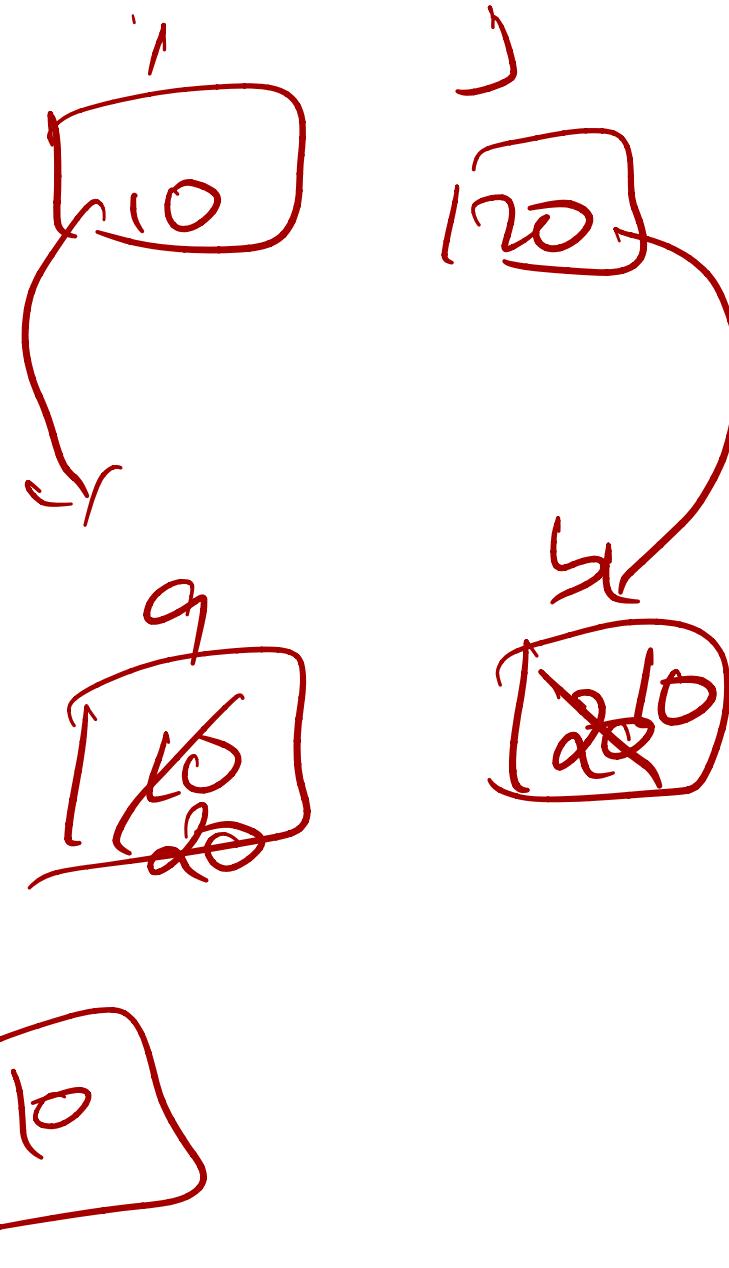
- Passing arguments by value which means contents of actual arguments in the calling function is copied to the formal arguments in the called function.
- Change in formal arguments is not reflected in actual arguments.

## C program to swap two numbers using function (Pass by value)

```
#include<stdio.h>
void swap(int,int);
int main()
{
    int i,j;
    printf("Enter two numbers\n");
    scanf("%d%d",&i,&j);
    printf("Before Swapping:\ni=%d\nj=%d\n",i,j);
    swap(i,j);
    printf("After Swapping:\ni=%d\nj=%d\n",i,j);
    return 0;
}
```

```
void swap(int a,int b)
{
    int temp;
    temp=a;
    a=b;
    b=temp;
```

q) `printf("After swap  
value of 1st variable  
& 2nd variable is %d,%d",  
a,b);`



# Pass by Reference

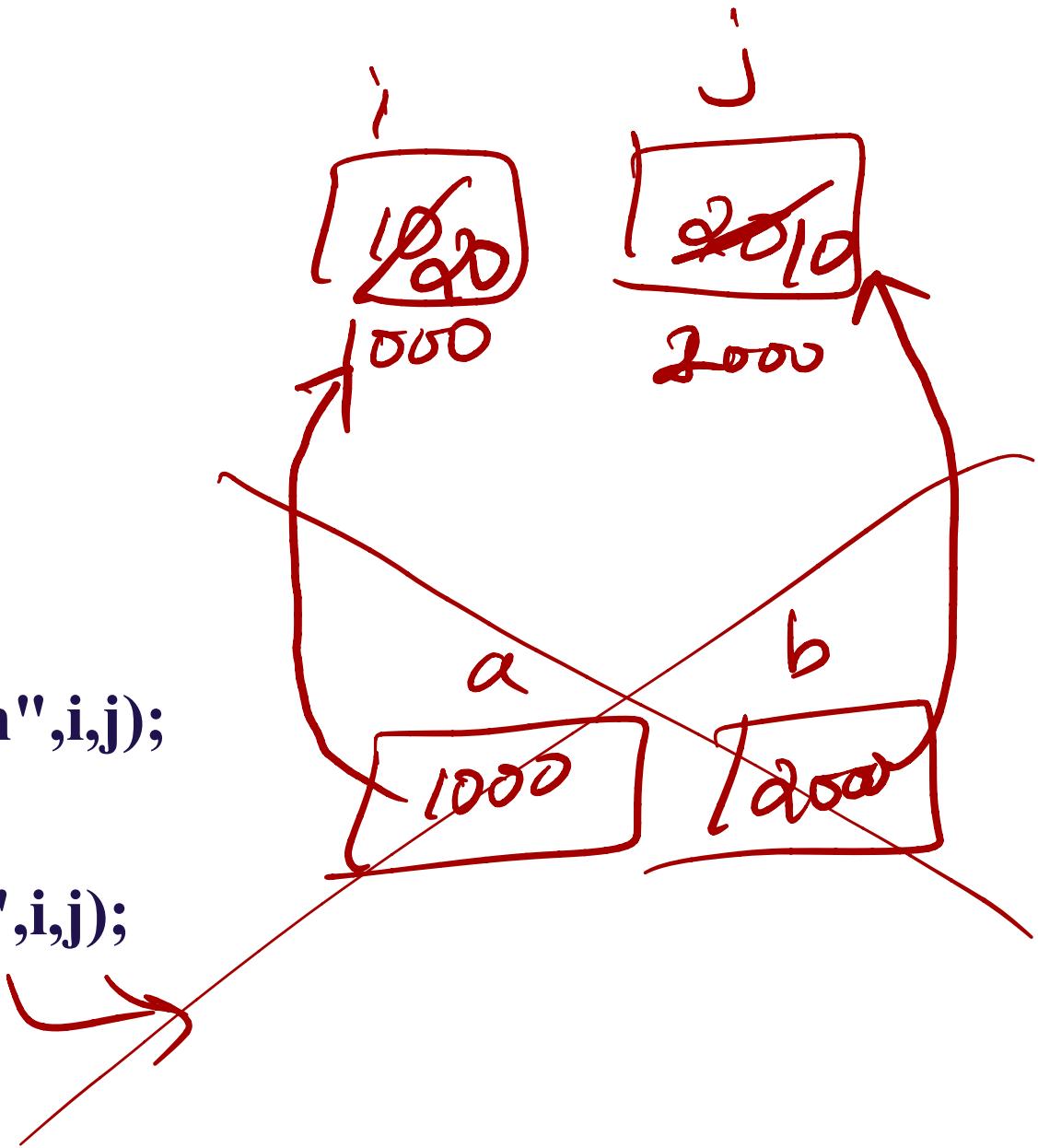
- Passing arguments by reference which means address of actual arguments in the calling function is copied to the formal arguments in the called function.
- When we pass addresses to a function, the arguments receiving the address should be pointers.
- Change in formal arguments is reflected in actual arguments.

## Program : C program to swap two numbers using function (Pass by reference)

```
#include<stdio.h>

void swap(int *,int *);

int main()
{
    int i,j;
    printf("Enter two numbers\n");
    scanf("%d%d",&i,&j);
    printf("Before swapping:\ni=%d\nj=%d\n",i,j);
    swap(&i,&j);
    printf("After swapping:\ni=%d\nj=%d\n",i,j);
    return 0;
}
```



```
void swap(int *a,int *b)
{
    int temp;
    temp=*a;
    *a=*b;
    *b=temp;
}
```

$*(\text{a})$   
 $*(\text{1000})$   
= 10

temp

temp  $\leftarrow$  ~~\*a~~  
10

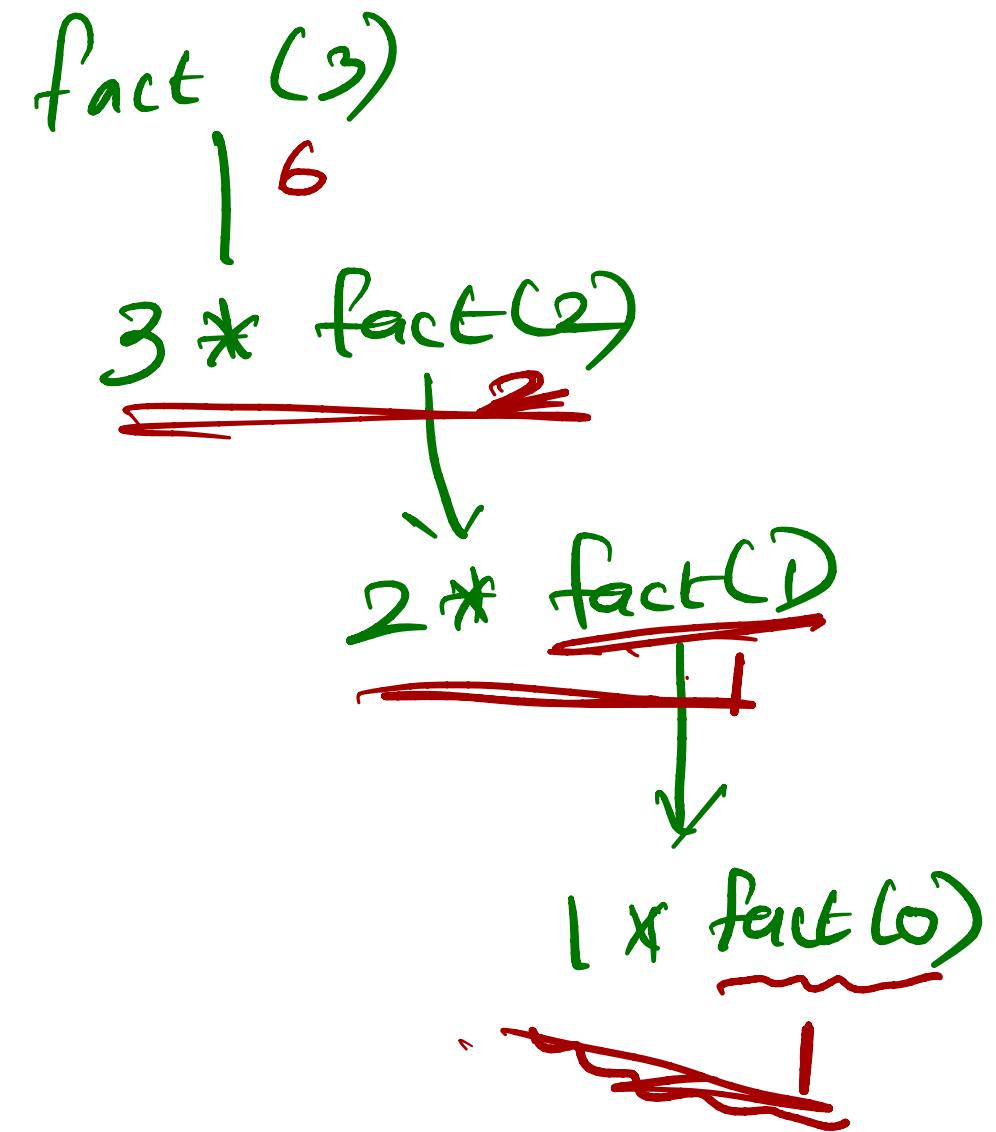
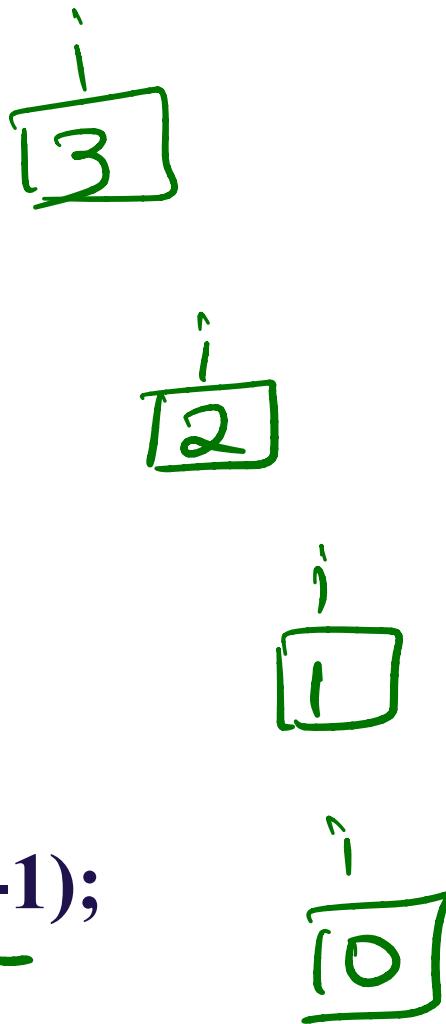
# Recursion

- *Recursion* is a process while which a function called itself repeatedly until some specified condition has been satisfied.
- The process is used for repetitive computation in which each action is stated in terms of previous results.

## Program : C program to find factorial of a number using recursion

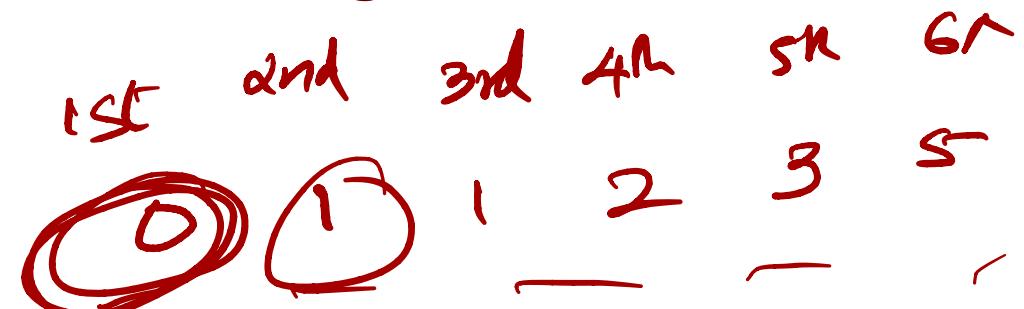
```
#include<stdio.h>
int fact(int); → int fact (int);
int main()
{
    int n;
    printf("Enter a number\n");
    scanf("%d",&n);
    if(n<0)
    {
        printf("Factorial does not exist\n");
    }
    else
    {
        printf("Factorial=%d\n",fact(n));
    }
    return 0;
}
```

```
int fact(int i)
{
    if(i==0)
    {
        return 1;
    }
    else
    {
        return i*fact(i-1);
    }
}
```



## Program : C program to print fibonacci series using recursion

```
#include<stdio.h>
int fib(int);
int main()
{
    int n,i;
    printf("Enter the limit\n");
    scanf("%d",&n);
    printf("Fibonacci series:\n");
    for(i=1;i<=n;i++)
    {
        printf("%d\t",fib(i));
    }
    return 0;
}
```



$$\text{fib}(1) = 0$$

$$\text{fib}(2) = 1$$

$$\text{fib}(3) = \text{fib}(2) + \text{fib}(1)$$

$$\begin{aligned}\text{fib}(4) &= \text{fib}(3) + \text{fib}(2) \\ \text{fib}(5) &= \text{fib}(4) + \text{fib}(3) \\ \text{fib}(n) &= \text{fib}(n-1) + \text{fib}(n-2)\end{aligned}$$

```
int fib(int k)
```

```
{  
    if(k==1)  
    {  
        return 0;  
    }  
    else if(k==2)  
    {  
        return 1;  
    }  
    else  
    {  
        return fib(k-1)+fib(k-2);  
    }  
}
```

## Output

Enter the limit

5

Fibonacci series:

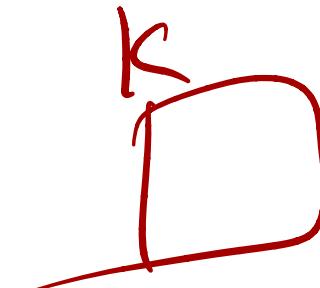
0      1      1      2      3

# Array as Function Parameters

- It is also possible to pass an entire array as an argument to a function.
- For this, it is sufficient to list the name of the array and size of the array as arguments.
- The name of the array interpreted as the address of the memory location containing the first address element.
- The formal argument declaration needs a set of empty square brackets to indicate that it is an array.

## Program : C program to to find largest element among n numbers using function

```
#include<stdio.h>
int largest(int [],int);
int main()
{
    int a[10],n,i,l;
    printf("Enter the size\n");
    scanf("%d",&n);
    printf("Enter elements:\n");
    for(i=0;i<n;i++)
    {
        scanf("%d",&a[i]);
    }
    l=largest(a,n);
    printf("Largest=%d\n",l);
    return 0;
}
```



```
int largest(int p[],int k)
```

```
{
```

```
    int b,j;
```

```
    b=p[0];
```

```
    for(j=1;j<k;j++)
```

```
{
```

```
    if(p[j]>b)
```

```
{
```

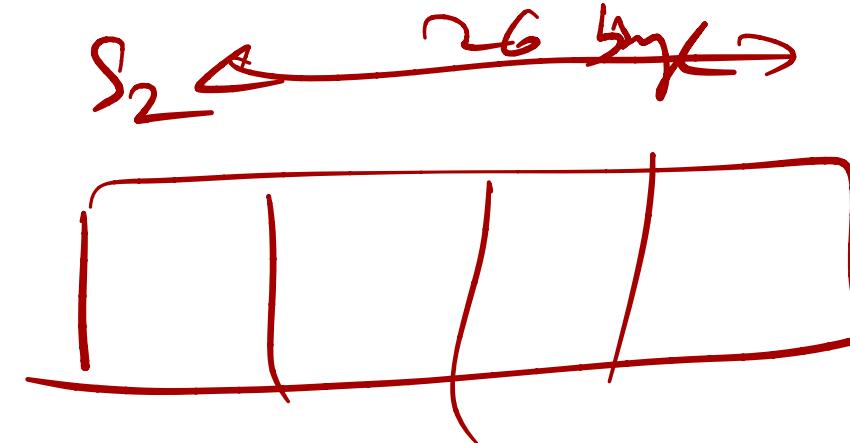
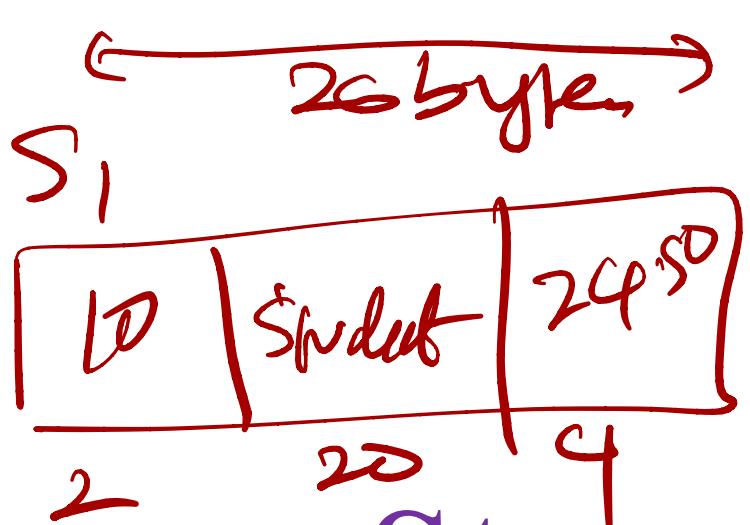
```
    b=p[j];
```

```
}
```

```
}
```

```
return b;
```

```
}
```



## Structure and Union

Struct Book  
 2by First Page  
 20 byt clear name [20]  
 4 byt float price  
 $S_1, S_2$  → Structure variables.

# Structure

- A *structure* is a group of variables of different types under one name.
- It is also called *conglomerate data type*.
- Here, we can refer collectively to a set of data items that differ among themselves in type.
- The individual data items are said to occupy *fields* within the structure.
- The individual elements within the structure are called *members*.

# Defining a Structure

- The general form of structure declaration is:

**struct**    *Structure Name*

{

**member\_1;**

**member\_2;**

.....

}**variable\_list;**



which can also be declared as:

**struct name**

{

**member\_1;**

**member\_2;**

.....

};

**int main( )**

{

**struct name variable\_list;**

.....

.....

**return 0;**

}

- **struct** is the keyword indicating a structure definition.
- **variable\_list** is the names of structure type variables.
- Individual fields are declared inside the braces, separated by a **;**
- The declaration **member\_1** declares the first field, **member\_2** declares the second field and so on.
- Syntax for declaring members are,

**data\_type field\_name;**

# Processing Structure Variables

- The members within a structure have no separate identity without the structure variable name.
- The link between a member and structure variable is established by using the *member operator* ., which is also known as *dot operator* or *period operator*.

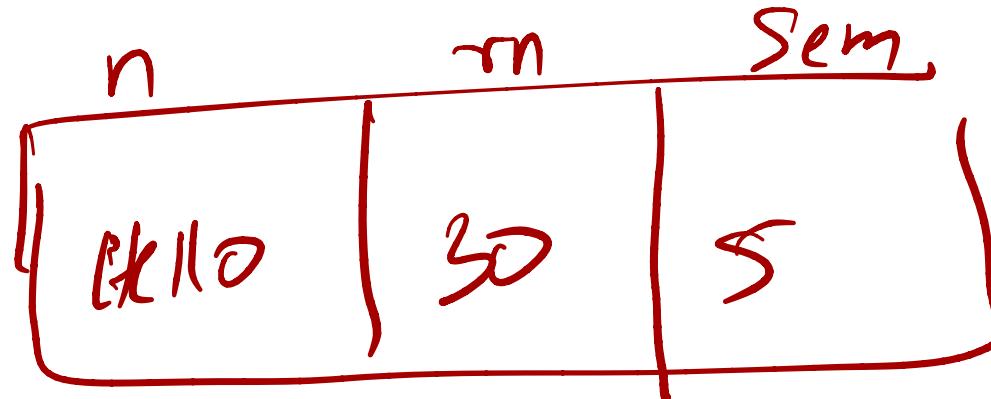
## **Question No. 1**

Write a C program to store information of a student using structure.

## Program

```
#include<stdio.h>
struct Student
{
    char n[10];
    int rn;
    int sem;
}st;
int main()
{
    printf("Enter name,roll number and semester details of student 1\n");
    scanf("%s%d%d",&st.n,&st.rn,&st.sem);
    printf("Entered name,roll number and semester details of student 1\n");
    printf("\n%d\n%d\n",st.n,st.rn,st.sem);
    return 0;
}
```

ST



10 byte 2 byte 2 byte

st.n = Hello

st.rn = 30

st.sem = 5

## Question No. 2

Using structure, read and print data of n employees (Name, Employee Id and Salary).

Struct emp

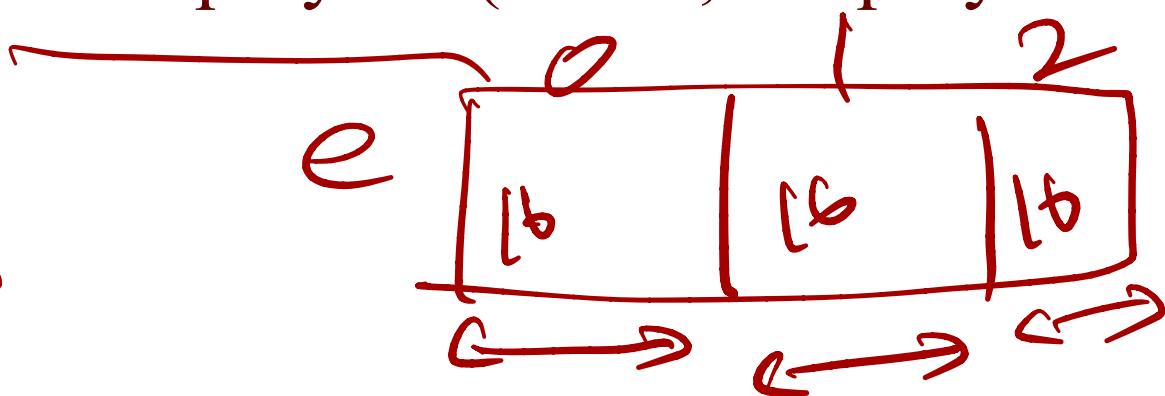
```
struct emp
{
    char name[10];
    int id;
    float salary;
}
```

e[10] 1 2

name[10]: 10 bytes

id: 2 bytes

salary: 4 bytes



## Program

```
#include<stdio.h>
struct
{
    char n[10];
    int id;
    float s;
}e[10];
int main()
{
    int m,i;
    printf("Enter the limit\n");
    scanf("%d",&m);
    for(i=0;i<m;i++)
    {
        printf("Enter name,id and salary of Employee %d:\n",i+1);
        scanf("%s%d%f",e[i].n,&e[i].id,&e[i].s);
    }
}
```

```
for(i=0;i<m;i++)
{
    printf("Entered name,id and salary of Employee %d:\n",i+1);
    printf("%s\n%d\n%0.2f\n",e[i].n,e[i].id,e[i].s);
}
return 0;
}
```

# Difference between Array and Structure

| Array                                                 | Structure                                          |
|-------------------------------------------------------|----------------------------------------------------|
| Array is collection of homogeneous data.              | Structure is the collection of heterogeneous data. |
| Array data are access using index.                    | Structure elements are access using . operator.    |
| Array allocates static memory.                        | Structures allocate dynamic memory.                |
| Array element access takes less time than structures. | Structure elements takes more time than Array.     |

# Union

- A *union* is a group of variables of different types under one name.
- The syntax of union is same as that of structure.
- In structures, each member has its own storage location, whereas all the members of union uses the same location.

# Defining a Union

- The general form of union declaration is:

**union**

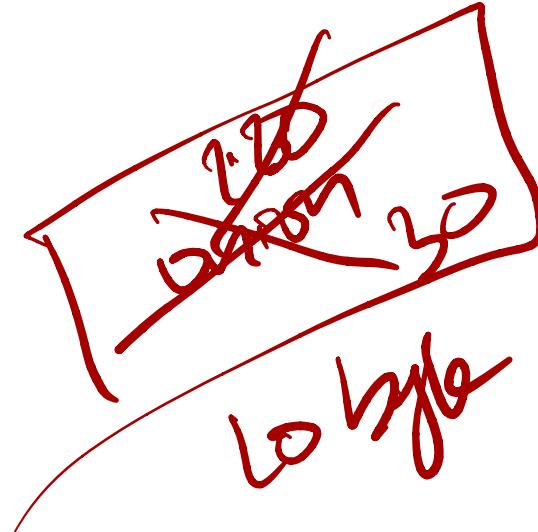
{

**member\_1**;

**member\_2**;

.....

}**variable\_list**;



union book  
{ char name[10] num  
    int pages; float price; }  
book

# Difference between Structure and Union

|                           | STRUCTURE                                                                                                                                                                                    | UNION                                                                                                                                                                                         |
|---------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Keyword                   | The keyword <b>struct</b> is used to define a structure                                                                                                                                      | The keyword <b>union</b> is used to define a union.                                                                                                                                           |
| Size                      | When a variable is associated with a structure, the compiler allocates the memory for each member. The size of structure is <b>greater than or equal to the sum of sizes of its members.</b> | when a variable is associated with a union, the compiler allocates the memory by considering the size of the largest memory. So, size of <b>union is equal to the size of largest member.</b> |
| Memory                    | Each member within a structure is assigned unique storage area of location.                                                                                                                  | Memory allocated is shared by individual members of union.                                                                                                                                    |
| Value Altering            | Altering the value of a member will not affect other members of the structure.                                                                                                               | Altering the value of any of the member will alter other member values.                                                                                                                       |
| Accessing members         | Individual member can be accessed at a time.                                                                                                                                                 | Only one member can be accessed at a time.                                                                                                                                                    |
| Initialization of Members | Several members of a structure can initialize at once.                                                                                                                                       | Only the first member of a union can be initialized.                                                                                                                                          |

### **Question No. 3**

Write a C program to store information of a student using union.

## Program

```
#include<stdio.h>
union
{
    char n[10];
    int rn;
    int sem;
}st;
int main()
{
    printf("Enter name of student 1\n");
    scanf("%s",st.n);
    printf("Entered name of student 1: %s\n",st.n);
    printf("Enter roll number of student\n");
    scanf("%d",&st.rn);
    printf("Entered roll number of student: %d\n",st.rn);
```

```
printf("Enter semester details of student\n");
scanf("%d",&st.sem);
printf("Entered semester details of student: %d\n",st.sem);
return 0;
}
```

# Storage Classes

# Scope and Lifetime of Variables

## Scope

- *Scope defines the visibility of a variable.*
- *It defines where a variable can be accessed.*
- *The scope of a variable is **local** or **global**.*
- *The variable defined within the block has local scope. They are visible only to the block in which they are defined.*
- *The variable defined in global area is visible from their definition until the end of program. It is visible everywhere in program.*

## Lifetime

- *The lifetime of a variable defines the duration for which the computer allocates memory for it.*
- *The duration between allocation and deallocation of memory.*
- *In C language, a variable can have automatic, static or dynamic lifetime.*

**Automatic** – *A variable with automatic lifetime are created. Every time, their declaration is encountered and destroyed. Also, their blocks are exited.*

**Static** – *A variable is created when the declaration is executed for the first time. It is destroyed when the execution stops/terminates.*

**Dynamic** – *The variables memory is allocated and deallocated through memory management functions.*

# Storage Classes

- *Storage classes specify the scope, lifetime and binding of variables.*
- *To fully define a variable, one needs to mention not only its ‘type’ but also its storage class.*
- *A variable name identifies some physical location within computer memory, where a collection of bits are allocated for storing values of variable.*

# Storage Classes in C

There are four storage classes in C.

- 1) *Automatic*
- 2) *Static*
- 3) *Register*
- 4) *External*



The keywords **auto**, **static**, **register** and **extern** are used for above storage classes respectively.

We can specify a storage class while declaring a variable.

## Syntax

```
storage_class data_type variable_name;
```

# Automatic storage class

- Syntax to declare automatic variable is:

*auto data\_type variable\_name;*

auto int i = 0;

- Features

~~Storage area : Memory~~

~~Default initial value : Garbage value~~

~~Scope : Local to the block in which the variable is defined~~

~~Lifetime : Till the control remains within the block in which the variable is defined~~

- All the variables which are declared inside a block or function without any storage class specifier are automatic variables.
- Automatic variables came into existence each time the function is executed and destroyed when execution of the function completes.

# Register storage class

- Syntax to declare register variable is:

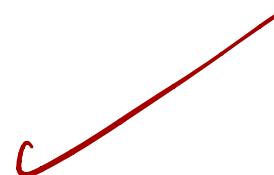
**register data\_type variable\_name;**

- Features

**Storage area :** CPU Registers



**Default initial value :** Garbage value



**Scope :** Local to the block in which the variable is defined



**Lifetime :** Till the control remains within the block in which the variable is defined

- Register storage class can be applied only to automatic variables.
- Its scope, lifetime and default initial value are same as that of automatic variables.
- Automatic variables are stored in memory but register values are stored in CPU registers.
- Registers are small storage units present in the processor.
- Variables stored in registers can be accessed much faster than the variables stored in memory.

# Static storage class

- Syntax to declare static variable is:

**static data\_type variable\_name;**

- Features

**Storage area :** Memory

**Default initial value :** Zero

**Scope :** Local to the block in which the variable is defined

**Lifetime :** Value of the variable continues to exist between different function calls

- Static variables are initialized to zero if not initialized.
- Static variables does not disappear when the function is no longer active.

# External storage class

- Syntax to declare external variable is:

**extern    data\_type    variable\_name;**

- Features

**Storage area :** Memory

**Default initial value :** Zero

**Scope :** Global

**Lifetime :** Till the program's execution does not comes to an end

- External variables are declared outside all the functions, therefore are available to all functions that want to use them.

| <b>Properties</b> | <b>Storage</b> | <b>Default Initial Value</b> | <b>Scope</b>                                        | <b>Life</b>                                                                |
|-------------------|----------------|------------------------------|-----------------------------------------------------|----------------------------------------------------------------------------|
| Storage Class     |                |                              |                                                     |                                                                            |
| <b>Automatic</b>  | Memory         | Garbage Value                | Local to the block in which the variable is defined | Till the control remains within the block in which the variable is defined |
| <b>Register</b>   | CPU Registers  | Garbage Value                | Local to the block in which the variable is defined | Till the control remains within the block in which the variable is defined |
| <b>Static</b>     | Memory         | Zero                         | Local to the block in which the variable is defined | Value of the variable continues to exist between different function calls  |
| <b>External</b>   | Memory         | Zero                         | Global                                              | Till the program's execution doesn't come to an end                        |

**EST102**  
**PROGRAMMING IN C**

**MODULE V**

# **SYLLABUS**

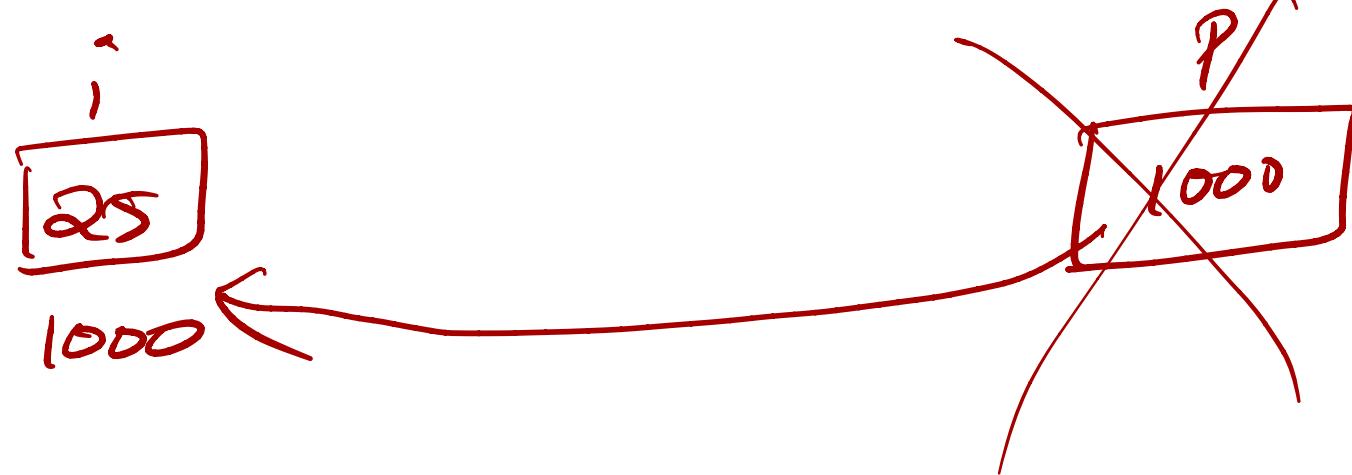
## **Pointers and Files**

**Basics of Pointer:** declaring pointers, accessing data through pointers, NULL pointer, array access using pointers, pass by reference effect

**File Operations:** open, close, read, write, append

**Sequential access and random access to files:** In built file handling functions (rewind(), fseek(), ftell(), feof(), fread(), fwrite())

Simple programs covering pointers and files



## Basics of Pointer

```
void main()
{
    int i;
    set(&i);
    printf("0d", i);
}
```

~~void set(int \*p)~~

~~\*p = 25;~~

~~p = &i;~~

~~i~~

~~25~~

# Pointer

- *The pointer in C language is a variable.*
- *It is also known as locator or indicator that points to an address of a value.*
- *A pointer is a derived data type that refers to another data variable by storing the variable's memory address rather than data.*

*Consider the declaration,*

```
int i = 5;
```

*This declaration tells the C compiler to:*

- a) *Reserve space in memory to hold the integer value.*
- b) *Associate the name i with this memory location.*
- c) *Store the value 5 at this location.*

1000

5

i

*We see that the computer has selected memory location 1000 as the place to store the value 5. The important point is, i's address in memory is a number.*

# Symbols used in Pointer

| <i>Symbol</i>                | <i>Name</i>                 | <i>Description</i>                         |
|------------------------------|-----------------------------|--------------------------------------------|
| &<br><i>(Ampersand sign)</i> | <i>Address Operator</i>     | <i>Determine the address of a variable</i> |
| *                            | <i>Indirection Operator</i> | <i>Access the value of an address</i>      |

# Declaring and Initializing Pointers

- The process of assigning the address of a variable to a pointer variable is known as *initialization*.
- The only requirement here is that variable quantity must be *declared* before the initialization takes place.
- General syntax for declaring and initializing a pointer variable is,

```
data_type *pointer_variable;  
pointer_variable=&variable_name;
```

*Asterisk(\*) can be located immediately before the pointer variable, or between the data type and pointer variable, or immediately after the data type.*

For example,

int a;

a=10;

int \*ptr; //declaration

ptr = &a; //initialization

int i = 10;

int \*p;

p = &i;

We can also combine the initialization with the declaration.

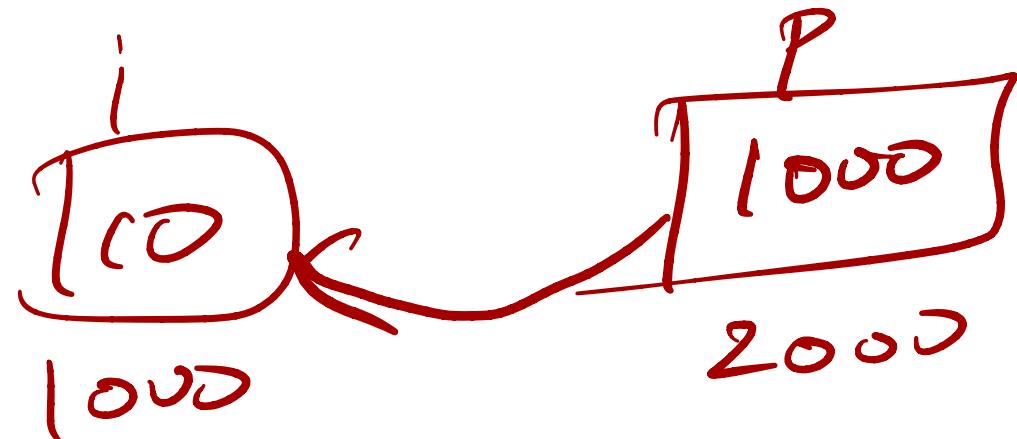
int a=10;      i = 10

int \*ptr = &a;      p = 1000

\*p = 10

p = 1000

&p = 2000

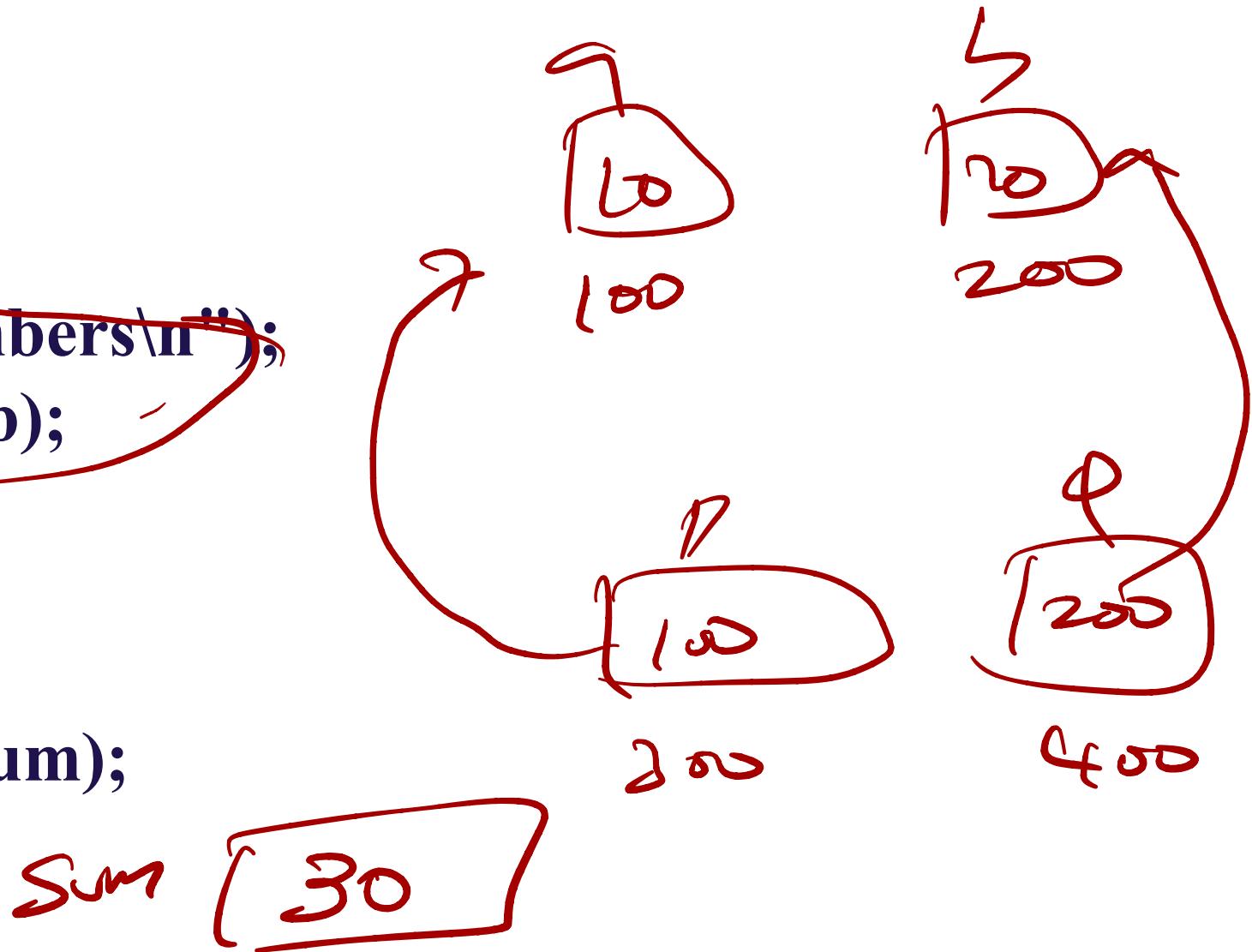


## **Question No. 1**

Write a C program to add two numbers using pointers.

## Program

```
#include<stdio.h>
int main()
{
    int a,b,*p,*q,sum;
    printf("Enter two numbers\n");
    scanf("%d%d",&a,&b);
    p=&a;
    q=&b;
    sum=*p+*q;
    printf("Sum=%d\n",sum);
    return 0;
}
```



## **Question No. 2**

Write a C program to swap two numbers using pointers.

## Program

```
#include<stdio.h>
int main()
{
    int a,b,*m,*n,temp;
    printf("Enter two numbers\n");
    scanf("%d%d",&a,&b);
    m=&a;
    n=&b;
    printf("Before swapping:\na=%d\nb=%d\n",a,b);
    temp=*m;
    *m=*n;
    *n=temp;
    printf("After swapping:\na=%d\nb=%d\n",a,b);
    return 0;
}
```

## **Output**

Enter two numbers

30

20

Before swapping:

a=30

b=20

After swapping:

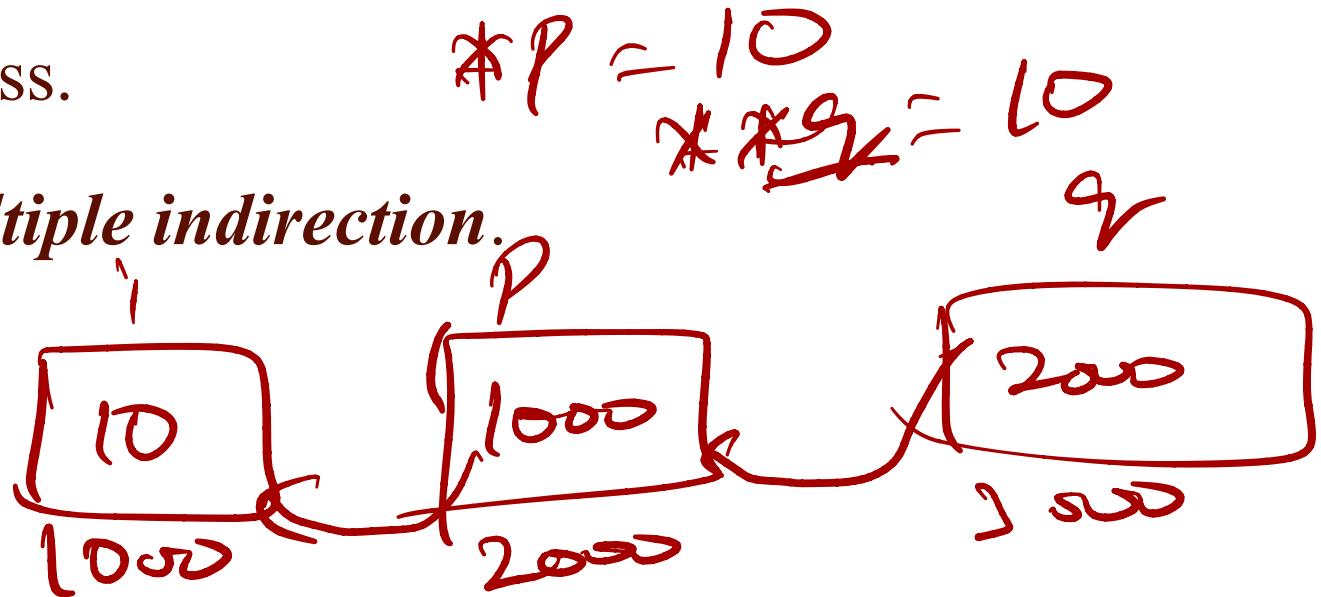
a=20

b=30

# Pointer to a Pointer (Chain of Pointers)

- It is possible to make pointer to point to another pointer.
- Pointer is a variable that contains address of another variable. Now this variable itself might be another pointer. Thus, we now have a pointer that contains another pointer's address.
- The representation is called *multiple indirection*.

$$P = \&i ; \\ q = \&P ;$$



### **Question No. 3**

What will be the output of the following C program?

```
#include<stdio.h>
int main()
{
    int a,*p,**q;
    a=120;
    p=&a;
    q=&p;
    printf("Value of a=%d\n",a);
    printf("Value available at *p=%d\n",*p);
    printf("Value available at **q=%d\n",**q);
    return 0;
}
```

## **Question No. 4**

Write a C program to find sum of n numbers using pointers.

## Program

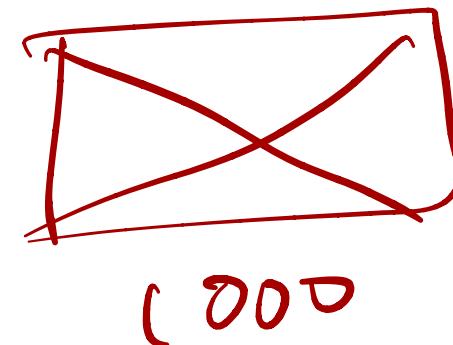
```
#include<stdio.h>
int main()
{
    int a[10],i,n,sum,*ptr;
    printf("Enter the limit\n");
    scanf("%d",&n);
    printf("Enter the elements\n");
    sum=0;
    ptr=a; //ptr=&a[0];
    for(i=0;i<n;i++)
    {
        scanf("%d",&a[i]);
        sum=sum+*ptr;
        ptr++;
    }
}
```

```
printf("Sum=%d\n",sum);
return 0;
}
```

# NULL Pointer

- A null pointer in C is a pointer that is assigned to zero or NULL where a variable that has no valid address.
- The null pointer usually does not point to anything.
- In C programming language NULL is a macro constant that is defined in a few of the header files like stdio.h, alloc.h, mem.h, stddef.h, stdlib.h.
- Also, note that NULL should be used only when we are dealing with pointers only.

int \*p = NULL;



- Syntax for declaring NULL pointer is as follows:

```
int *pointer_var;
```

```
pointer_var=NULL;
```

or

```
int *pointer_var = NULL;
```

or

```
int *pointer_var;
```

```
pointer_var=0;
```

or

```
int *pointer_var = 0;
```

# Advantages of using Pointer

- 1) *Less time in program execution*
- 2) *Working on the original variable*
- 3) *With the help of pointers, we can create data structures (linked-list, stack, queue).*
- 4) *Returning more than one values from functions*
- 5) *Searching and sorting large data very easily*
- 6) *Dynamically memory allocation*

# Disadvantages of using Pointer

- 1) *Sometimes by creating pointers, such errors come in the program, which is very difficult to diagnose.*
- 2) *Sometimes pointer leaks in memory are also created.*
- 3) *If extra memory is not found then a program crash can also occur.*

**File :- it is a group of related information which is  
stored under a name.**

# Files

- A file represents a sequence of bytes, regardless of it being a text file or a binary file.
- C programming language provides access on high level functions as well as low level (OS level) calls to handle file on your storage devices.

# Types of files in c

- Text Files
- Binary Files

## Types of Files in C



file.txt



file.bin

## 1. Text Files

- A text file contains data in the **form of ASCII characters** and is generally used to store a stream of characters.
- Each line in a text file ends with a new line character ('\n').
- It can be read or written by any text editor.
- They are generally stored with .txt file extension.
- Text files can also be used to store the source code.

## 2. Binary Files

- A binary file contains data in **binary form** (i.e. 0's and 1's) instead of ASCII characters. They contain data that is stored in a similar manner to how it is stored in the main memory.
- The binary files can be created only from within a program and their contents can only be read by a program.
- More secure as they are not easily readable.
- They are generally stored with .bin file extension.

# Need for File Handling in C

- Reusability helps to retain the data collected after the software is run.
- Huge storage capacity you don't need to think about the issue of mass storage using data.
- Save time Unique applications need a lot of user feedback. You can quickly open some aspect of the code using special commands.
- Portability You can quickly move file material from one operating device to another without thinking about data loss.

# File Operations

- 1) Creation of a new file  
(fopen with attributes as “a” or “a+” or “w” or “w++”)
- 1) Opening an existing file  
(fopen)
- 1) Reading from file  
(fscanf or fgets)
- 1) Writing to a file  
(fprintf or fputs)
- 1) Moving to a specific location in a file  
(fseek, rewind)
- 1) Closing a file  
(fclose)

Handwritten red notes on the right side of the slide:

- Abbreviations:
  - r
  - w
  - a
  - rb
  - wb
  - ab
- A brace groups the abbreviations r, w, and a under the label "text".
- A brace groups the abbreviations rb, wb, and ab under the label "at".

Types of files :- Files are 2 types.

1. Text files
2. Binary files.

**Text file :-**

It is in human readable format. It contains the data in the form of alphabets, digits and symbols.

**Binary file :-** It is a machine readable file. It contains the data in the form of 0 and 1.

## Operations of files :

1. Create a file. ✓
2. Open a file. ✓
3. Close a file. ✓
4. Read the file. ✓
5. Write the file. ✓
6. Append the file. ✓

## File modes :-

### 1. W :

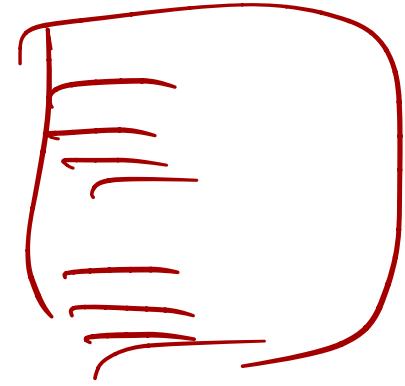
- ❖ It stands for write mode.
- ❖ It allows to write.
- ❖ It creates a new file.
- ❖ If the file already exists then it deletes and recreate the file.

R:

- ❖ It stands for read mode.
  - ❖ It allows to read.
  - ❖ It opens an existing file.
  - ❖ If the file does not exist then it produce error.
- 

A :-

- ❖ It stands for append mode.
- ❖ It allow to append.
- ❖ If opens an existing file.
- ❖ If the file does exist then it produce error.



W+ :-

- ❖ It stands for write plus mode.
- ❖ It allows to write and read.
- ❖ It creates a new file.
- ❖ If the file already exists then it deletes and recreate the file.

**R+**

- ❖ It stands for read plus mode.
- ❖ It allows to read and write.
- ❖ It opens an existing file.
- ❖ If the file does not exist then it produce error.

A+ :-

- ❖ It stands for append plus mode.
- ❖ It allow to append and read.
- ❖ If opens an existing file.
- ❖ If the file does exist then it produce error.

Wb

- ❖ It stands for write binary mode.
- ❖ It allows to write.
- ❖ It creates a new file.
- ❖ If the file already exists then it deletes and recreate the file.

Rb

- ❖ It stands for read binary mode.
- ❖ It allows to read.
- ❖ It opens an existing file.
- ❖ If the file does not exist then it produce error.

Ab :-

- ❖ It stands for append binary mode.
- ❖ It allow to append.
- ❖ If opens an existing file.
- ❖ If the file does exist then it produce error.

## File handling functions: -

1. Fopen() :- it is used to open a specified file in specified mode.
2. Fclose() :- it is used to close a file.
3. Fcloseall() :- it is used to close all the opened files.
4. Fgetc() :- it is used to read a character from a file.
5. Fputc() :- it is used to put/store a character in a file.
6. Fgets() :- it is used to read a string from file.
7. Fputs() :- it is used to put a string in a file.

8. Fread() :- it is used to read a structured information from a file.

9. Fwrite() :- it is used to write a structured information to a file.

10.Fscanf() :- it is used to read any type of data from a file.

11.Fprintf() :- it is used to write any type of data in a file.

12.Feof() :- it is used to test whether the file pointer reached to end of file or not.

13.Ftell() :- it is used to identify the location of a file pointer.

14.Rewind () :- it is used to move the file pointer to the beginning of a file.

15.Fseek() :- it is used to move the file pointer from one location to another location.

File pointer

C language doesnot use any file directly from harddisk.

It always loaded in to memory.

It return the memory address of file.

We need to store this file addresss.

So we need a file pointer.

Syntax:

FILE \*fp;

FILE \*fp;

Here FILE is a pre defined structure. It was defined in stdio.h header file

File error handling

Whenever we ask to open a file then computer opens a file and returns the file address like 100,200,300,etc. if the comouter unable to open the file it return NULL address.

If the file pointer has NULL address means we got error while opening a file.

```
If(fp==null)
{printf("file error");
Exit(0);
}
```

| Function Name and syntax           | Purpose                                           |  |
|------------------------------------|---------------------------------------------------|--|
| fputc(char,fileptr)                | Writes a char to the file                         |  |
| fgetc(fileptr)                     | Reads char from file                              |  |
| putw(int,fileptr)                  | Writes an integer to the file                     |  |
| getw(fileptr)                      | Reads an integer from file                        |  |
| fputs(string,fileptr)              | Writes a string to the file                       |  |
| fgets(strptr,numofchars,fileptr)   | Reads a string from file                          |  |
| fprintf(fileptr,formattedstring)   | Writes formatted data to the file                 |  |
| fscanf(fileptr,formatteddata)      | Reads formatted data from file                    |  |
| fwrite(ptr,size,n,fileptr)         | Writes an entire block to the file                |  |
| fread(ptr,size,n,fileptr)          | Reads an entire block from file                   |  |
| fseek(fileptr,displacement,origin) | To set file pointer at specified position in file |  |
| ftell(fileptr)                     | Returns the current position of filepointer       |  |
| rewind(fileptr)                    | Moves file pointer to beginning of file           |  |

# Declaring a file pointer

- Before opening a file, we have to declare a pointer to a structure of type **FILE** called a file pointer.
- The **FILE** type is a structure that contains information needed for reading and writing a file.
- The information generally includes the operating system's name for the file, the current character position in the file, whether the file is being read or written and so on.

- This structure declares within the header file **stdio.h**.
- The declaration of the file pointer made as follows,

**FILE \*fp;**

**fp** is the name of the file pointer which is declared as a pointer to a structure of type **FILE**.

# Opening a file for creation and edit

- Once the file pointer has been declared, the next step is to open an appropriate file to perform any operation.
- This is done using the function **fopen()**, specified in the header file **stdio.h**.
- To open a file, the syntax is:  
**fp=fopen("filename","mode");**
- The **fopen()** function takes two parameters, both of which are strings.

- The parameter **filename** is the name of the disk file to open.
- The file name generally consists of two parts: a primary name and an optional extension, which separates from the primary name by a period (.).
- The second parameter **mode** specifies how the file is to be opened, *ie*, for reading, writing both reading and writing, appending at the end of the file etc.
- If the call to the **fopen()** function is successful, the function returns a pointer of type **FILE**.
- This pointer must use in subsequent operations on the file, such as reading from or writing.
- If the file cannot open for some reason, **fopen()** returns a NULL pointer. ·

# Closing a file

- When all operations on a file have complete, it must be closed.
- As a result, the file disassociates from the file pointer.
- Although when the program exits, all open files are closed automatically, however explicitly closing a file has the following advantages,
  - Prevents accidental misuse of the file.
  - As there is a limit to the number of files that can keep open simultaneously, closing a file may help open another file.
  - The closed file can again open in a different mode as per requirement.

- In C, a file is closed by the function **fclose()**.
- It has the following syntax,

**fclose(fp);**

- This function accepts a file pointer as an argument and returns a value of type int.
- If the **fclose()** function closes the file successfully, then it returns an integer value **0**. Otherwise, it returns **EOF**.

# **feof()**

**feof()** function is a file handling function in C programming language which is used to find the end of a file.

In a C program, we use **feof()** function as,

**feof(fp);**

where,

**fp** – file pointer

# fwrite()

*The fwrite() function is used to write records (sequence of bytes) to the file.*

*A record may be an array or a structure.*

*Syntax of fwrite() function*

```
fwrite( ptr, int size, int n, FILE *fp );
```

*The fwrite() function takes four arguments.*

**ptr :** *ptr is the reference of an array or a structure stored in memory.*

**size :** *size is the total number of bytes to be written.*

**n :** *n is number of times a record will be written.*

**FILE\* :** *FILE\* is a file where the records will be written in binary mode.*

**fp :** *file pointer*

# fread()

*The fread() function is used to read bytes from the record.  
A record may be an array or a structure.*

*Syntax of fread() function*

```
fread( ptr, int size, int n, FILE *fp );
```

*The fread() function takes four arguments.*

**ptr :** *ptr is the reference of an array or a structure where data will be stored after reading.*

**size :** *size is the total number of bytes to be read from file.*

**n :** *n is number of times a record will be read.*

**FILE\* :** *FILE\* is a file where the records will be read.*

**fp :** *file pointer*

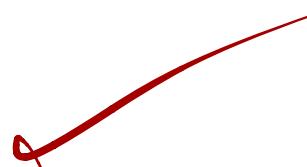
# Random access to files

*Random accessing of files in C language can be done with the help of the following functions –*

1) **ftell ()**



2) **rewind ()**



3) **fseek ()**



# **ftell()**

*It returns the current position of the file pointer.*

*For example,*

```
FILE *fp;
```

```
int n;
```

---

---

---

```
n = ftell (fp);
```

**ftell()** is used for counting the number of characters which are entered into a file.

# **rewind()**

*It makes file pointer move to beginning of the file.*

*The syntax is as follows –*

**rewind (file\_pointer);**

# fseek()

*It is to make the file pointer point to a particular location in a file.*

*The syntax is as follows –*

**fseek(file\_pointer, offset, position);**

## Offset

*The number of positions to be moved while reading or writing. It can be either negative or positive.*

*Positive - forward direction.*

*Negative – backward direction.*

## Position

*It can have three values, which are as follows –*

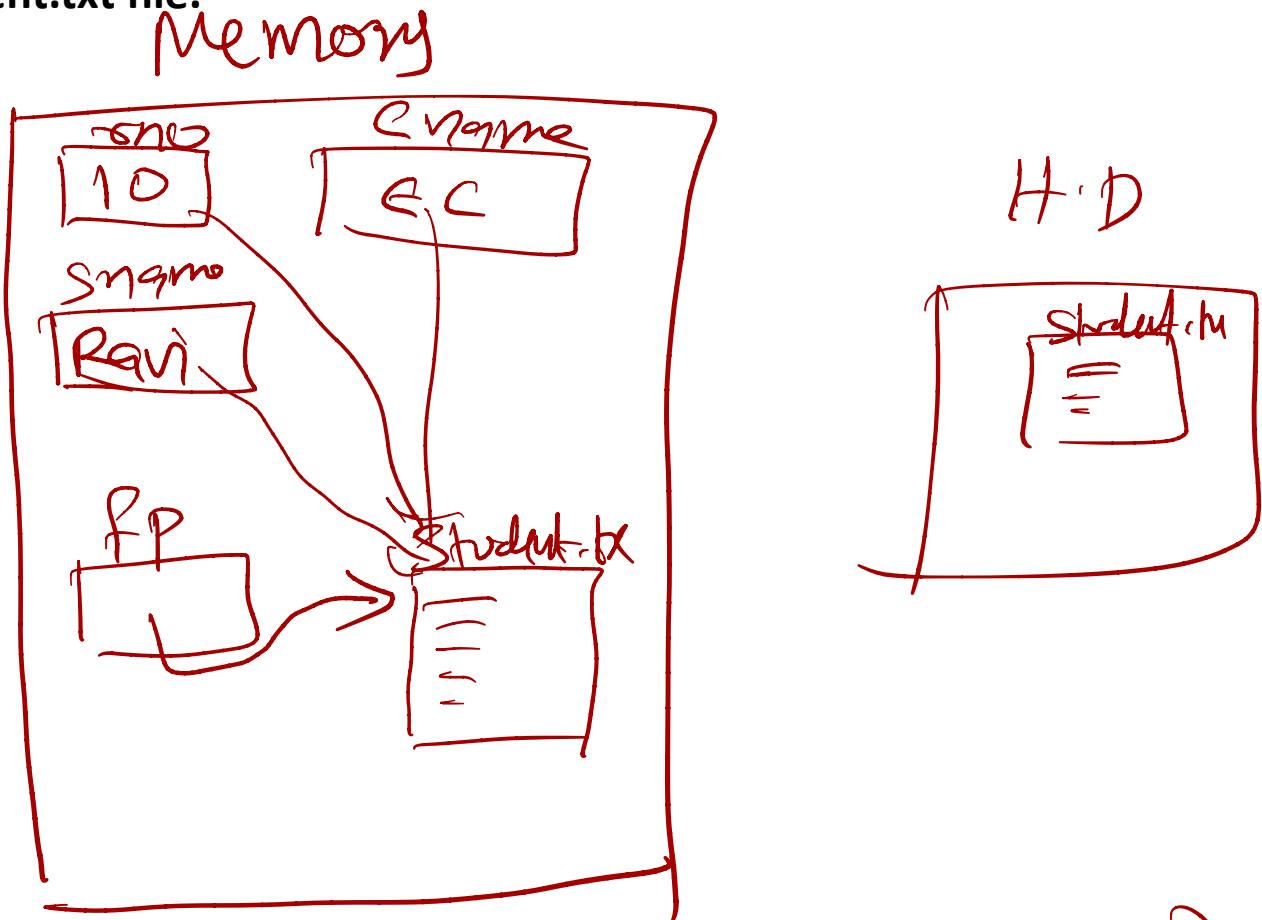
*0 – Beginning of the file.*

*1 – Current position.*

*2 – End of the file.*

1. WAP to print / write rno,sname,cname in student.txt file.

```
#include<stdio.h>
void main()
{
    int rno;
    char sname[100], cname[100];
    FILE *fp;
    fp=fopen("student.txt","w");
    if( fp==NULL )
    {
        printf("file error");
        exit(0);
    }
    printf("Enter rno, sname, cname ");
    scanf("%d %s %s", &rno, &sname, &cname);
    fprintf(fp, "%d %s %s", rno, sname, cname);
    fclose( fp );
    printf("Data successfully write to student.txt file");
}
```

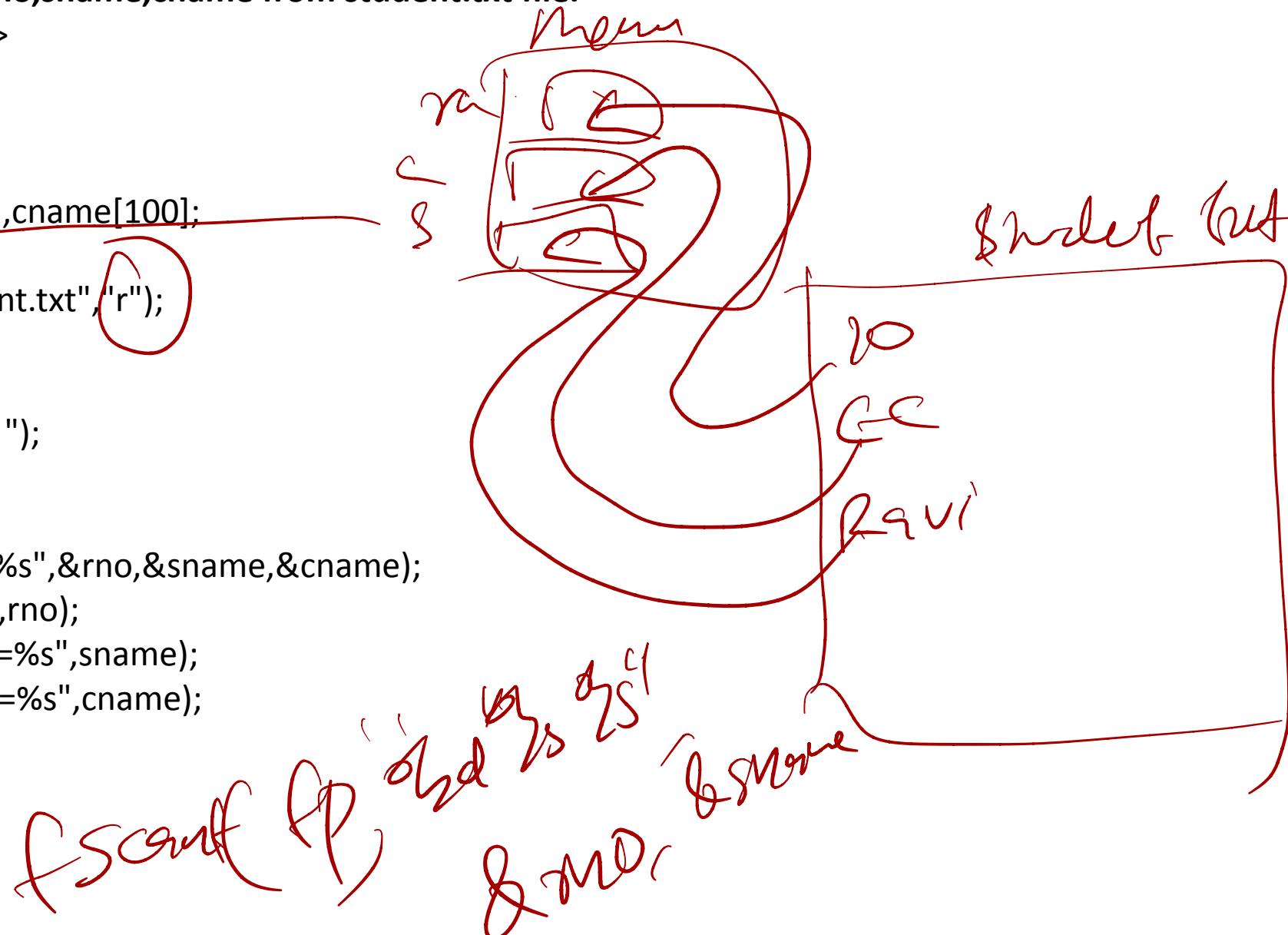


$\text{fprintf}(fp, "%d %s %s", rno, sname, cname)$

✓ ✓ ✓

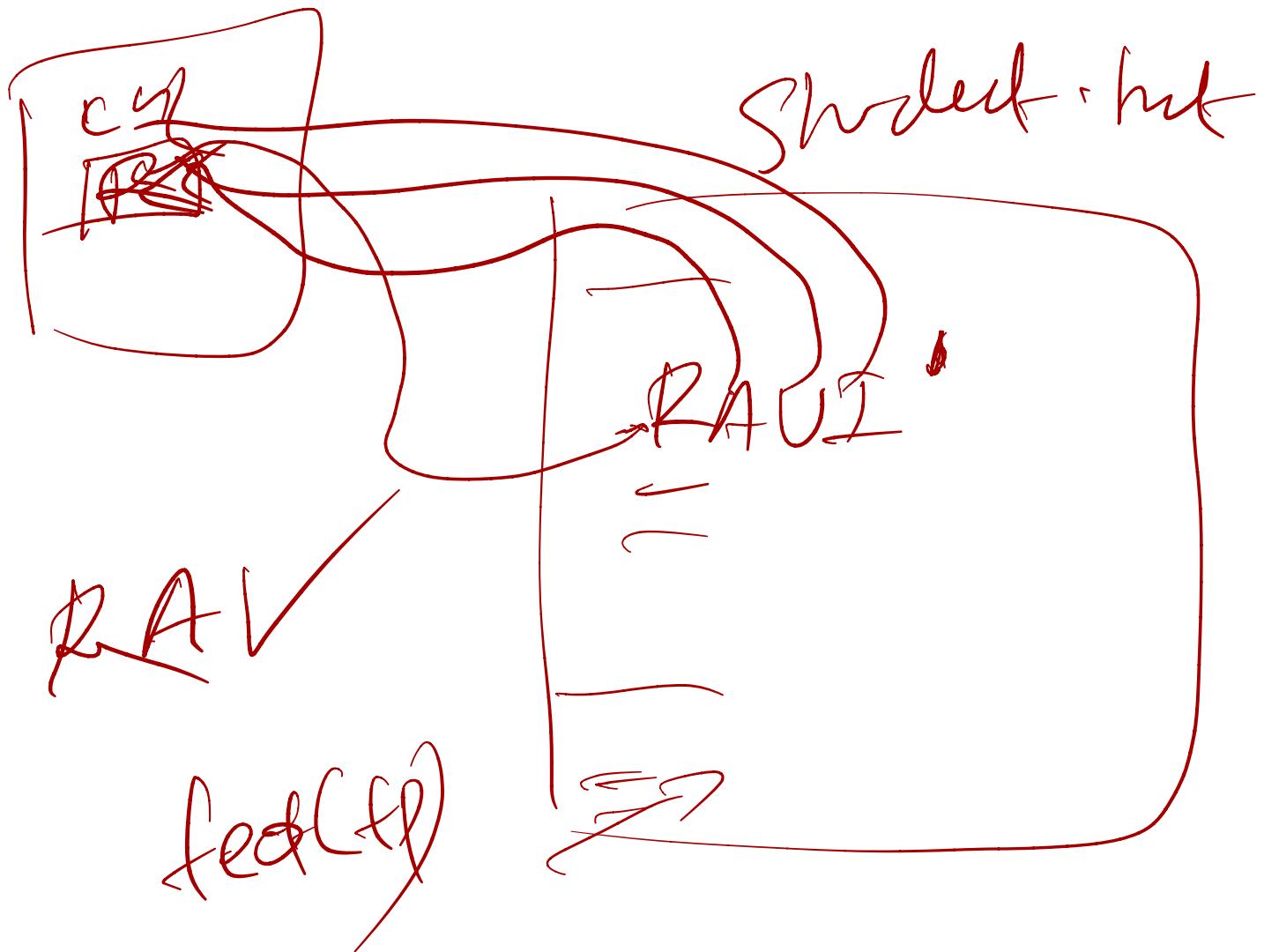
2. WAP to read rno,sname,cname from student.txt file.

```
#include<stdio.h>
void main()
{
    int rno;
    char sname[100],cname[100];
    FILE *fp;
    fp=fopen("student.txt","r");
    if( fp==NULL )
    {
        printf("File error ");
        exit(0);
    }
    fscanf(fp,"%d%s%s",&rno,&sname,&cname);
    printf("Rno=%d",rno);
    printf("\nSname=%s",sname);
    printf("\nCname=%s",cname);
    fclose(fp);
}
```



**WAP to read entire information of a file.**

```
#include<stdio.h>
void main()
{
    char ch;
    FILE *fp;
    fp=fopen("student.txt","r");
    if( fp==NULL )
    {
        printf("File error");
        exit(0);
    }
    while( !feof(fp) )
    {
        fscanf(fp,"%c",&ch);
        if( feof(fp) ) →
            break;
        printf("%c",ch);
    }
    fclose( fp );
}
```



**WAP to copy one file to another file.**

```
#include<stdio.h>
void main()
{
    char ch;
    FILE *fp1,*fp2;
    fp1=fopen("student.txt","r");
    fp2=fopen("dup.txt","w");
    if( fp1==NULL )
    {
        printf("File 1 error");
        exit(0);
    }
    if( fp2==NULL )
    {
        printf("File 2 error");
        exit(0);
    }
    while( !feof( fp1 ) )
    {
        fscanf(fp1,"%c",&ch);
        fprintf(fp2,"%c",ch);
    }
    printf("Student.txt copied to dup.txt");
    fclose(fp1);
    fclose(fp2);
}
```

**WAP to merge 2 files.**

```
#include<stdio.h>
void main()
{
    char ch;
    FILE *fp1,*fp2,*fp3;
    fp1=fopen("Student.txt","r");
    fp2=fopen("Employee.txt","r");
    fp3=fopen("Merged.txt","w");
    if( fp1==NULL )
    {
        printf("File 1 error");
        exit(0);
    }
    if( fp2==NULL )
    {
        printf("File 2 Error ");
        exit(0);
    }
    if( fp3==NULL )
    {
        printf("File 3 error");
        exit(0);
    }
    while( !feof(fp1))
    {
        fscanf(fp1,"%c",&ch);
        fprintf(fp3,"%c",ch);
    }
    while( !feof(fp2))
    {
        fscanf(fp2,"%c",&ch);
        fprintf(fp3,"%c",ch);
    }
    fclose(fp1 );
    fclose(fp2);
    fclose(fp3);
    printf('Student.txt and Employee.txt are merged into Merged.txt');
}
```

**WAP to separated vowels and consonants of a student.txt file.**

```
#include<stdio.h>
void main()
{
    char ch;
    FILE *fp1,*fp2,*fp3;
    fp1=fopen("Student.txt","r");
    fp2=fopen("Vowels.txt","w");
    fp3=fopen("Consonants.txt","w");
    if( fp1==NULL )
    {
        printf("File 1 error");
        exit(0);
    }
    if( fp2==NULL )
    {
        printf("File 2 Error ");
        exit(0);
    }
    if( fp3==NULL )
    {
        printf("File 3 error");
        exit(0);
    }
    while( !feof(fp1))
    {
        fscanf(fp1,"%c",&ch);
        if( ch=='a' || ch=='e' || ch=='i' || ch=='o' || ch=='u' || ch=='A'
        || ch=='E' || ch=='I' || ch=='O' || ch=='U')
            fprintf(fp2,"%c",ch);
        else
            fprintf(fp3,"%c",ch);
    }
    fclose(fp1);
    fclose(fp2);
    fclose(fp3);
    printf("data seperated");
}
```

**Thank You  
Good Luck**