



# Computational

## Thinking for

## Structured

## Design

# Introductory Class

KLEF

(CTSD)

BES-1

# Introduction to Computers



Literally



Modern Computers

More Pervasive in Life

Computers has to do much more than to calculate



Electronic Device



Receives



Input



Stores and Processes

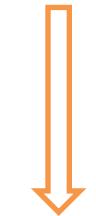


As per the  
instructions  
given

(Programmable)



Output





# Simple Flow of Computer

2 , 3

Addition ie  $(2 + 3)$

5

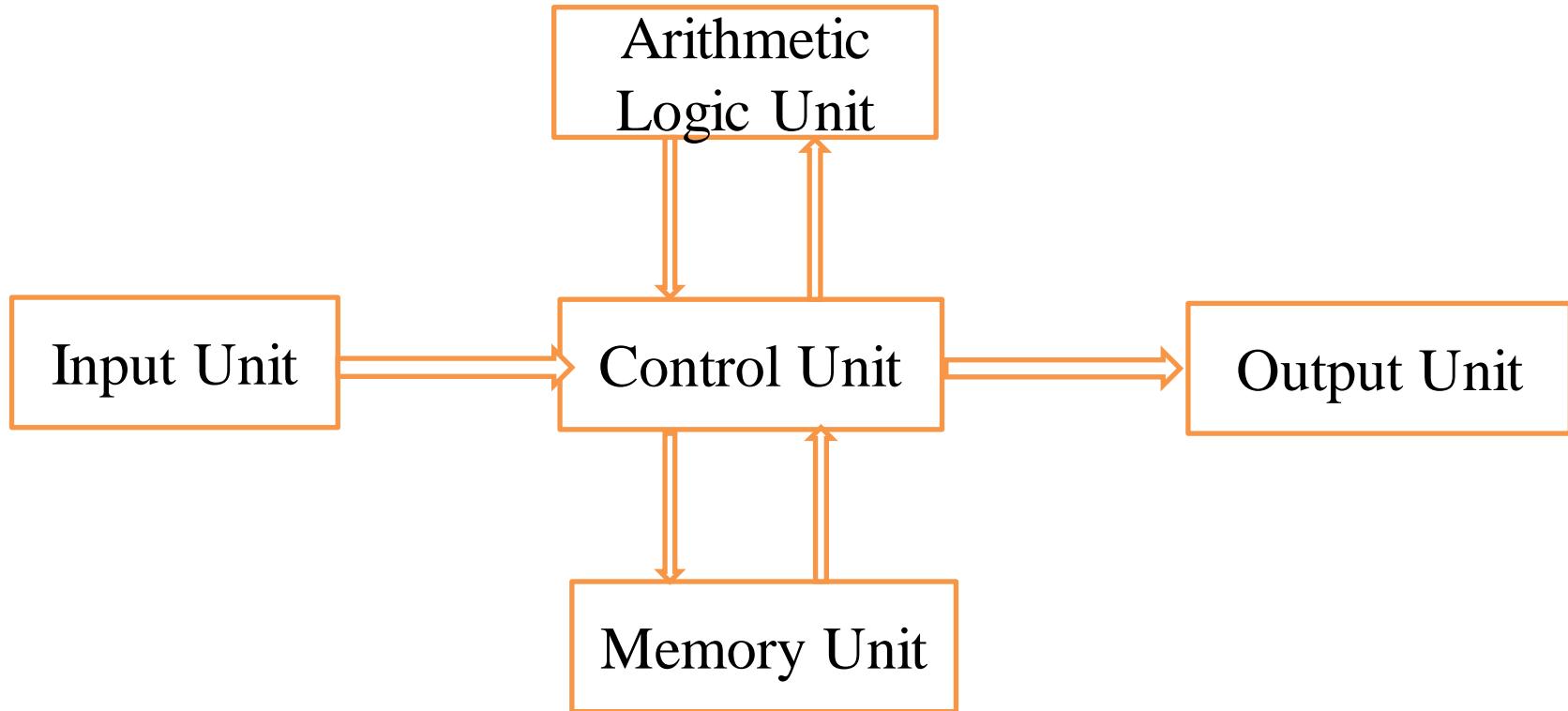


Data

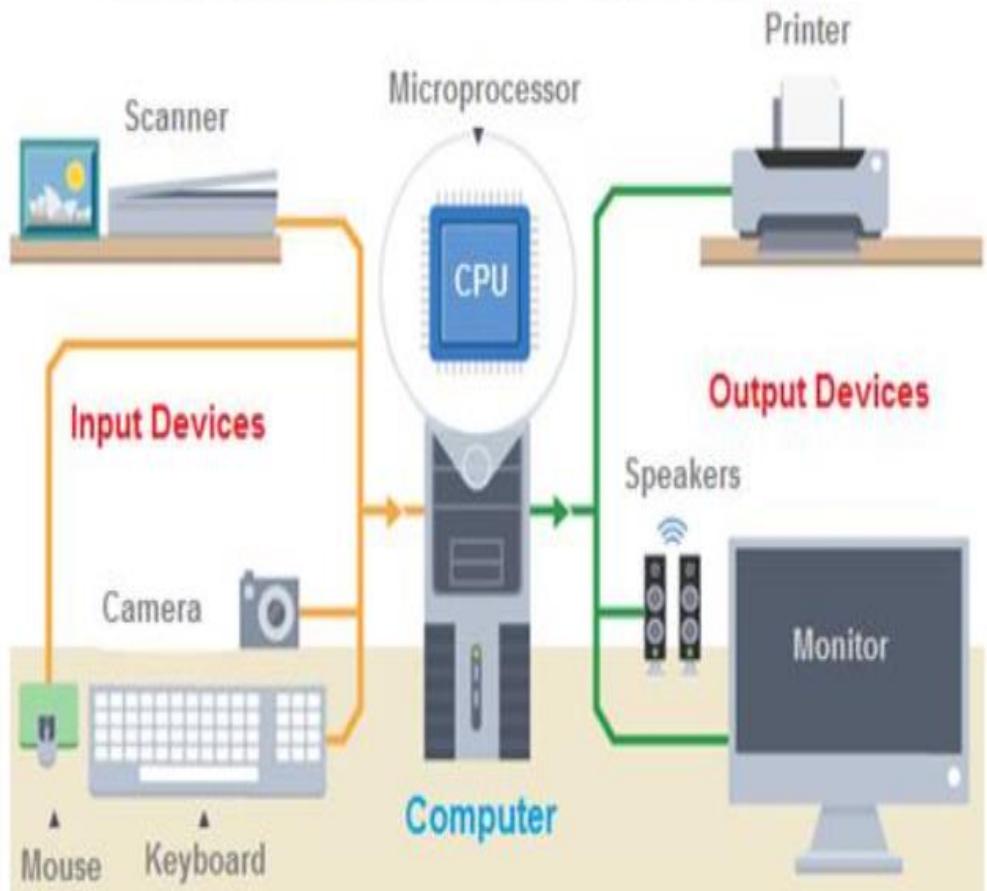
Applies any  
operation

Obtained after  
Processing  
instructions

# Input-Process-Output Model

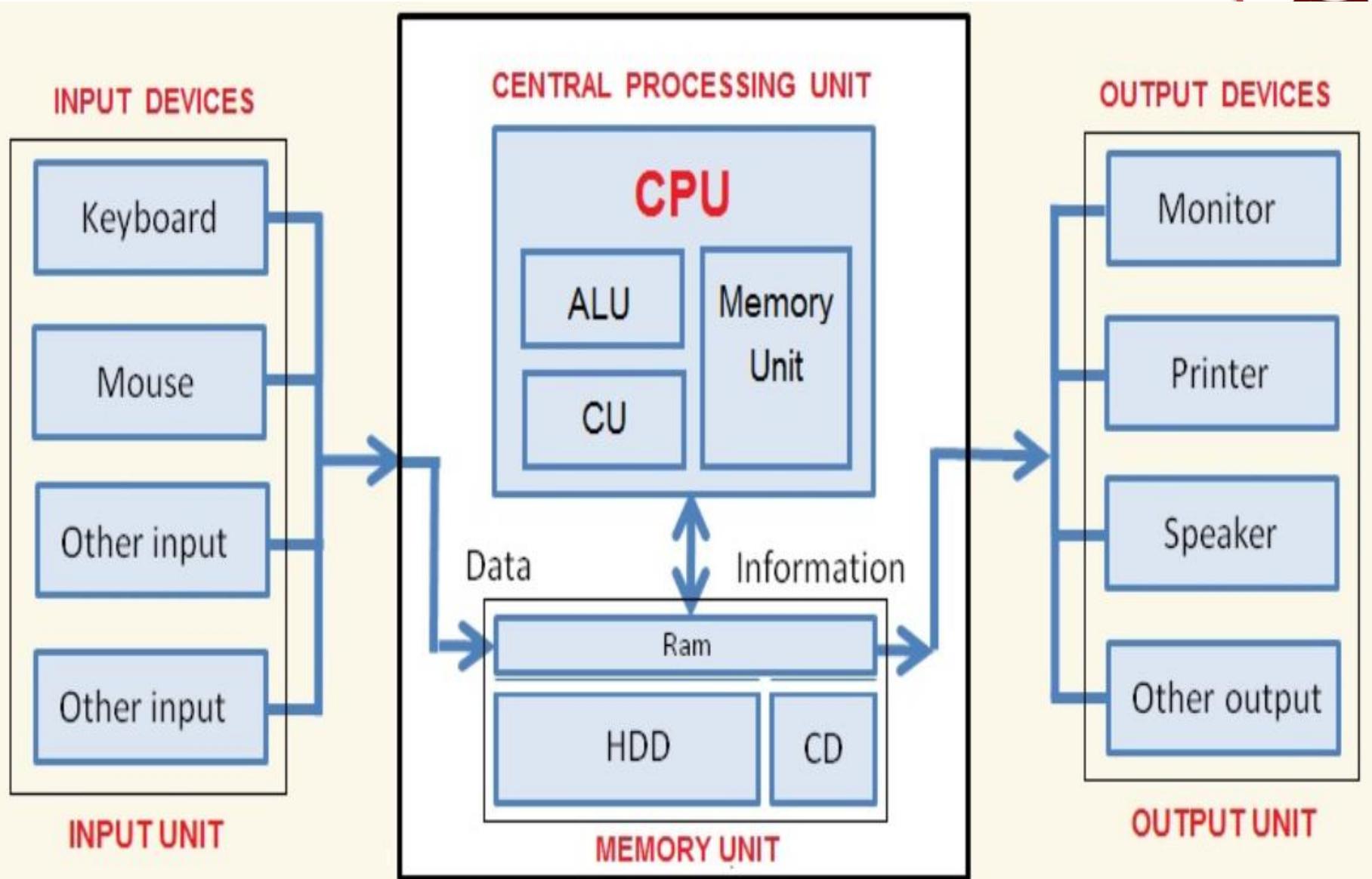


## Computer System - External Devices



## Computer - Internal Components





# Input Devices

**Input data  
and  
Instructions**



**keyboard**



**JoyStick**



**Mouse**

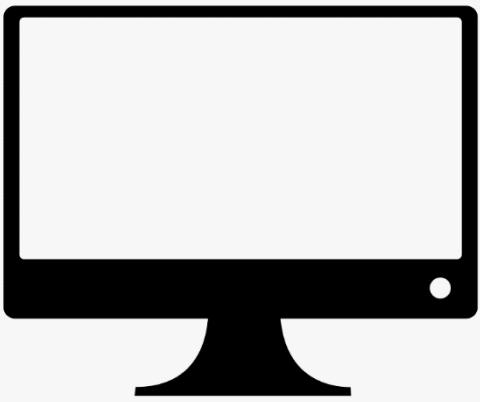


**Web Cam**



**Scanner**

# Output Devices



Monitor



Printer



Speakers



Plotters



Projector

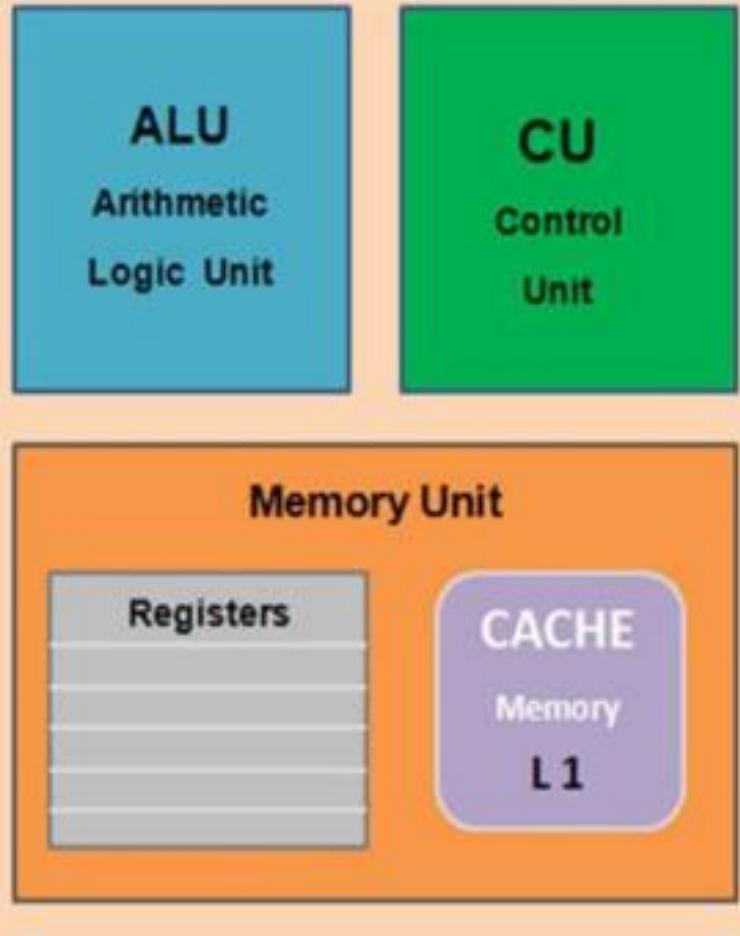


Headsets

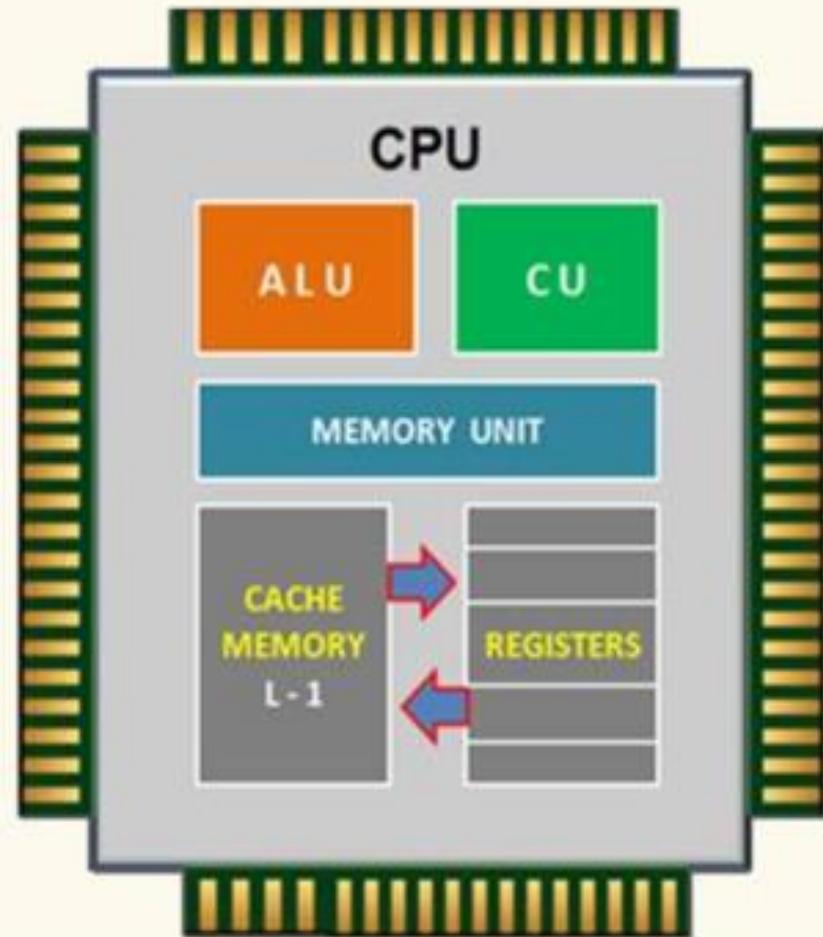
**Device Used to provide information for user in the required format.**



## Central Processing Unit

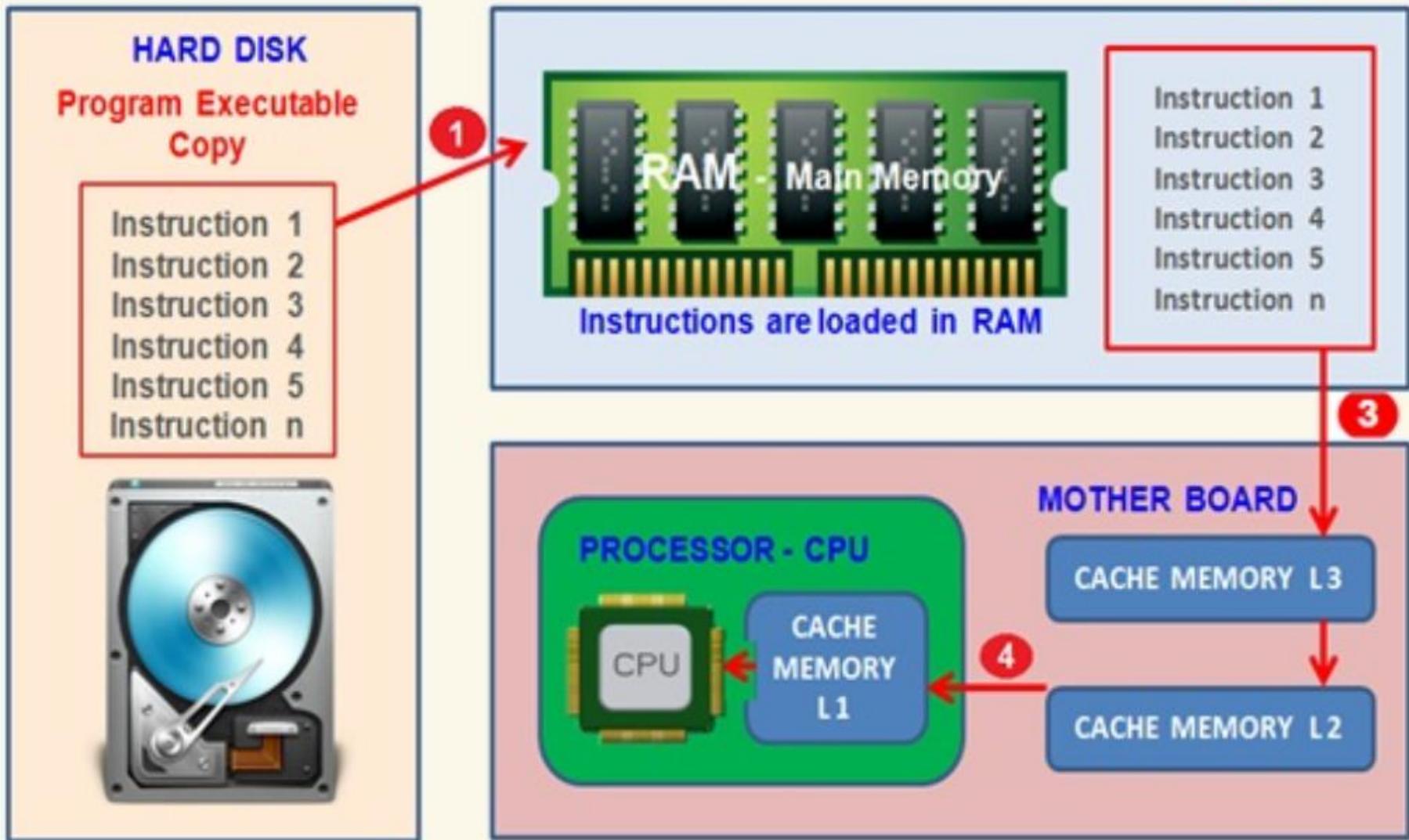


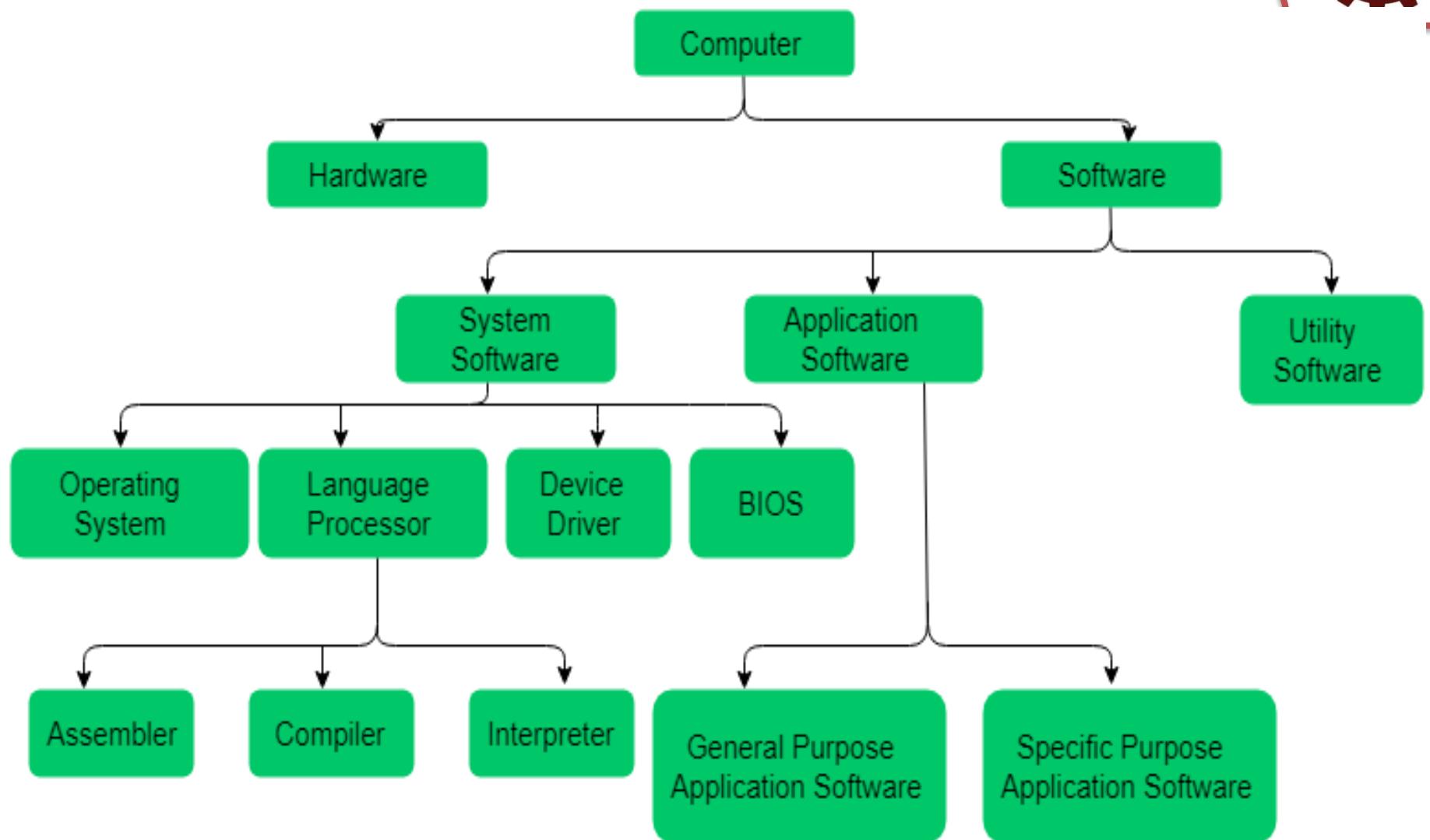
## Central Processing Unit



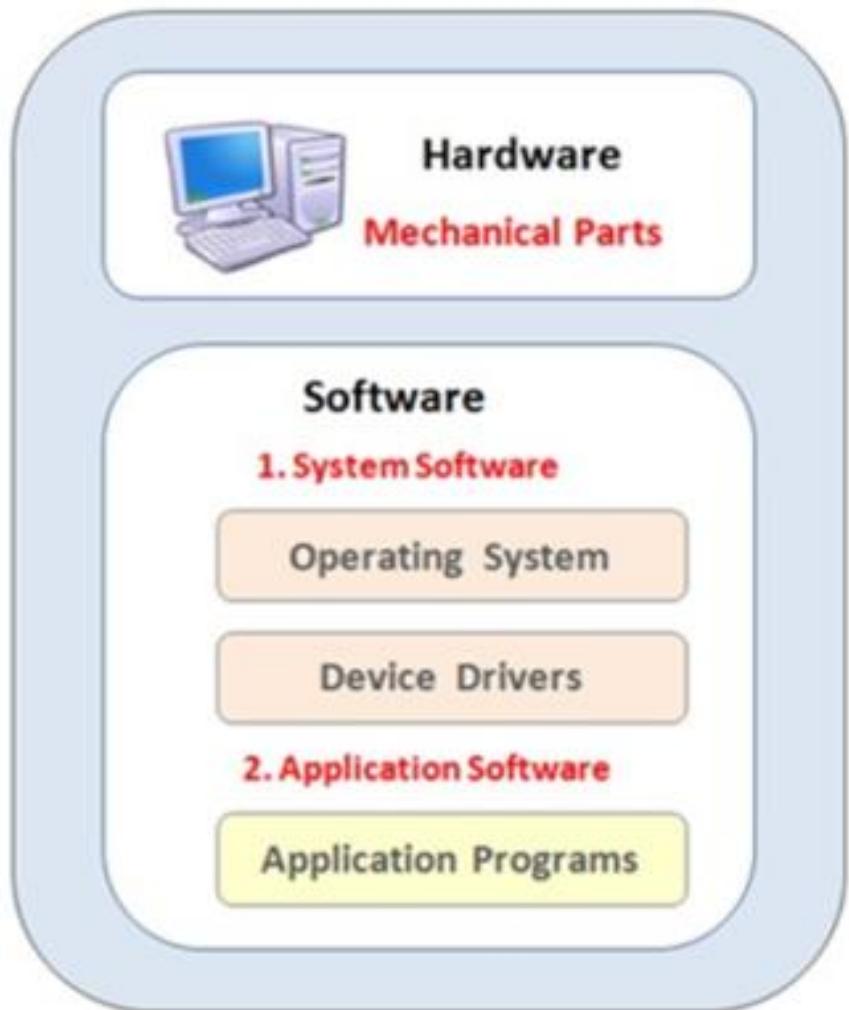
# CPU Executes Program Instructions Step-By-Step

2

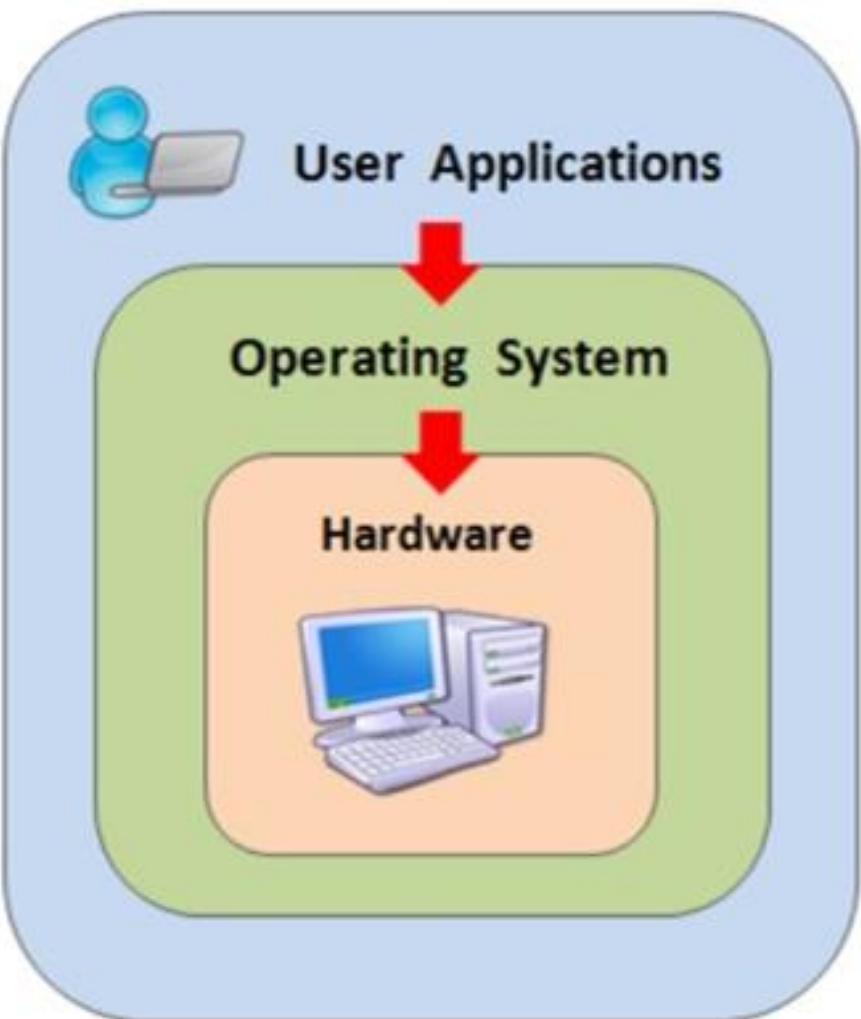




# Computer Components



# Computer Architecture



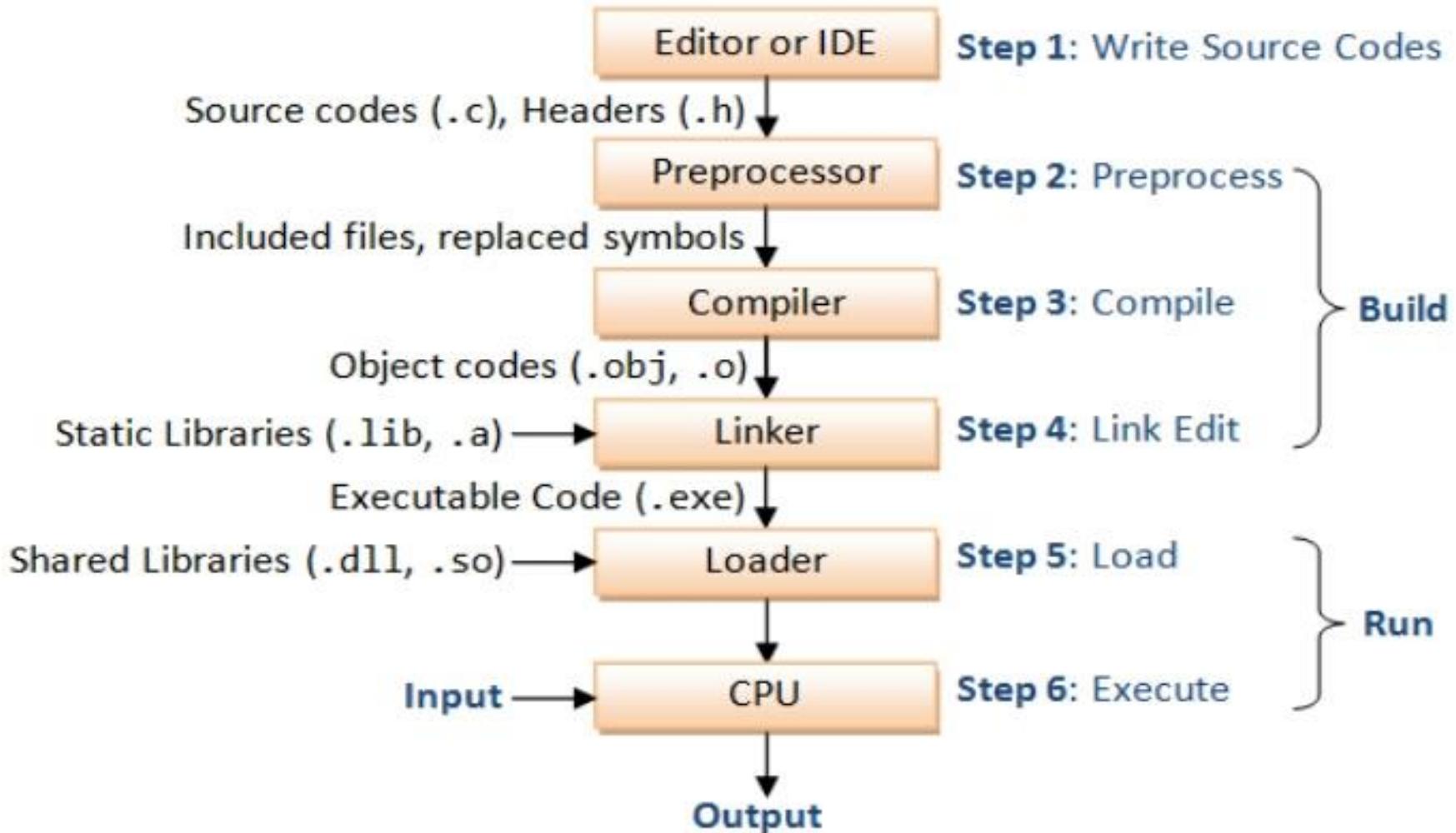
## Application Software



## System Software



# Language Processor

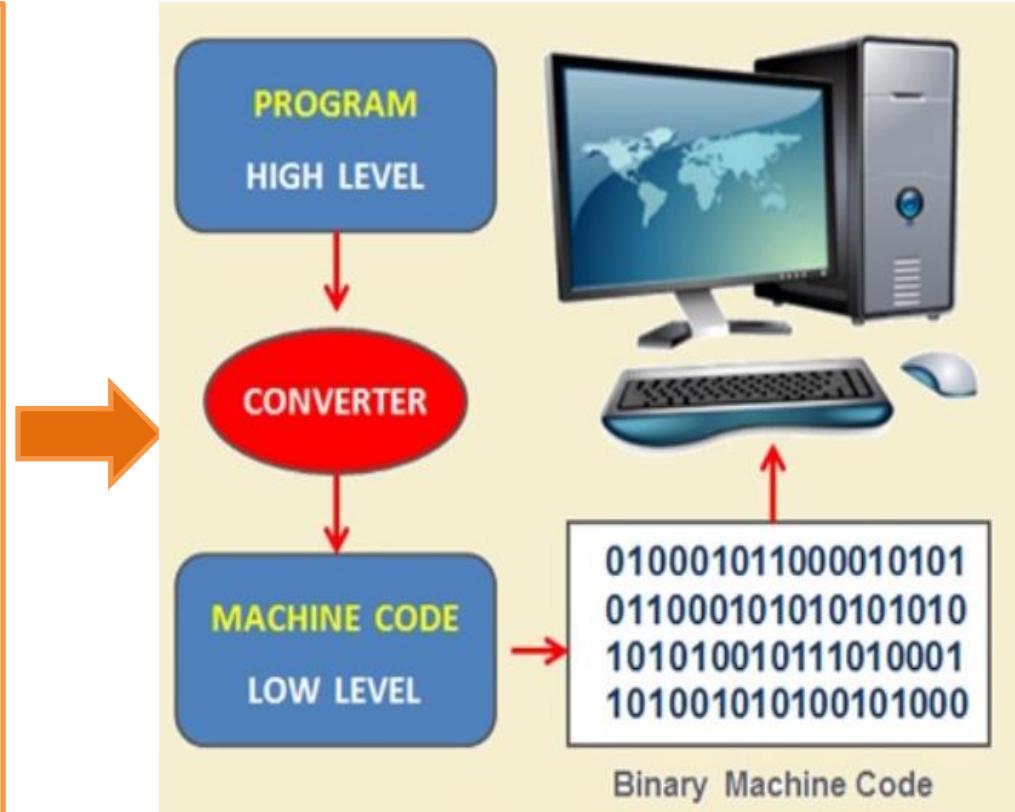


# Computer Program and How it is Translated into System

```

int main()
{
    //variable declaration
    int a,b,sum;
    a =10; //assign a=10;
    b =20; //assign b=20;
    sum=a + b; //assign a+b value to sum
    printf("%d \n",sum);
    return 0;
}//end of the program

```



*C Language is a case sensitive programming Language.*



# Program Needs to be converted into Binary System

System	Base	Digits
Binary	2	0 1
Octal	8	0 1 2 3 4 5 6 7
Decimal	10	0 1 2 3 4 5 6 7 8 9
Hexadecimal	16	0 1 2 3 4 5 6 7 8 9 A B C D E F

Computer use binary number system because the processor inside the computer can execute only binary commands.



# Computational

## Thinking for

## Structured

## Design

# Introduction to C

KLEF

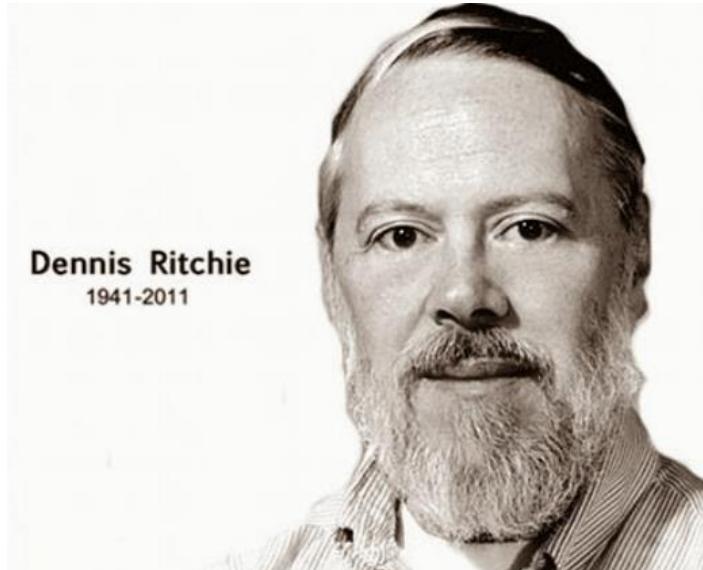
(CTSD)

BES-1



# Introduction to C

General Purpose  
High Level  
Language



**BCPL**

**B**

**1972**

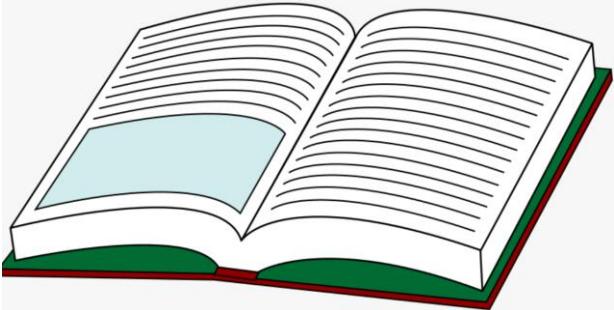
**C**

**KLEF**

**(CTSD)**

**BES-1**

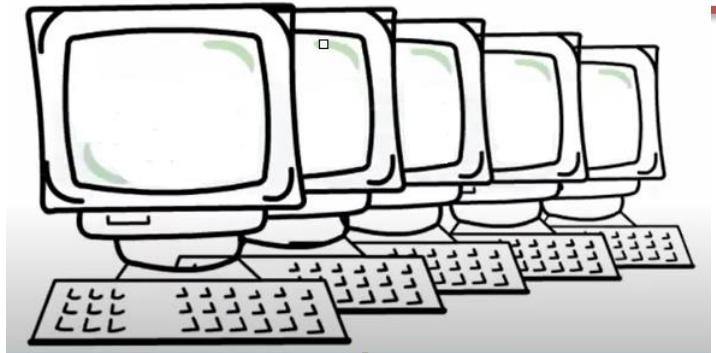
# C Became So Popular



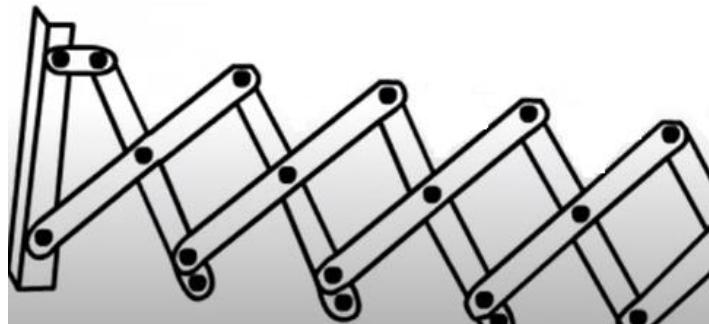
Easy to Learn



Structured Language



Complied on Variety of  
Computers & also Faster  
Execution



Extensible



# Uses of C



Developed for System Development particularly for Operating System

## Examples where C is Used

Operating System

Assemblers

Network Drivers

Language Compliers

Data Bases Systems

Application Programs

Text Editors



# First Program in C

.c

"Hello.c"

Program

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

```
int main()
```

```
{
```

```
printf("HelloWorld");
```

```
return 0;
```

```
}
```

Program

```
#include<stdio.h>
main()
{
    printf("HelloWorld");
}
```

- ❖ Scope of main has started
- ❖ Marks the beginning of main function
- ❖ All the statements are enclosed in braces

- ❖ Pre-processor Directive
- ❖ This statement has to be included in all programs
- ❖ Informs complier to include Files
- ❖ stdio → headerfile → printf
  
- ❖ main() denotes the function
- ❖ Every C program should have a main function
- ❖ Denotes the start of the program
- ❖ printf is a library function which prints what ever the user requires



1972

BCPL and B

Introduction  
to C

C Program

```
Program
#include<stdio.h>
main()
{
    printf("HelloWorld");
}
```

Reasons why  
C Became  
Popular



# Computational Thinking for Structured Design **21SC1101**

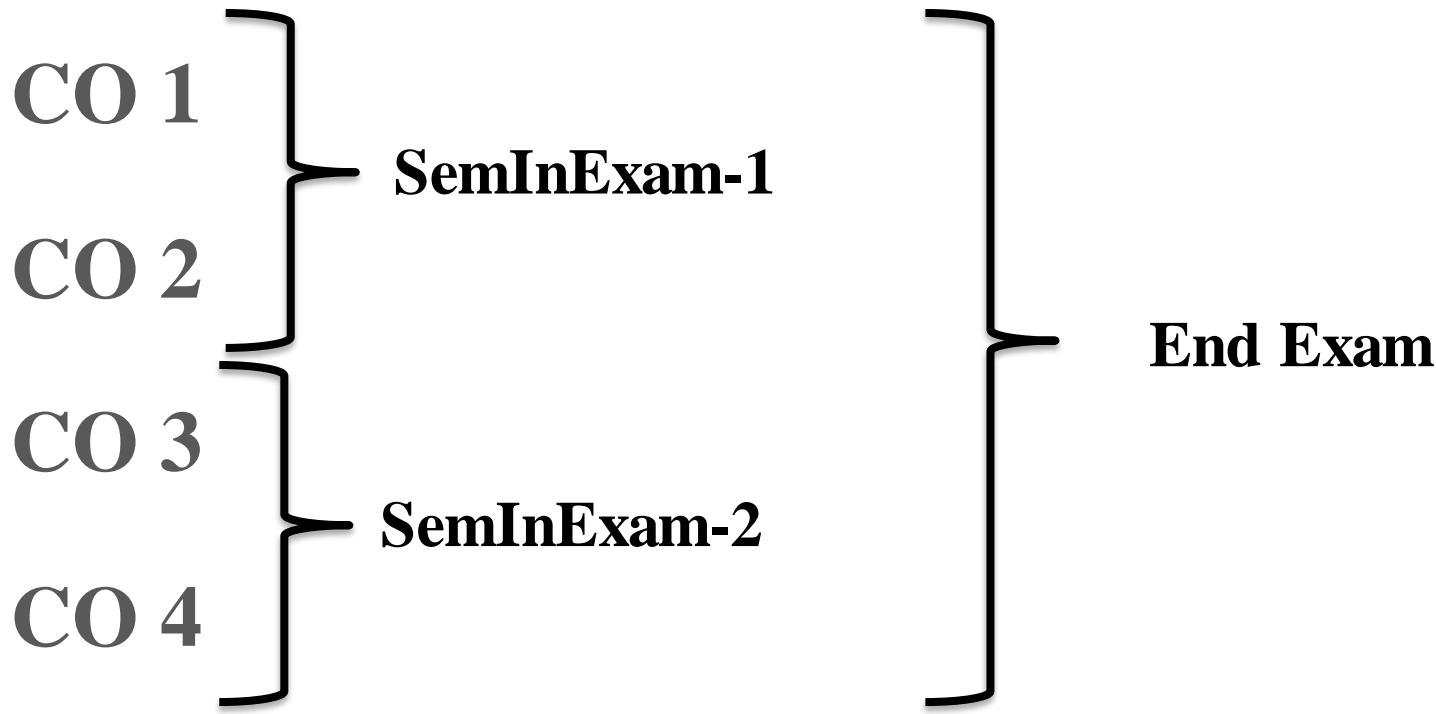
***KLEF***

***(CTSD)***

***BES-1***



# Course Road Map



**INTERNAL  
ASSESSMENT**

**60**

**EXTERNAL  
ASSESSMENT**

**40**



End Sem	Skill	50	6	
Summative (External)	End Sem	100	24	40
	Lab	50	10	

In Sem	SIE-1	50	12	
Summative	SIE-1	50	12	
	LIE	50	8	
	SkillIE	50	4	

In Sem Formmmative	GlobalPlatform	50	4	24
	SkillContinous	50	4	
	ALM	50	5	
	HA	50	5	
	LabContinous	50	6	

Minimum of 85% Attendance



- Course Title : **(CTSD)**
- Course Code : **21SC1101**
- L-T-P-S Structure : **3-0-2-6**
- Credits : **5.5**
- Pre-requisite : **Problem Solving Ability**  
**Logical Thinking**



**Text Books :** 1. Brian W. Kernighan, Dennis M. Ritchie, "The C Programming Language: ANSI C Version", 2/e, Prentice-Hall/Pearson Education-2005. 2. E. Balagurusamy, "Programming in ANSI C" 4thed., Tata McGraw-Hill Education, 2008. 3. R. F. Gilberg, B. A. Forouzan, "Data Structures", 2nd Edition, Thomson India Edition-2005.

**Reference Books :** 1. Mark Allen weiss, Data Structures and Algorithm Analysis in C, 2008, Third Edition, Pearson Education. 2. Horowitz, Sahni, Anderson Freed, "Fundamentals of Data structures in C", 2nd Edition-2007. 3. Robert Kruse, C. L. Tondo, Bruce Leung, Shashi Mogalla, "Data structures and Program Design in C", 4th Edition-2007. 4. C for Engineers and Scientists – An Interpretive Approach by Harry H. Cheng, Mc Graw Hill International Edition-2010. 5. Jeri R. Hanly, Elliot B. Koffman, "Problem Solving and Program Design in C", 7/e, Pearson Education-2004. 6. Jean Paul Tremblay Paul G.Sorenson, "An Introduction to Data Structures with applications", 2nd Edition.



## LAPTOP IS REQUIRED FOR CLASSES

### Laptop Specification:

- Processor : i-5 10 gen iris plus graphics integrated processor, (min of 6 MB cache)
- RAM : 8 GB DDR4 , Bluetooth : 5.0, OS : Windows 10 single language edition
- Hard disk : 1TB HDD with OPTANE Memory cache(16 GB/32G)
- Display : 14"/15" HD (minimum) Matty type.
- Wireless :802.11 ac wave2 type (minimum)

Specifications



# Computational

## Thinking for

## Structured

## Design

### Introduction to Structured Programming



# Structured Programming

## Programming Style

**QUALITY**

**CLARITY**

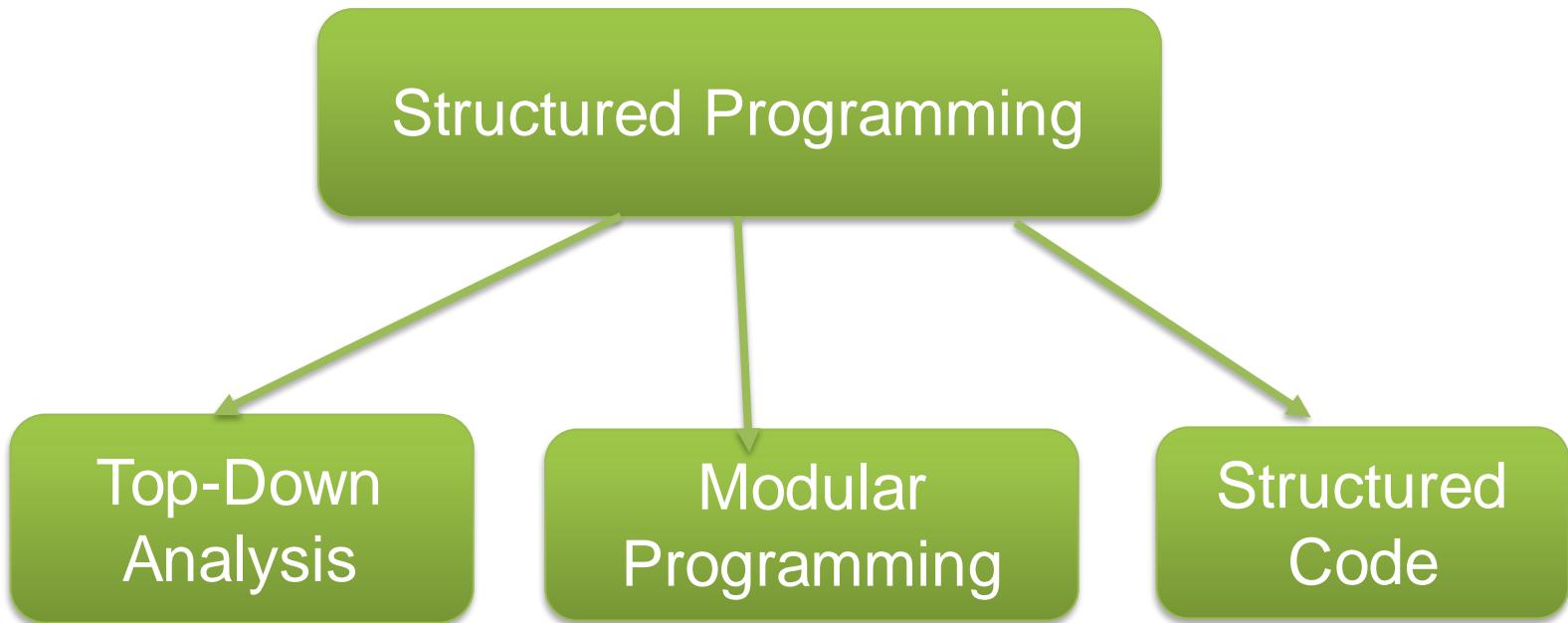
**DEVELOPMENT TIME**

**BLOCK STRUCTURES**

**KLEF**

**(CTSD)**

**BES-1**

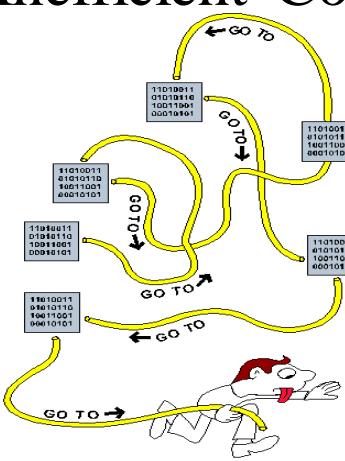


- Better Way to Program
- Systematic Organization

```
main()
{
    ...
    ...
    ...
    goto X
    ...
}
```



Inefficient Code



SpaghettiCode



# Introduction to Structured Programming

## Structured Programming

□ Top-Down Analysis

□ Modular Programming

□ Structured Code



## Top-Down Analysis



Start A yellow star-shaped lightbulb with rays emanating from it, symbolizing an idea or start.



**How to proceed?**



# Top-Down Analysis

Method

Program Analysis

Analyse the Solution

Approach

```
main()
{
    ...
    ...
    ...
}
```



```
main()
{
    ...
}
fun1()
{
    ...
}
fun2()
{
    ...
}
```



# Idea

- Sub Division of Program
- Analyse and Appropriate Solution is generated

## Advantages

- Reduces Problem Solving
- Not Limited to Particular type of Program



## Modular Programming

**Program**

**Solution**

**Larger Problems**

**Developing Solution**

**Complicated**





# Designing Style

Independent Units

Modules

Functions

Variables

Source Code

control

```
main()
{
    ...
}

function1()
{
    ...
}

function2()
{
    ...
}
```

Entry point

return



# Contd. Modular Programming

Solution  
Larger Problems

Which are  
difficult to Program

## Advantages

Small Modules  
are Manageable

Saves  
Production Time

Reusable  
Code

# Structured Code

Old  
Languages



Unstructured  
Code

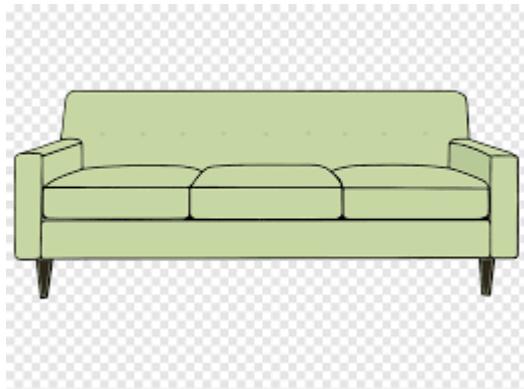
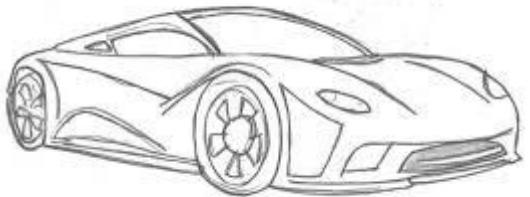


Well  
Organised  
Code



# Advantages

- Improves Problem Solving
- Organisation of Program
- Generalisation of Program Methodology
- Clear Structure & Description
- Easily Modifiable



**KLEF**

**(CTSD)**

**BES-1**

Parts are manufactured separately

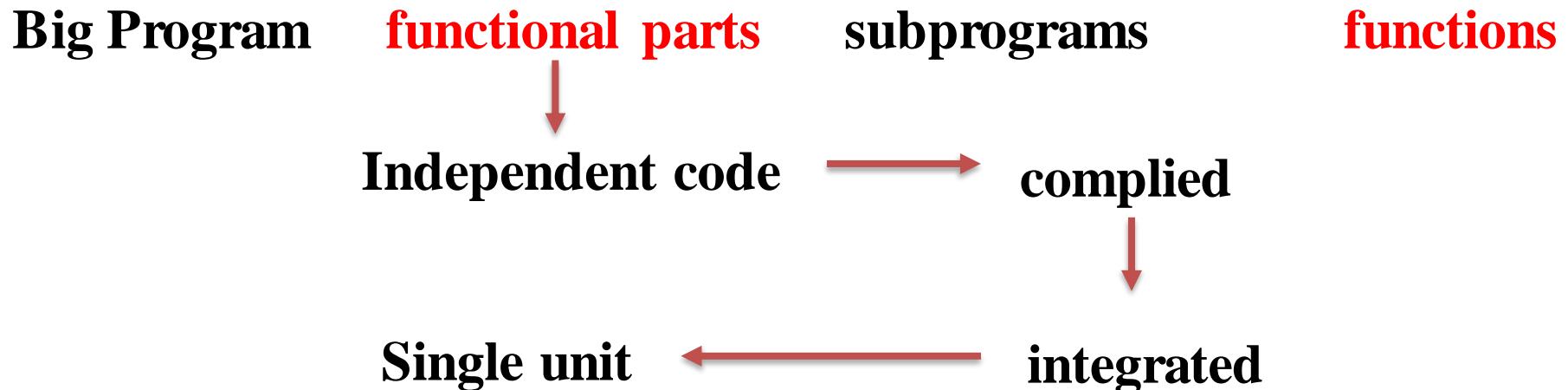


Then shipped and assembled later





- ❖ Every C Program must have main function and any no.of other user defined functions.
- ❖ Program large complex
- ❖ Difficult understand debug test
- ❖ Difficult identify logicalerrors
- ❖ Repetitions of same set of statements a no of times within program.
- ❖ Difficult to update the program.





# Computational Thinking for Structured Design

Introduction to  
**Algorithms & Flow Charts**

*KLEEE*

*(CTSD)*

*BES-1*

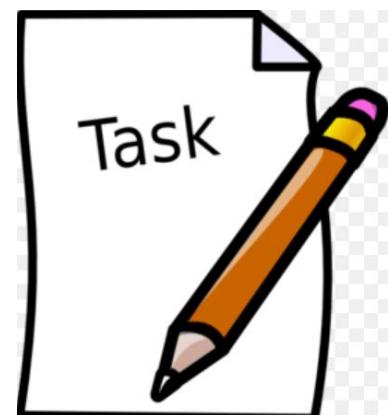
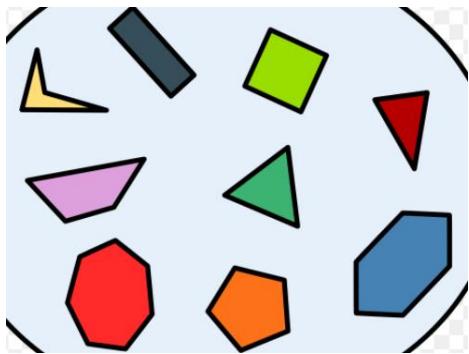


# *ALGORITHM*



of

complete



*KLEF*

*(CTSD)*

*BES-1*



In general whenever we come across a problem we follow some procedure to solve that problem

PROBLEM

PROCEDURE

SOLVE

The step by step procedure what we follow to solve a problem is called as an algorithm

INPUT

PROCESS

OUTPUT



# Simple Flow of Computer

2 , 3

Addition ie  $(2 + 3)$

5



Data

Applies any  
operation

Obtained after  
Processing  
instructions

Not only in computer in general life also we use algorithms to do any work

Example: If you see back side of a noodle packet you can see step wise procedure which says how to cook noodles. That's nothing but an algorithm





# Example: Withdraw cash from ATM

- 1. Go to the ATM**
- 2. Insert your card into  
the machine**
- 3. Press in your code**
- 4. Choose “Withdraw”  
and enter Amount  
required**
- 5. Take the cash, slip and  
card.**



# Need of Algorithms:



1. To understand the basic idea of the problem.



2. To find an approach to solve the problem.

3. The Algorithm gives a clear description of requirements and goal of the problem to the designer.



4. To understand the flow of the problem.

5. We can measure and analyze the complexity (time and space) of the problems concerning input size without implementing and running it; it will reduce the cost of design.



$$f=m*a$$

Identify the Variables

m

a

f

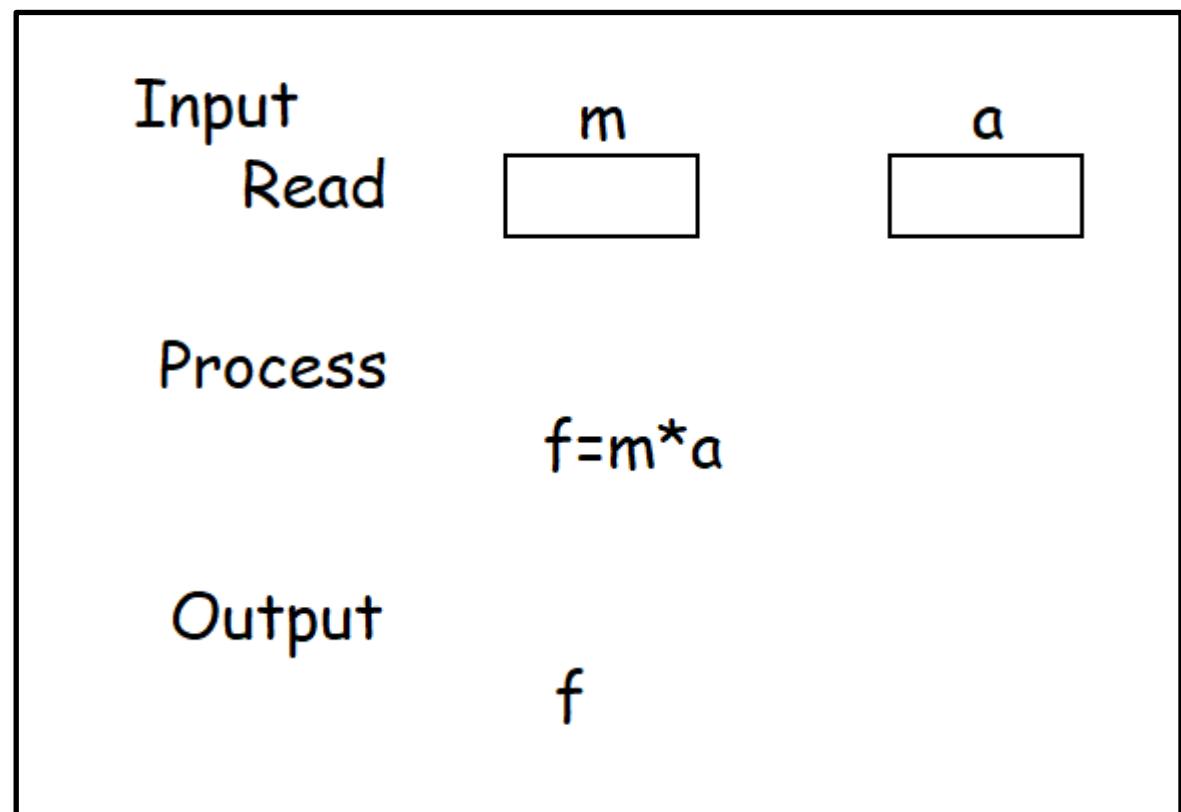
Step 1: Start

Step 2: Read m,a

Step 3: calculate  $f=m*a$

Step 4: Print f

Step 5: Stop





Design an algorithm for the Following equations

$$EQ = a^2 + b^2 + 2ab$$

**Step1:Start**

**Step2:Read a,b  
Step3:Calculate**

$$EQ = (a*a) + (b*b) + (2*a*b)$$

**Step4:Display EQ**

**Step5:Stop**

**Input**

? a ,b

**Process**

$$EQ = (a*a) + (b*b) + (2*a*b)$$

**Output**

? EQ



# Flow Charts

- Pictorial representation of an algorithm
  
- Uses a set of symbols for graphical representation.

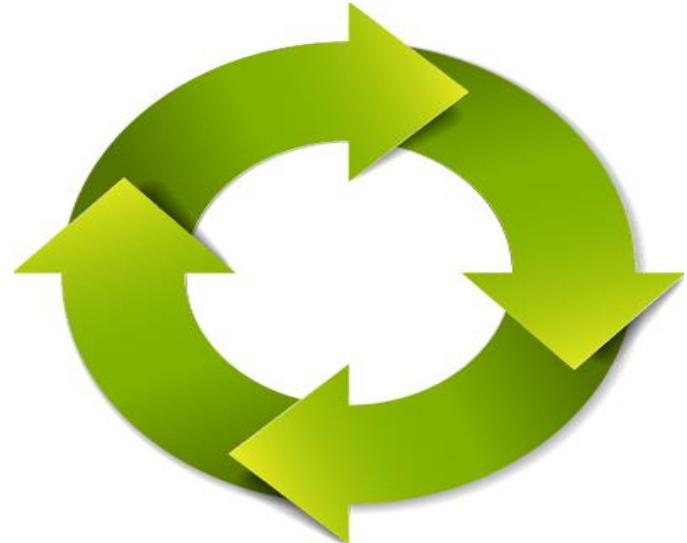
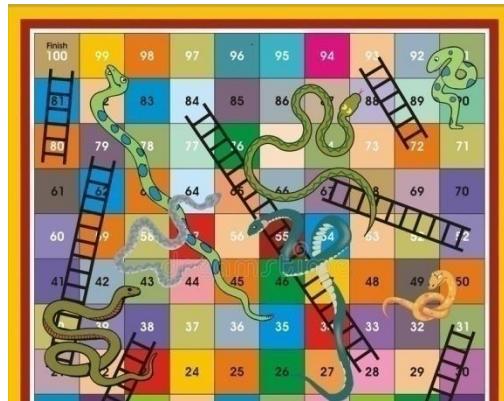
# Symbols used in flow chart



Symbol	Purpose
	Start/Stop
	<u>Input/Output</u>
	Processing/Computation
	Decision Making
	Connecting arrow
	Connector

# Types of Algorithm & Flowcharts Constructs

Sequence





## Algorithm

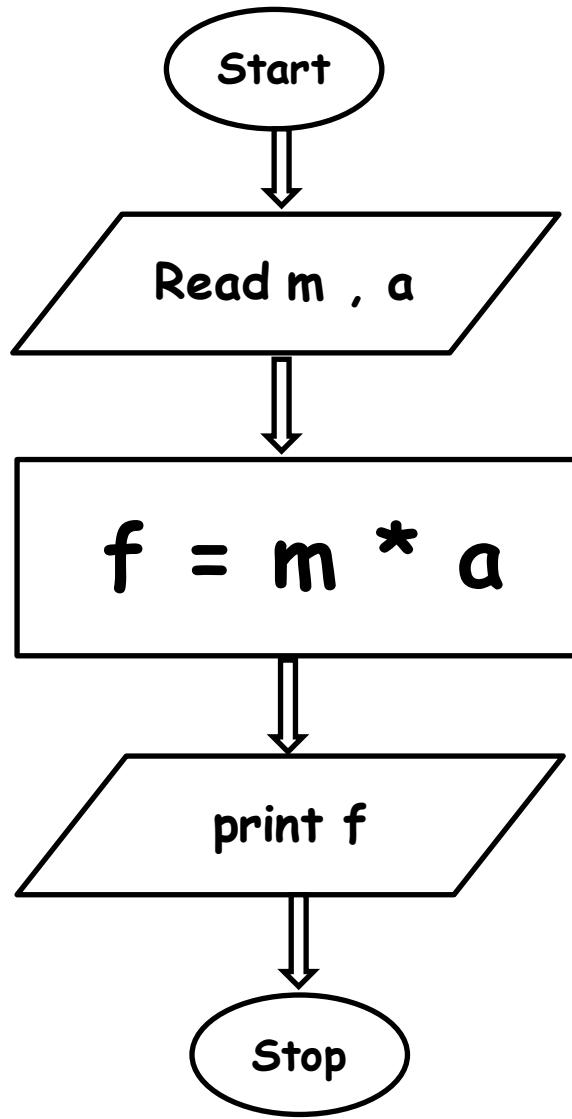
Step 1: Start

Step 2: Read m, a

Step 3: calculate  $f = m * a$

Step 4: Print f

Step 5: Stop



# Area of Rectangle Flow Chart



$aor=l*b$

**Input :l and b**

**Process:**

$aor=l*b$

**Output: aor**

**Step 1:Start**

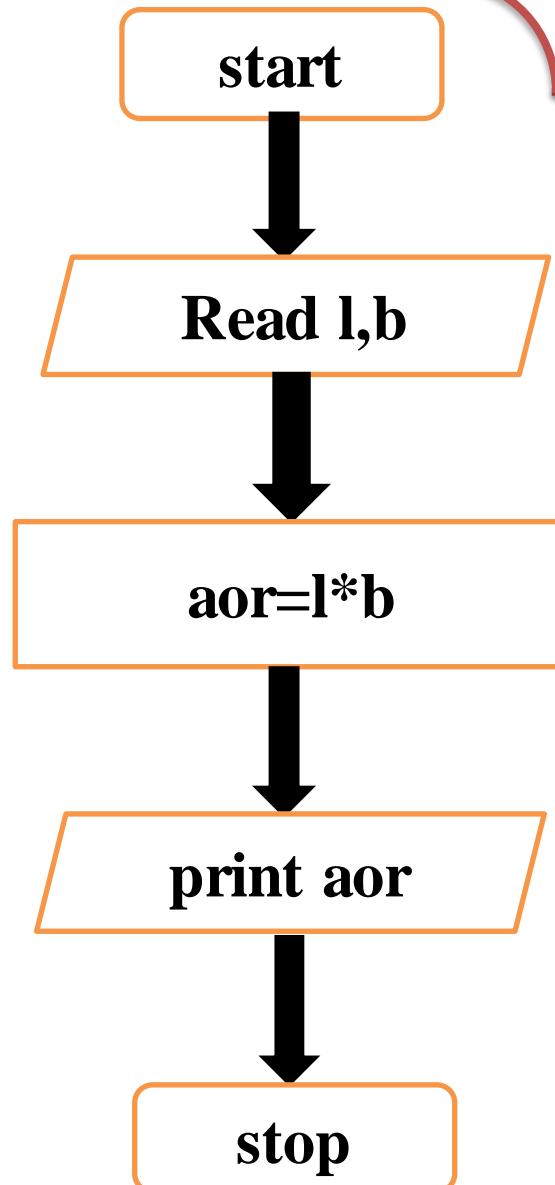
**Step 2:Read l,b**

**Step 3:Calculate**

$aor=l*b$

**Step 4:Display aor**

**Step 5:Stop**





## 2. Design an algorithm for the Following equations

$$EQ = a^2 + b^2 + 2ab$$

**Step1:Start**

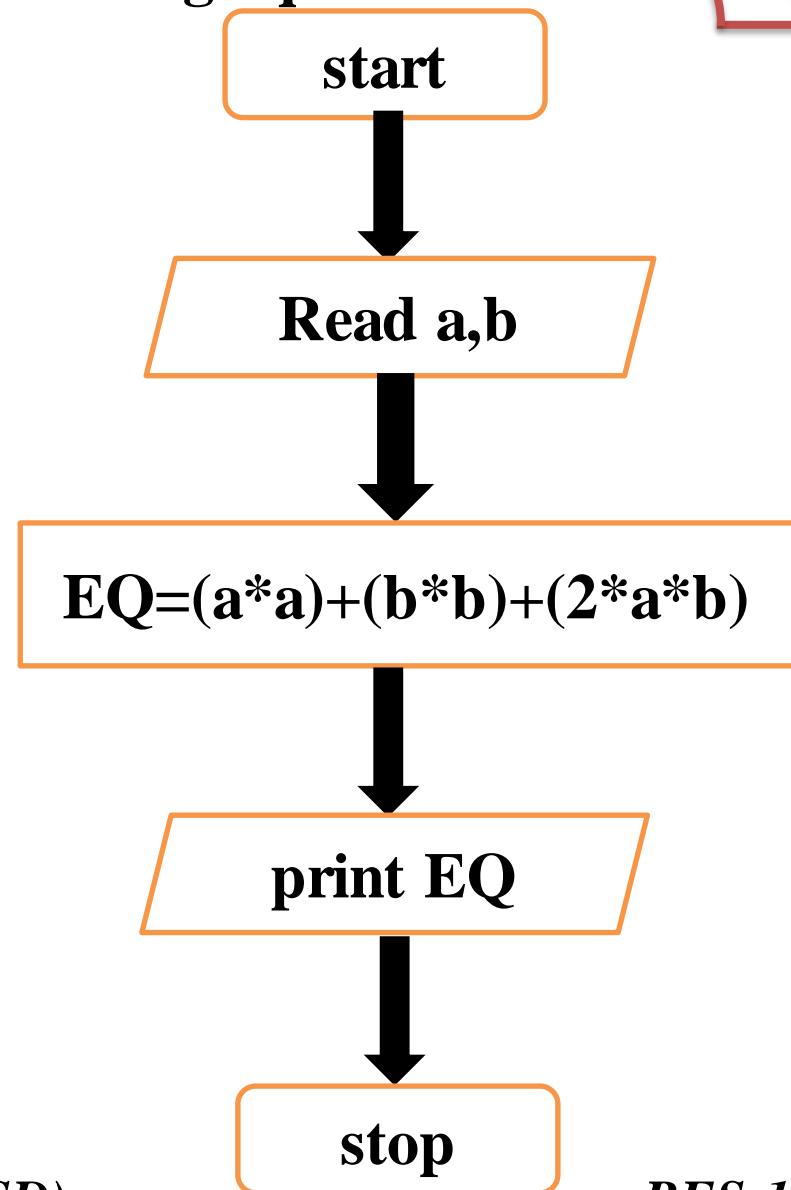
**Step2:Read a,b**

**Step3:Calculate**

$$EQ = (a*a) + (b*b) + (2*a*b)$$

**Step4:Display EQ**

**Step5:Stop**





$$Eq = a^2 + b^2$$

Areaofcircle=3.14\*r<sup>2</sup>

SimpleInterest=(p\*t\*r)/100

Direct Implementation



Sequence Algorithms

# Selection (Decision Making)

- Applicable when a problem has more than one outcome.
- The outcome is based on the input



- Example
  - Greatest of two numbers
  - Check if the number is even or odd



# Selection Algorithm - An Example

Algorithm to print greatest of two numbers

- **Input-** two integers a and b
- **Process-** compare the two values a and b
- **Output-** Display the greatest number



# To print greatest of two numbers

Step 1    **start**

Step 2    **input a and b**

Step 3    **if a is greater than b then**

**3.1 print a is greatest**

**else**

**3.2 print b is greatest**

Step 4    **stop**



## Step1:Start

Step2:Read a,b

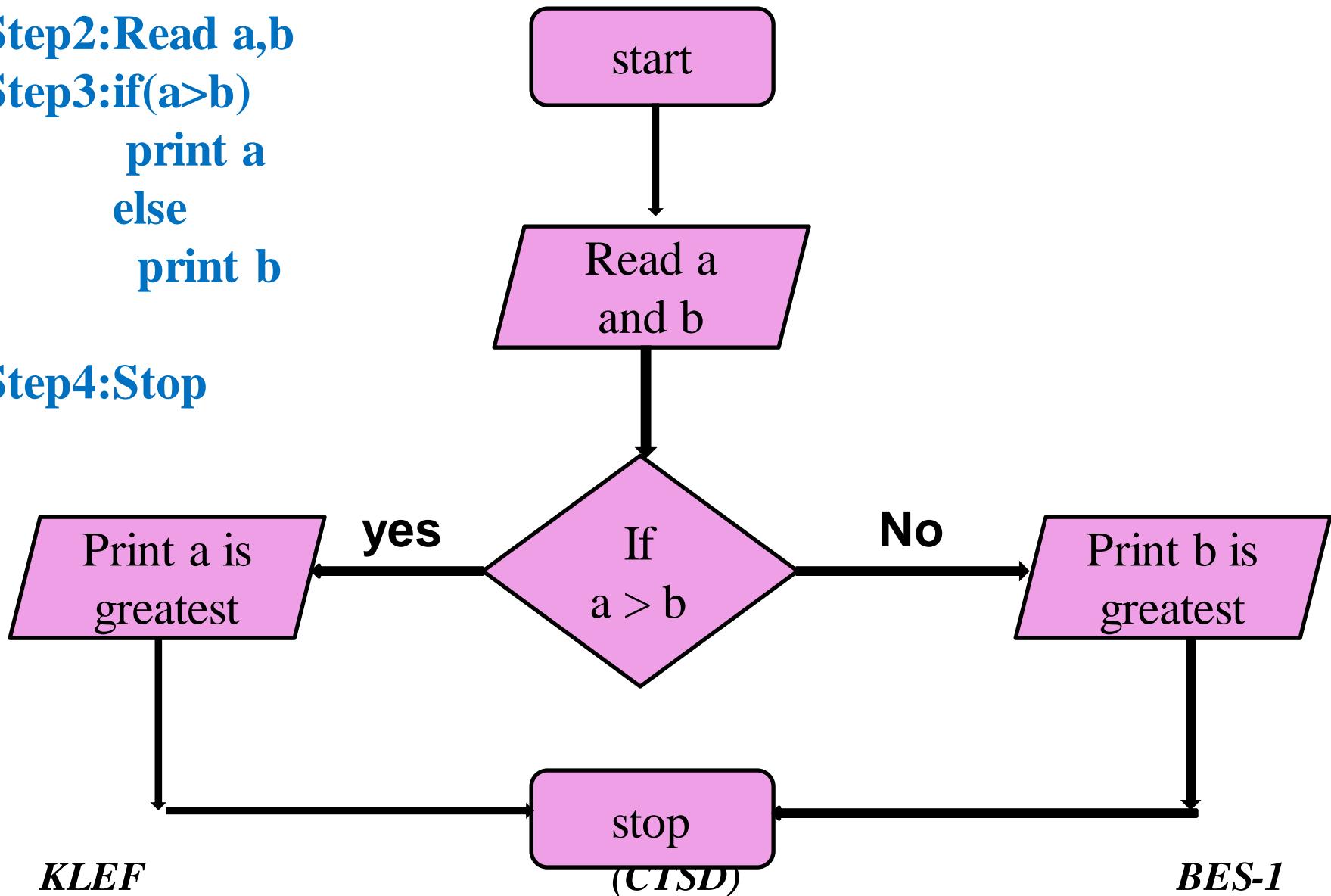
Step3:if( $a > b$ )

    print a

else

    print b

Step4:Stop





2. Design an algorithm to check whether the given number is positive or negative or zero

**Input:** num

**Step1:Start**

**Step2:Read num**

**Step3: if(num==0)**

**print “Zero”**

**else if(num<0)**

**print “Negative”**

**else**

**printf “Positive”**

**Step4:Stop**

**Process and Output**

**num= =0      Zero**

**num<0      Negative**

**num>0      Positive**

# Design a flow chart to check given number is zero, positive, or negative.

Step1: start

Step2: read x

Step3: if  $x == 0$

    print "x is zero"

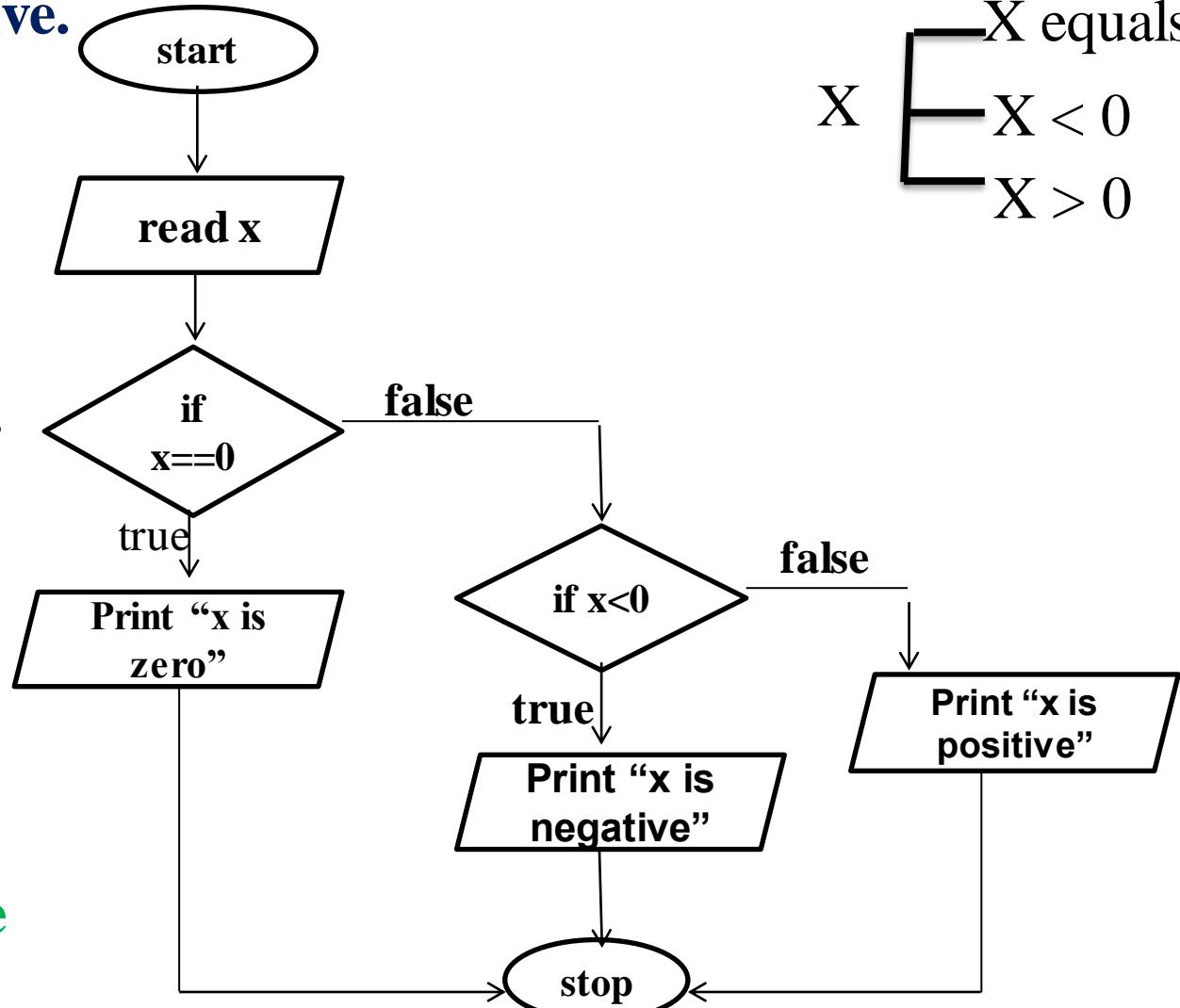
else if( $x < 0$ )

    print "x is negative"

else

    print "x is positive"

Step4: stop



4

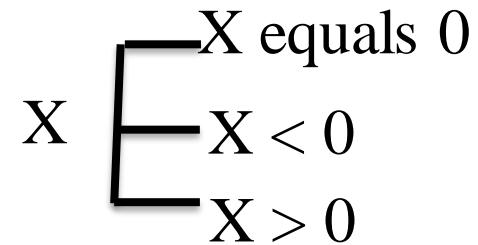
**X is positive**

0

**X is zero**

-17

**X is negative**





# Design an algorithm to illustrate the following. A company calculates discounts allowed to customers on the following basis

**Step1:** start

**Step2:** read q

**Step3:** if ( $q \geq 1$  and  $q \leq 99$ )

    print “discount is 5%”

    else if ( $q \geq 100$  and  $q \leq 199$ )

        print “discount is 7%”

    else if ( $q \geq 200$  and  $q \leq 499$ )

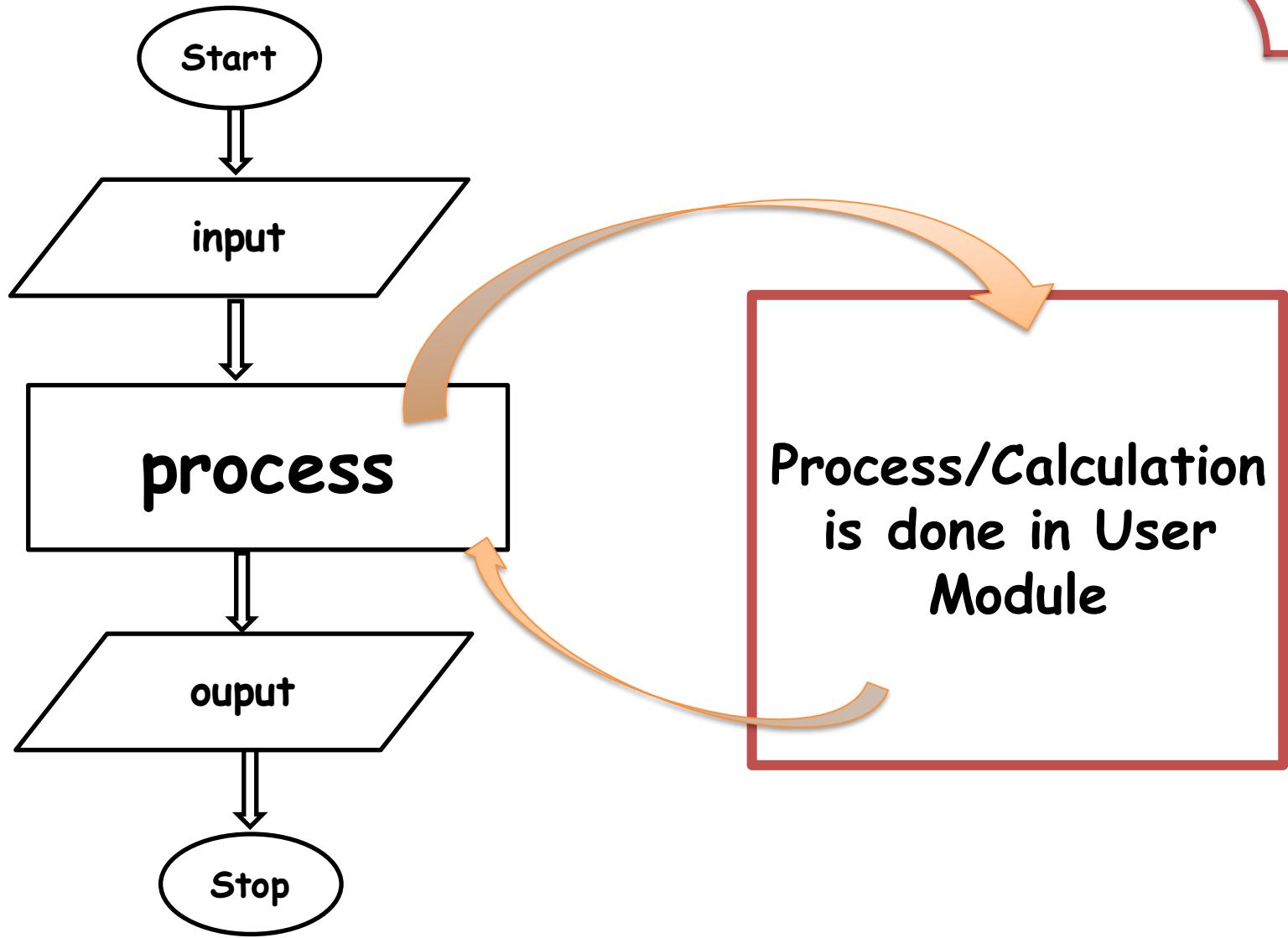
        print “discount is 9%”

    else

        print “discount is 10%”

**Step4 :** stop

Order quantity	Normal discount
1-99	5%
100-199	7%
200-499	9%
500 and above	10%



## Main Module



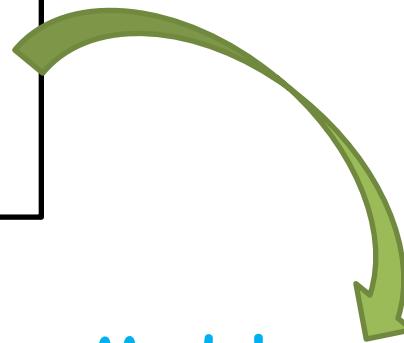
Step 1:Start

Step 2:Read m,a

Step 3:ans=calculate(m,a)

Step 4:Print ans

Step 5: Stop



User Module named as calculate  
and values of m and a forwarded  
to this block

Step 1:Start

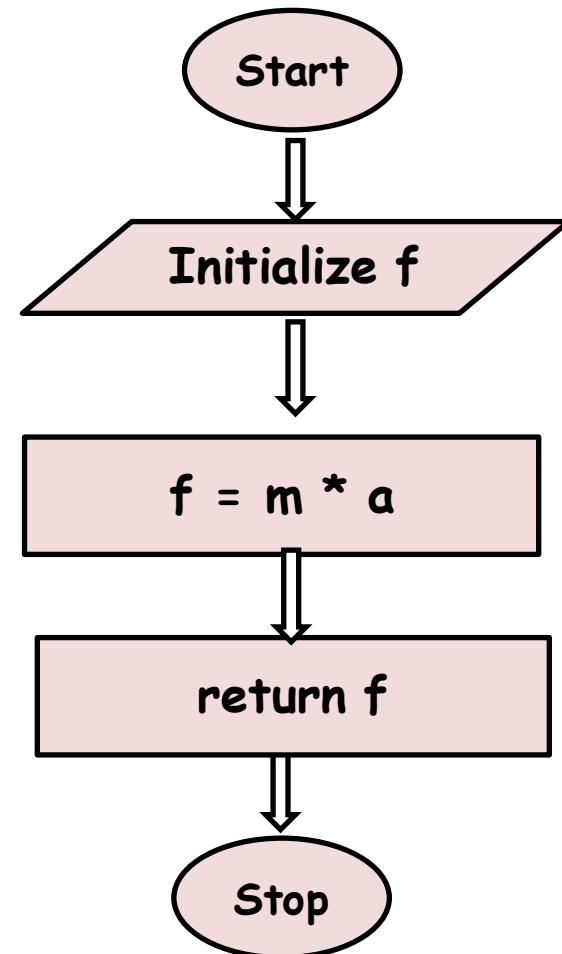
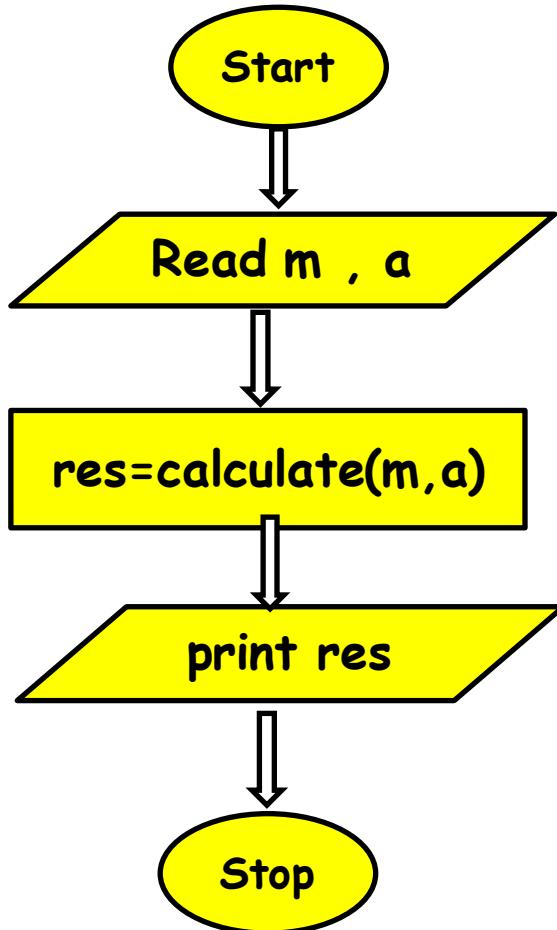
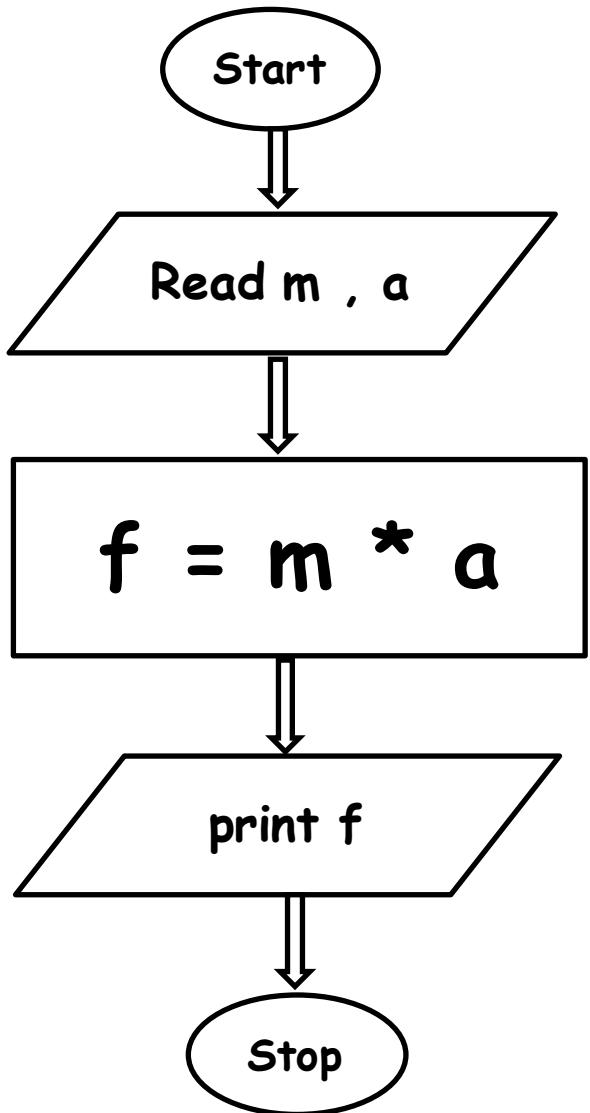
Step 2:Calculate  $f=m*a$

Step 3: return f

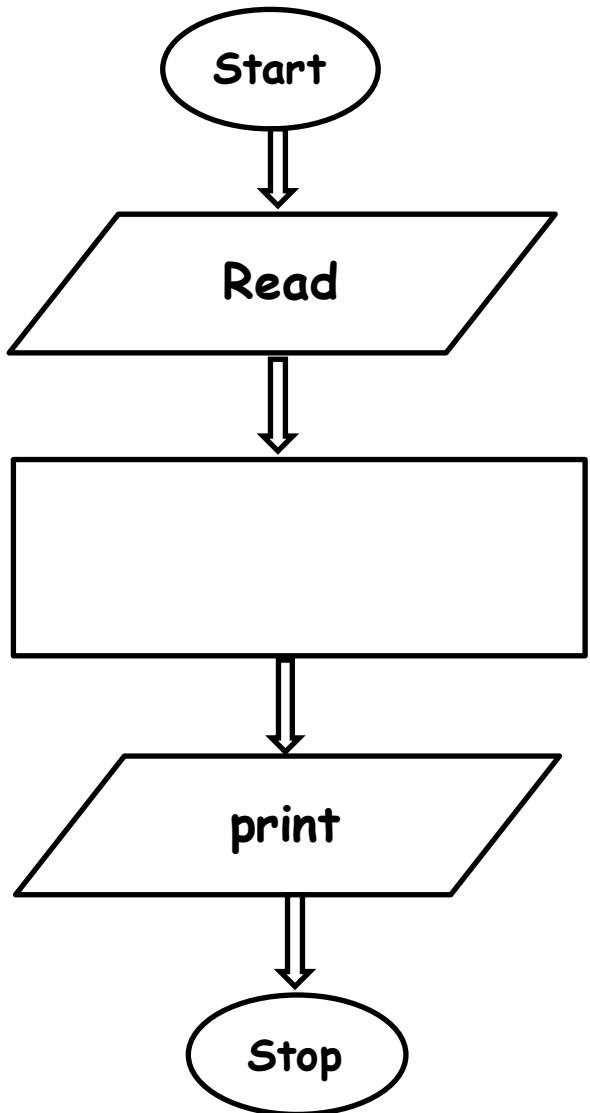
Step 4: Stop

## Modular Programming

# Modular Programming

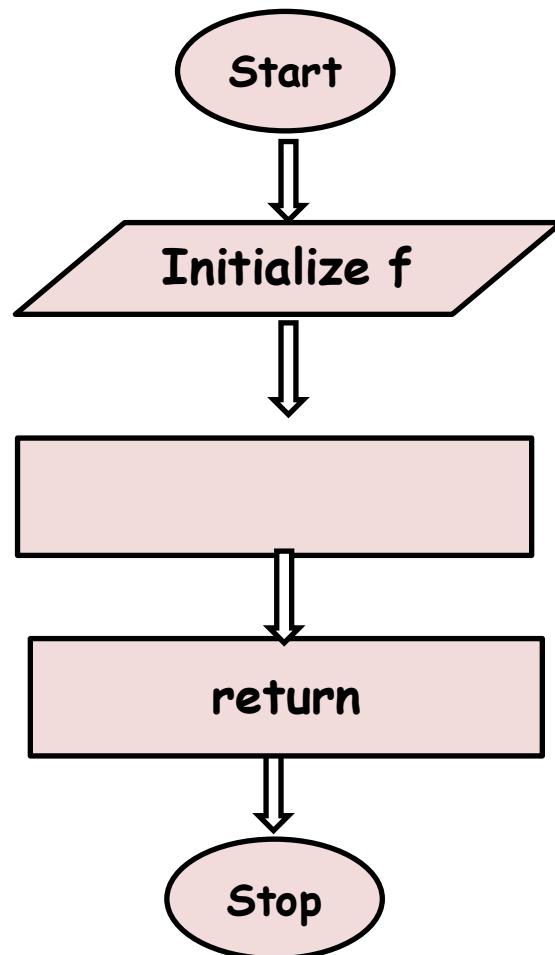
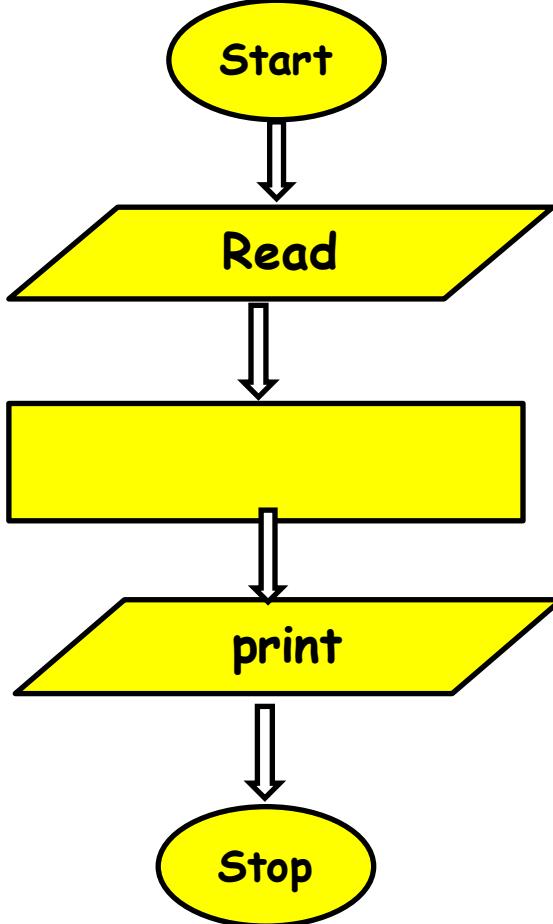


# Modular Programming



**KLEF**

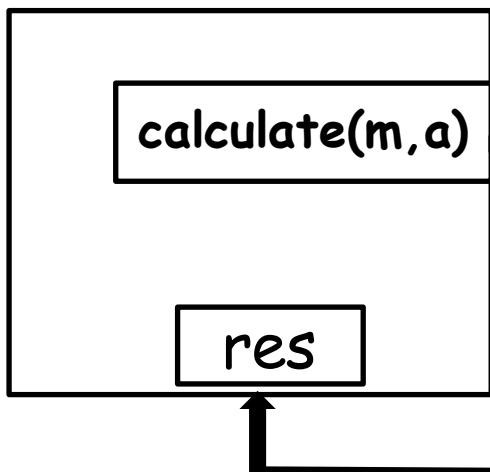
**(CTSD)**



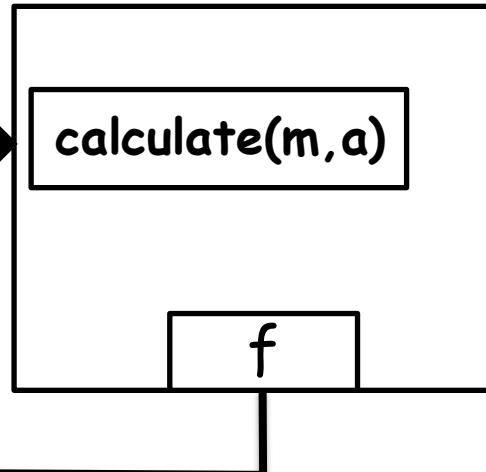
**BES-1**



Module Main



Module calculate





```
int calculate(int m, int a)
{
    int f;
    f=m*a;
    return f;
}
```

```
#include<stdio.h>
int calculate(int,int);
int main()
{
    int m,a;
    m=5,a=7;
    int res=calculate(m,a);
    printf('%d',res);
    return 0;
}
```



## Function Declaration:-

**returntype functionname(arglist);**

## Function Call:

**functionname(arglist);**

## Function Definition

**returntype functionname(arglist)**  
**{**  
**Body of Function;**  
**}**

**1. Problem Statement:** Ram and Hanuman measure and find that their houses are 50 miles apart. If they agree to meet at the midpoint between their two houses, how far will each of them travel in kilometres? (HINT: 1 mile = 1.6 Kilometres).



## Problem Comprehension:

- Each person must travel half of the distance to meet at midpoint.
- Here it is 25 miles
- 1 mile → 1.6 kms  
25 miles → ?

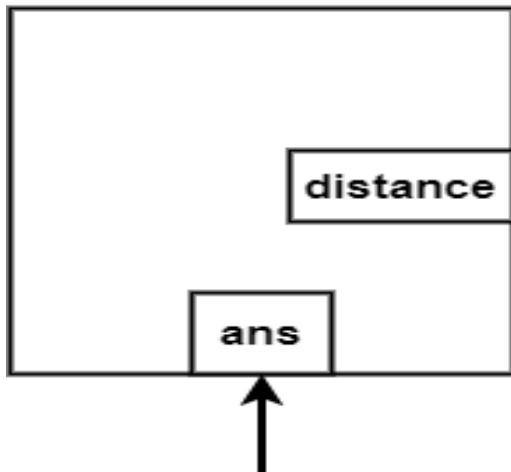
By cross multiplying we get  $? * 1\text{mile} = 25 * 1.6 \Rightarrow ? = 40\text{kms}$

## Sample Input & Output

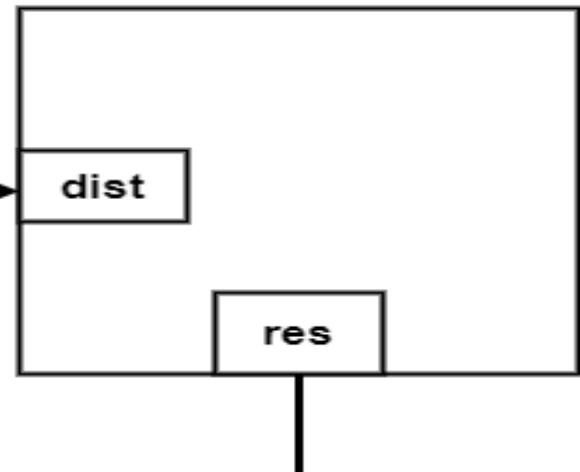
S. no	Input	Output
1	50	The distance to be travelled to meet at midpoint is 40 kms
2	60	The distance to be travelled to meet at midpoint is 48 kms

## Modular Design of the solution

Module main



Module midDistance





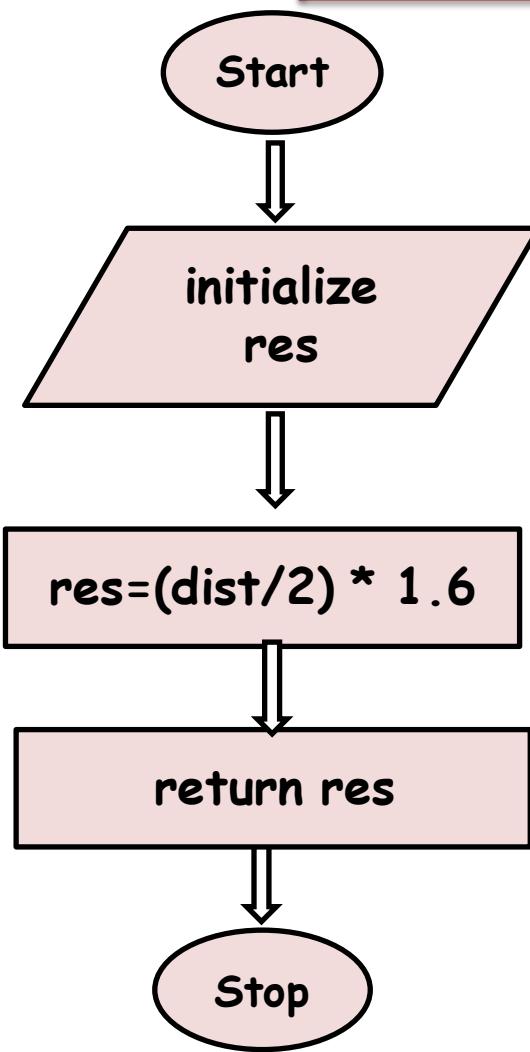
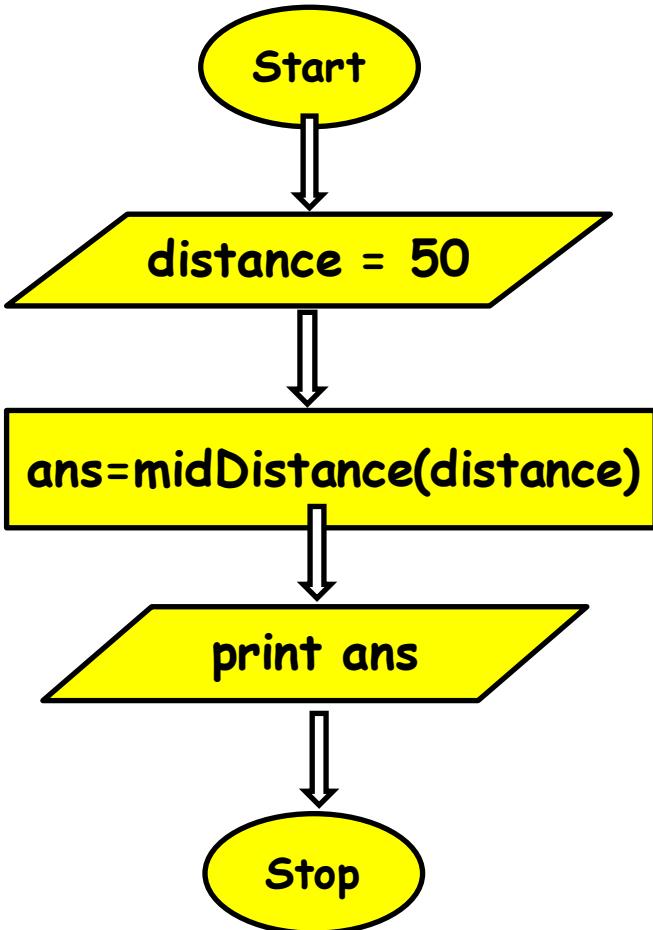
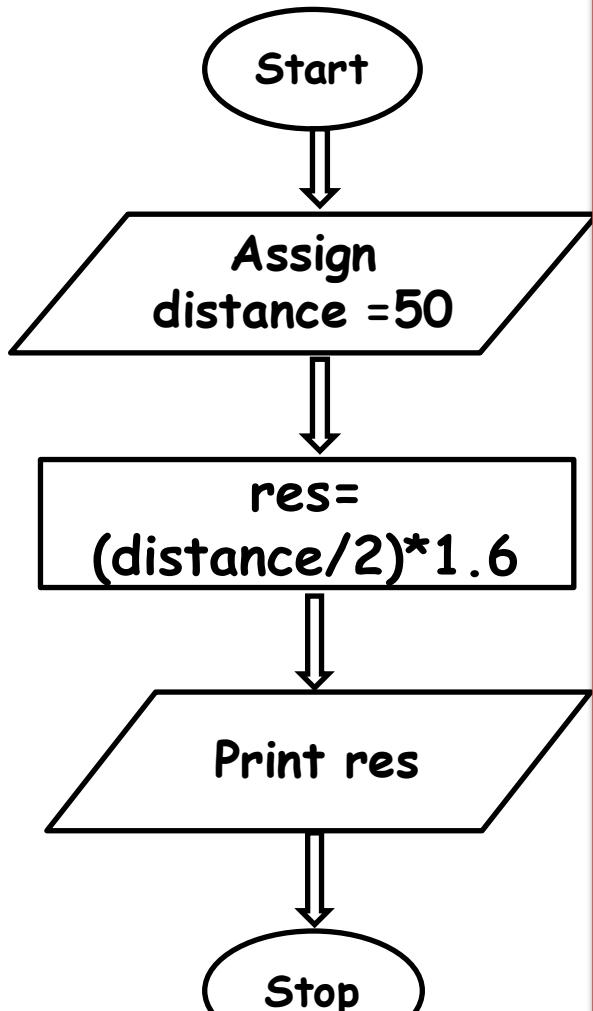
# Algorithm

## Algorithm for method main

Step 1: start  
Step 2: Declare variables distance, ans  
Step 3: distance = 50  
Step 4: ans=midDistance(distance)  
Step 5: print "The distance to be travelled to meet at midpoint is",ans  
Step 6: stop

## Algorithms for method midDistance

Step 1: start  
Step 2: declare variables res  
Step 3:  
Calculate  $res=(dist/2) * 1.6$   
Step 4: return res





```
#include<stdio.h>
float midDistance(float distance);
int main()
{
    int distance=50;
    float ans;
    ans=midDistance(distance);
    printf("%f",ans);
    return 0;
}
```

```
float midDistance(float dist)
{
    float res;
    res=(dist/2) * 1.6;
    return res;
}
```



## 2. Problem Statement:

Krishna's livelihood depends on selling milk. Every day he brings  $x$  gallons of milk from his village to Vijayawada. In Vijayawada, he pours 1 liter of milk to each of 36 households. Find the number of liters of milk left with him? (HINT: 1 gallon = 3.785 liters). Assume that Krishna has adequate amount of milk.



## Problem Comprehension:

- Let  $x$  be 12 gallons
- 1 gallon  $\rightarrow$  3.785 litres

12 gallons  $\rightarrow$  ?

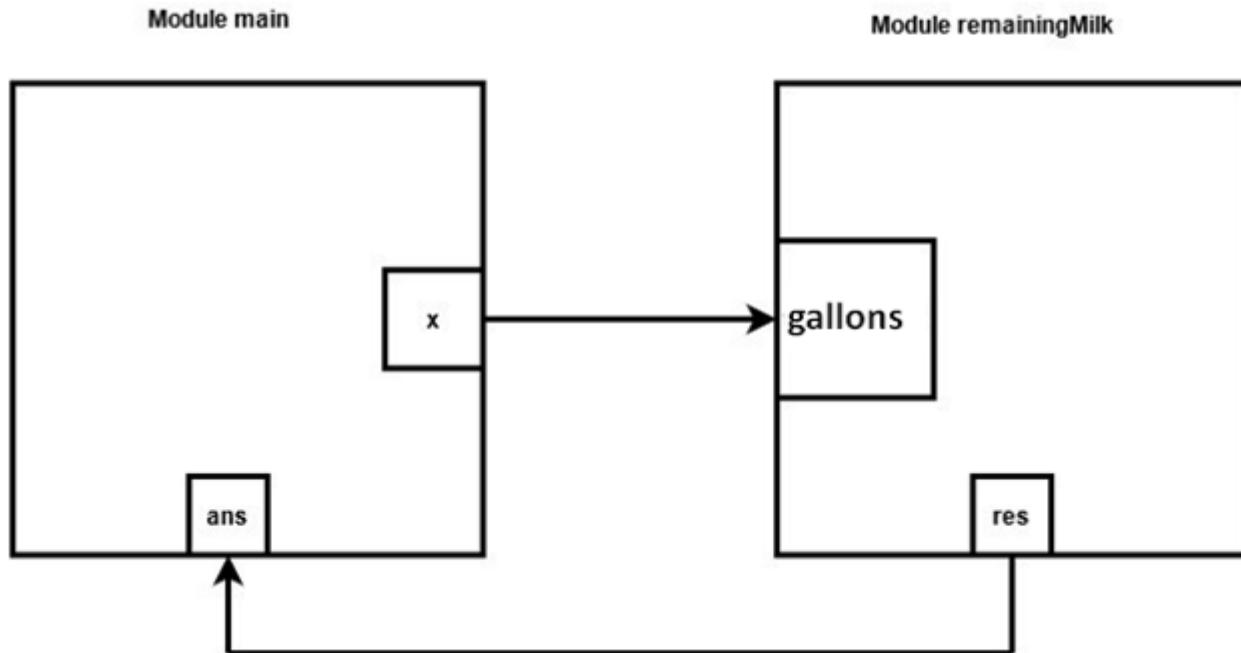
By cross multiplying we get  $? * 1 \text{ gallon} = 12 * 3.785 \Rightarrow ? = 45.42 \text{ litres}$

- if he pours 36 litres, then  $45.42 - 36 = 9.42 \text{ litres}$  will be left with him

## Sample Input & Output

S. no	Input	Output
1	12	The number of litres left = 9.42 litres
2	10	The number of litres left = 1.85 litres

## Modular Design of the solution





# Algorithm

## Algorithm for method main

Step 1: start

Step 2:

Step 3:

Step 4: ans=remainingMilk(x)

Step 5:

Step 6: stop

## Algorithms for method remainingMilk

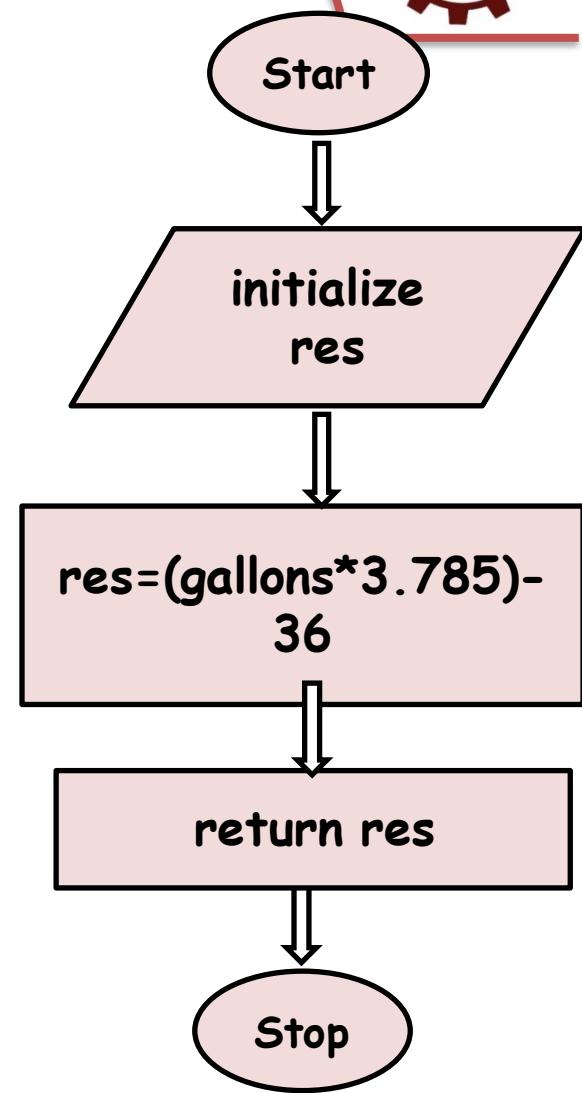
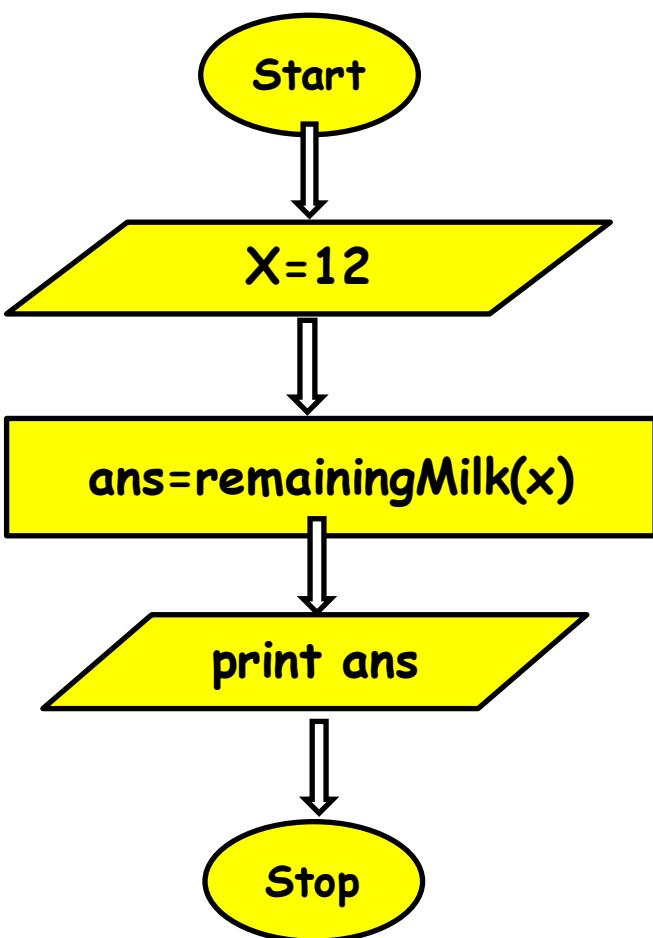
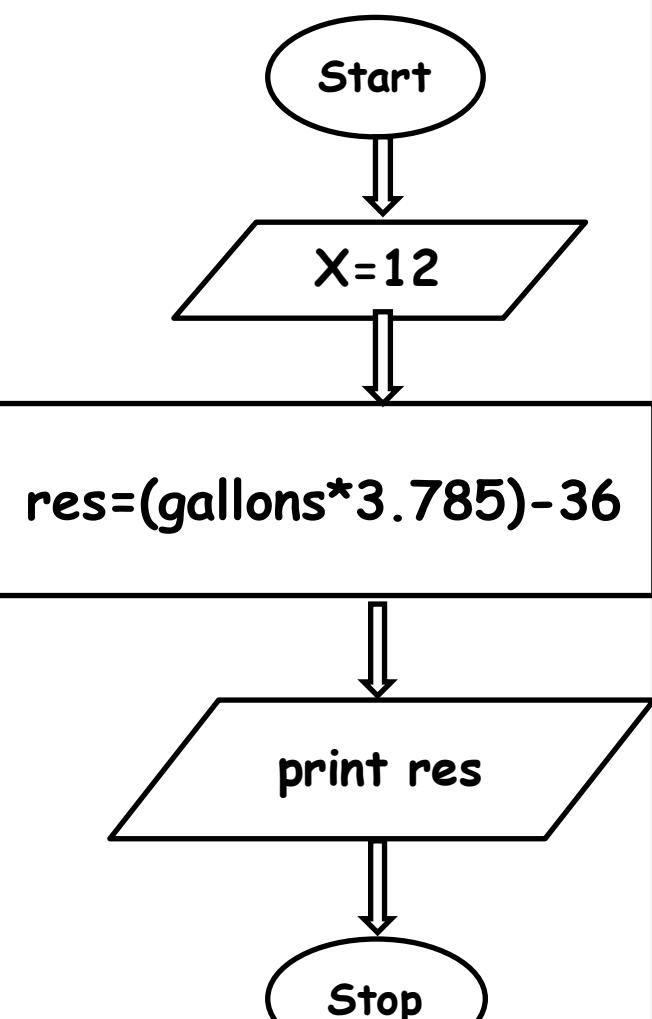
Step 1: start

Step 2: declare variable res

Step 3:

res =(gallons\*3.785)-36

Step 4: return res





```
#include<stdio.h>
float remainingMilk(int);
int main()
{
int x=12;
float ans;
ans = remainingMilk(x);
printf("%f", ans);
return 0;
}
```

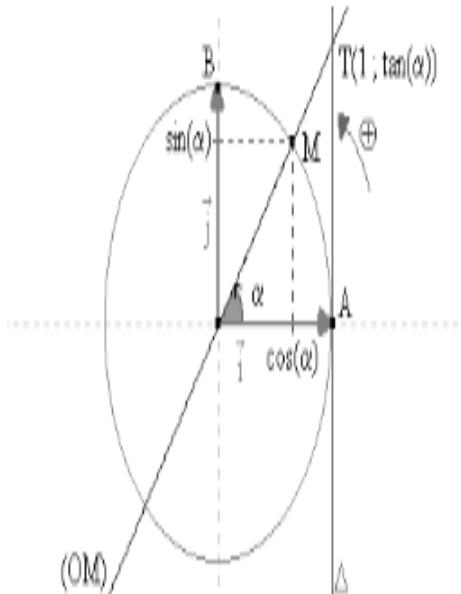
```
float remainingMilk(int gallons)
{
float res;
res =(gallons*3.785)-36;
return res;
}
```

### Problem 3:

Sudha is learning about the C library math.h functions. She found that there is `tan()` function in math.h header file. She wants to calculate  $\tan 45^\circ$ , but she finds that tan function accepts angle in only radians. Help Sudha to convert  $45^\circ$  to radians and calculate the value of  $\tan 45^\circ$ . (Hint: 1 degree =  $\pi/180$  radians)

### Problem Comprehension:

- Convert the degree to radians using  $1 \text{ degree} = \pi/180 \text{ radians}$
- $\tan(0.7854 \text{ radians}) = 1$





```
#include<stdio.h>
#include<math.h>
float calTan(float);
int main()
{
    float degree,ans,dist;
    degree=45;
    dist=calTan(degree);
    ans=tan(dist);
    printf("%f",ans);
    return 0;
}
```

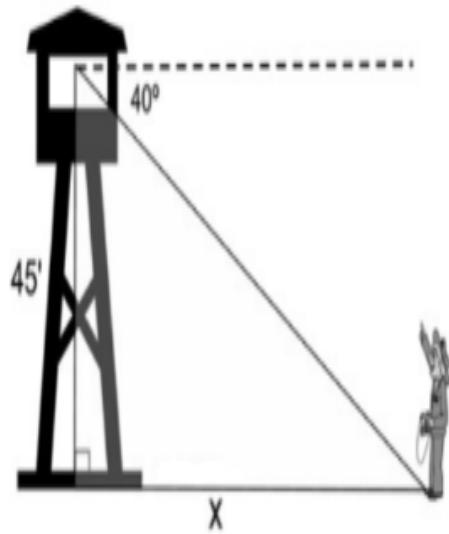
```
float calTan(float d)
{
    float res;
    res=d * (3.14/180);
    return res;
}
```

## Problem 4:

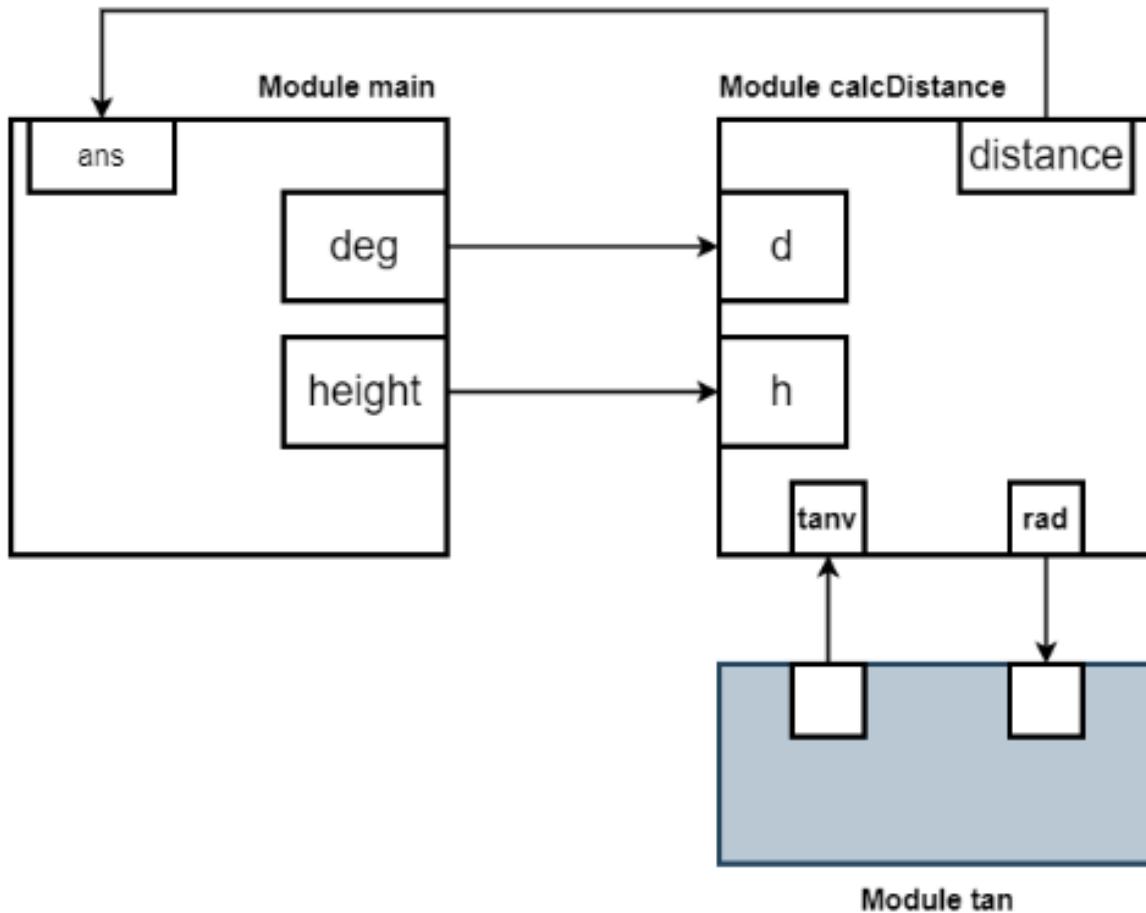
From the top of a tower, an Indian army soldier sees his enemy on the ground at an angle of depression of  $40^\circ$ . If the tower is 45 feet in height, how far is the enemy from the base of the tower.

### Problem Comprehension:

- We will use alternate interior angles from the parallel horizontal lines, so we place  $\theta^\circ$  inside the triangle by the enemy (bottom right)
- Convert angle to degrees in order to use inbuilt tan function available in math.h
- $\tan(\theta) = (\text{height}/\text{distance})$



## Modular Design of the solution





```
#include<stdio.h>
#include<math.h>
float calDistance(float, float);
int main()
{
    float deg, height, ans;
    deg=40;
    height=45;
    ans=calDistance(deg, height);
    printf("Distance=%f",ans);
    return 0;
}
```

```
float calDistance(float d, float h)
{
    float res, rad;
    rad= d * 3.14 /180;
    res=h/tan(rad);
    return res;
}
```

# KEYWORDS



**int  
float  
char**

**if  
else**

**for  
while  
do**

**struct  
union  
typedef  
enum**

**short  
long  
double**

**switch  
case  
break  
continue  
default  
goto**

**void  
return**

**signed  
unsigned**

**auto  
extern  
register  
static**

**sizeof  
const  
volatile**



# Input-Output Functions

# printf output function 2 usages

1.) `printf("message");`      syntax-Rule

**Ex**      **printf("HelloSection7Students");**

```
printf("gdfjsfjhskjdfhskjdhfk");
```

**any message can be printed here**

2.) **printf(“typespecifier”, variablename);**

Ex: printf("%d %d",a,b); 10 20

```
printf("a=%d b=%d",a,b);           a=10   b=20
```



scanf

## Format Specifier

Format specifier is a special character which is used to specify the data type of the value being read.

Some of the frequently used specifiers are as follows:

- s - strings
- d - decimal integers
- f - floating-point numbers
- c - a single character



scanf

inputfunction

a=10; //assigning value to a

scanf("typespecifier",&variablename);

scanf("%d",&a); //reading from user

## Operators

1) Arithmetic      +      -      \*      /(Q)

    %(R)      ++      --

2) Assignment      +=      -=      \*=      /=      %=

3) Relational      <=      >=      <      >

4) Equality      ==      !=      a=b      a==b

5) Logical      &&      ||      !



# Bit Wise Operators -AND

```
#include <stdio.h>
int main()
{
    int a=6, b=14;
    printf(" BitwiseAND for a&b is %d",a&b);
    return 0;
}
```

The binary value of 'a' and 'b' are 0110 and 1110

a AND b = 0110 && 1110 = 0110



# Bit Wise Operators -OR

```
#include <stdio.h>
int main()
{
    int a=6,b=14;
    printf("Bitwise OR  a|b is %d",a|b);
    return 0;
}
```

The binary representation of a & b be:

a = 0110

b = 1110

1110



# Bit Wise Operators –ExclusiveOR

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

```
int main()
```

```
{
```

```
    int a=6,b=14;
```

```
    printf("Bitwise exclusive OR a^b is  
%d",a^b);
```

```
    return 0;
```

The binary representation of a & b be:

```
}
```

a = 0110

b = 1110

1000



# Bit Wise Operators –Complement

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

```
int main()
```

```
{
```

```
    int a=8;
```

```
    printf("Bitwise Complementfor a is %d",~a);
```

```
    return 0;
```

```
}
```

$$\begin{array}{r} 0 \quad 1000 \quad 0111 \\ 1 \quad 0111 \quad 1000 \\ \hline & & 1 \\ & 1001 & \end{array} \quad -9$$



# Bit Wise Shift Operators

- **Left-shift operator**

**Operand  $<< n$**

- **Right-shift operator**

**Operand  $>> n$**



# Bit Wise Shift Operators

```
int a=2<<1;
```

Let's take the binary representation of 2 assuming int is 1 byte for simplicity.

Position	7	6	5	4	3	2	1	0
Bits	0	0	0	0	0	0	1	0

Now shifting the bits towards left for 1 time, will give the following result

Position	7	6	5	4	3	2	1	0
Bits	0	0	0	0	0	1	0	0

If you left shift like  $2 \ll 2$ , then it will give the result as 8. Therefore left shifting 1 time, is equal to multiplying the value by 2.



# Bit Wise Shift Operators

```
#include <stdio.h>
int main()
{
    int a=6;
    printf("The value of a<<2 is : %d ",  
a<<2);
    return 0;
}
```

$$6 \times 2 = 12 \times 2 = 24$$



# Bit Wise Shift Operators

int a=8>>1;

Let's take the binary representation of 8 assuming int is 1 byte for simplicity.

Position	7	6	5	4	3	2	1	0
Bits	0	0	0	0	0	1	0	0

Now shifting the bits towards right for 1 time, will give the following result

Position	7	6	5	4	3	2	1	0
Bits	0	0	0	0	0	0	1	0

Now the result in decimal is 4. Right shifting 1 time, is equivalent to dividing the value by 2.



# Bit Wise Shift Operators

```
#include <stdio.h>
int main()
{
    int a=8;
    printf("The value of a<<2 is : %d ",  
a>>2);
    return 0;
}
```

$$6 \times 2 = 12 \times 2 = 24$$



## Rules to create an identifier in ‘C’ program:- (Variables)

- ❖ An identifier should contains only **letters, digits and underscore symbol (\_).**
- ❖ A **keyword** is not used as an identifier.
- ❖ The first character in an identifier should be a **letter or underscore (\_)** but not digit.

❖ Ex:

```
int x23_fsa;  
int _sdfa;  
float _999fsal;
```



# Examples

```
int rollno=47;  
float height=7.1;  
float percentage=8.9;  
int apple=4;  
int basket;  
int age@16:  
float percen*tage;  
int height#ram;  
float $ram;int _$89y;  
float marks9maths;
```



## **Constants:-**

Constants are the terms that can't be changed during the execution of a program. For example 5, 6, 3.454, 'a', 'L', "hello" ext .

- a. Integer constants**
- b. Floating-point constants**
- c. Character constants**
- d. String constants**



## Integer constants:-

numeric constants without any fractional part or exponential part.

Decimal system

Octal system

Hexadecimal system.

- Octal number preceded by 0(zero).
- Hexadecimal preceded by 0x or 0X Examples;

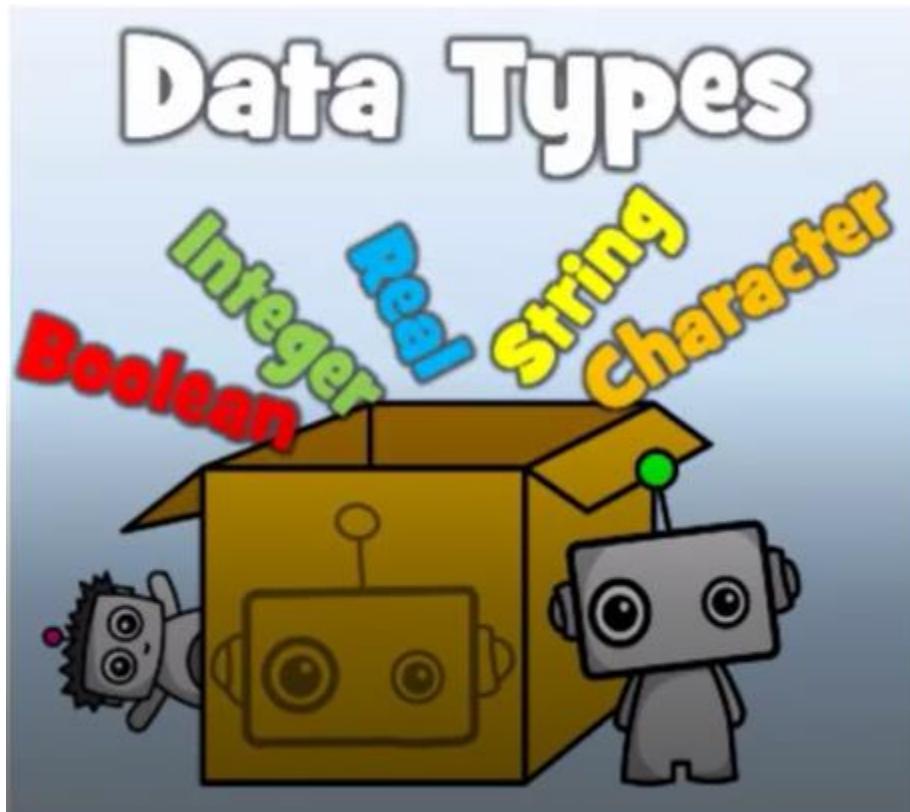
Decimal constants -- 99, -88, 123, etc.

Octal constants -- 04334, 0453, etc..

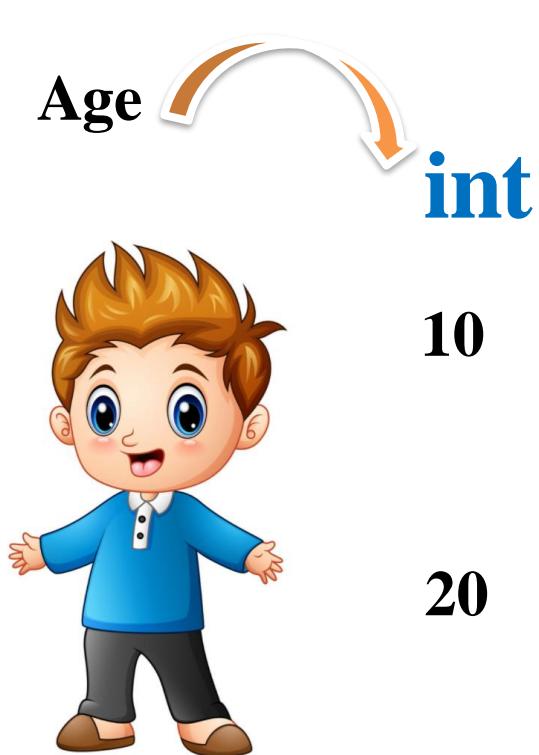
Hexadecimal constants -- 0x23 , 0X23, 0x34ab4.

# Introduction to Data Types

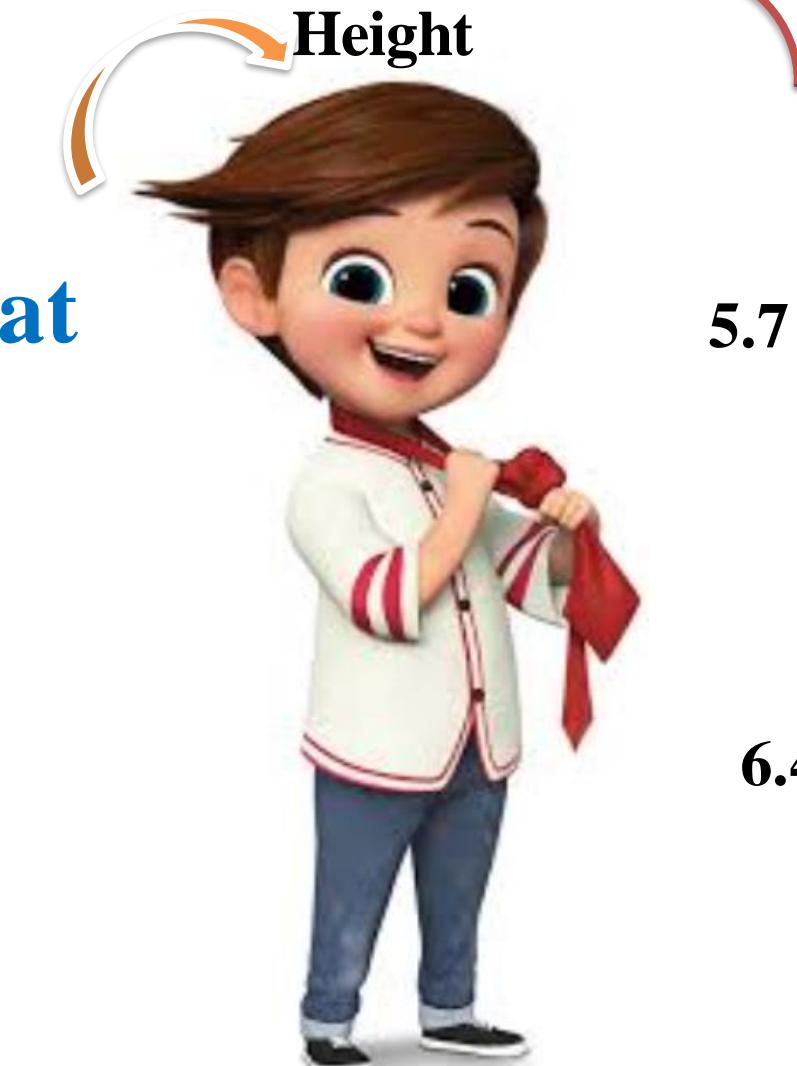
Data Types: int, int array, int pointer, float, float array, float pointer



To access data ,First we need to define its datatype



float



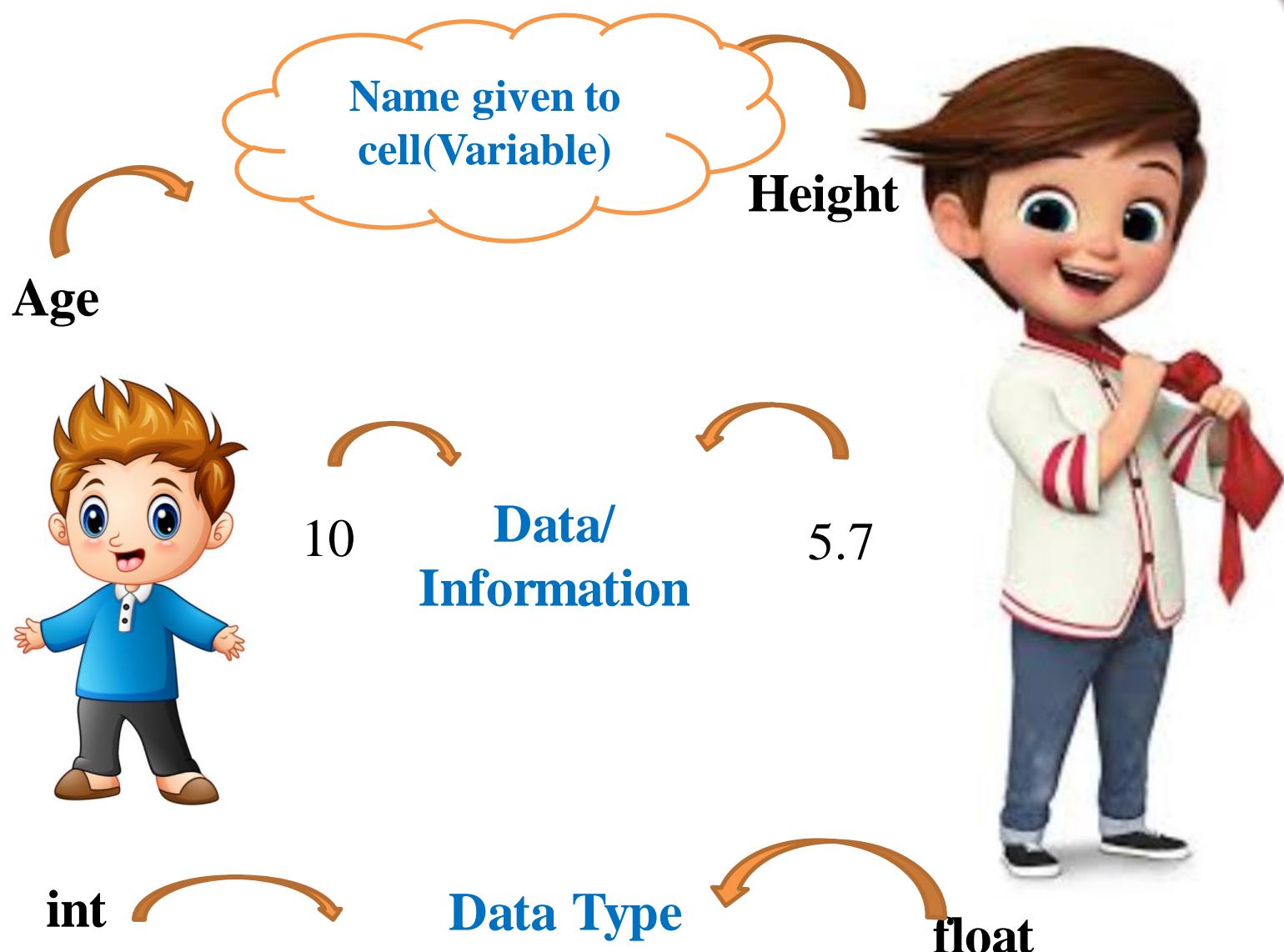
Non Fractional Value

KLEF

(CTSD)

Fractional Value

BES-1



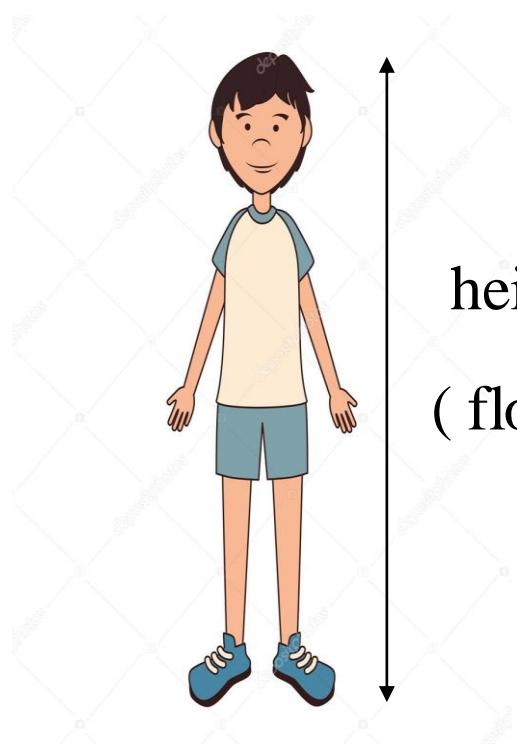
To access any data in our program first we need to define its data type.

He is 20  
Years old.

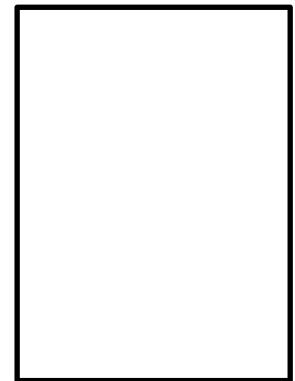
His height is  
5.7 Feet

age      20  
( int )

height      5.7  
( float )



Datatypes





# Data Types

## What is Data?

Data is a known fact which needs to be stored in memory to use in programming.

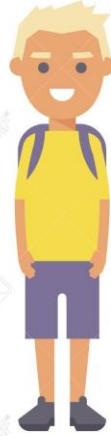
## Data Type

Data types are used to define a variable to hold those facts as data value.

## Variable

Name given to memory block which holds value. The value can be changed at any moment so called variable.

age



int

height



float

1. 20
2. 25

5.4

5.7

Data

Datatypes

Variables

age



int

height



float

1. 20
2. 25

5.4

5.7

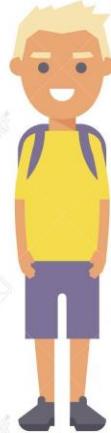
Name given to the cell  
(Variable)  
age

20002

Address of the cell  
(Memory Location)

How this is done in Memory????    int age = 20;

age



int

height



float

1. 20
2. 25

5.4

5.7

Try to store height!!!

Name given to the cell  
(Variable)

age

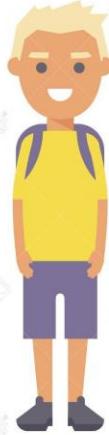


20002

Address of the cell  
(Memory Location)

How this is done in Memory???? `int age = 20;`

age



int

height



float

1. 20
2. 25

5.4

5.7

Name given to the cell  
(Variable)  
**height**




**20010**

Address of the cell  
(Memory Location)

So, need another Datatype

How this is done in Memory????

**float height = 5.4;**



## Integer Data

An integer is a Non-fractional value stored in 2 Bytes or 4 Bytes within range - $2^{15}$  to  $+2^{15}-1$  or - $2^{31}$  to  $+2^{31}-1$ .

2	0	-5	+56	-89	✓
\$25	2,366	6.	12.589	X	

```
int variable_name;
```

```
int age;  
age = 20;
```

## Float Data

Floating point data is a numerical data with fractional or real number value stored in 4 Bytes within range 3.4E-38 to 3.2E+38.

2.00	3.2	1.3E+5	2.5e-5	✓
2,300.00	\$25,857.50	X		

```
float variable_name;
```

```
float height;  
height = 5.7;
```



# DataTypes

- What type of data is allowed to store**
  
- How much memory is needed to store the data.**

# Data Types in C

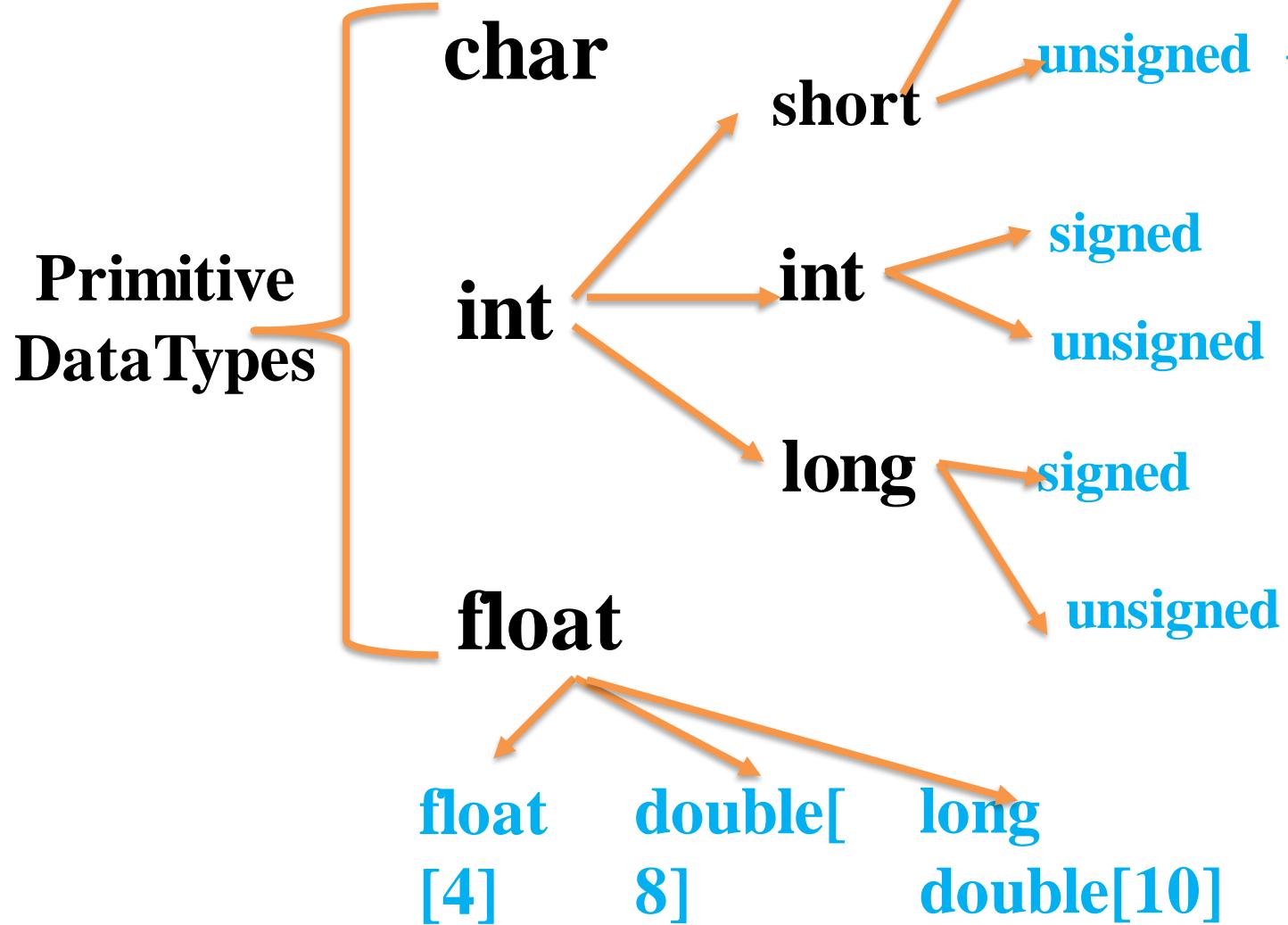
## Primitive

- int
- char
- float
- double

→ %d  
→ %f

## Derived

- Array
- Structure
- Union
- Enum
- Pointer





**10**

**short**

**2byte**

**756**

**int**

**2 or 4bytes**

**1234567890**

**long**

**4bytes**



```
#include <stdio.h>
int main()
{
    char ch = 'A';
    char str[20] = "Section7";
    float flt = 10.234;
    int no = 150;
    double dbl = 20.123456;
    printf("Character is %c \n", ch);
    printf("String is %s \n" , str);
    printf("Float value is %f \n", flt);
    printf("Integer value is %d \n" , no);
    printf("Double value is %lf \n", dbl);
    printf("Octal value is %o \n", no);
    printf("Hexadecimal value is %x \n", no);
    return 0;
```



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

```
int main()
```

```
{
```

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

```
    a=10;
```

```
    b=15;
```

```
    c=47;
```

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

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

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

```
}
```

a

10

b

15

c

47

1234

5678

910

int

a

10

b

15

c

47



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

```
int main()
```

```
{
```

```
    int no1, no2, sum;
```

```
    no1=10;
```

```
    no2=15;
```

```
    // calculating sum
```

```
    sum = no1 + no2;
```

```
    printf("%d + %d = %d", no1, no2, sum);
```

```
    return 0;
```

```
}
```

**no1**

10

**no2**

15

**sum**

25

1234

5678

910



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

```
int main()
```

```
{
```

```
    int m, c, e;
```

```
    // calculating formula e=m*c*c
```

```
    e = m * c * c;
```

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

```
    return 0;
```

```
}
```

**m**

7

**c**

2

**e**

28

1234

5678

910



```
#include <stdio.h>
int main()
{
    float number1 = 13.5;
    double number2 = 12.4;

    printf("number1 = %f\n", number1);
    printf("number2 = %lf", number2);
    return 0;
}
```

number1 = 13.500000  
number2 = 12.400000



#include <stdio.h>

int main()

{

char chr = 'a';

character = a

printf("character = %c", chr);

return 0;

#include <stdio.h>

}

int main()

{

char name[20];

scanf("%s", &name);

printf("Your name is: %s", name);

return 0;

}

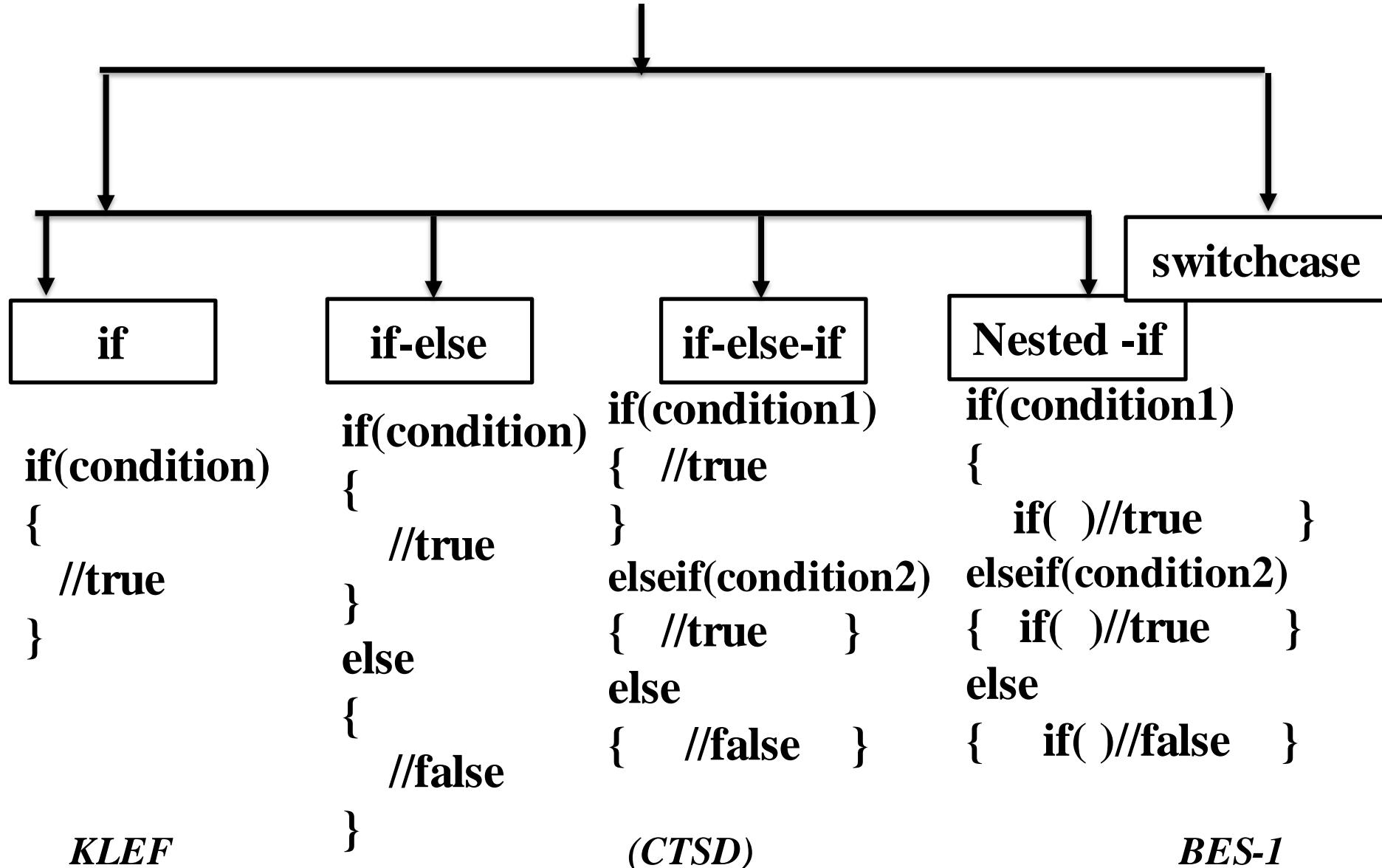


# Control Flow Statements

Decision Making using  
Conditional Statements



# Conditional Statements





```
#include<stdio.h>
int main()
{
    int no;
    no=5;

    if(no%2==0)
        printf("given no is even");
    else
        printf("given no is odd");
    return 0;
}
```

no  
5



```
#include<stdio.h>
int main()
{
    int no;
    no=5;

    if(no==0)
        printf("given no is zero");
    else if(no>0)
        printf("given no is positive");
    else
        printf("given no is negative");
    return 0;
}
```

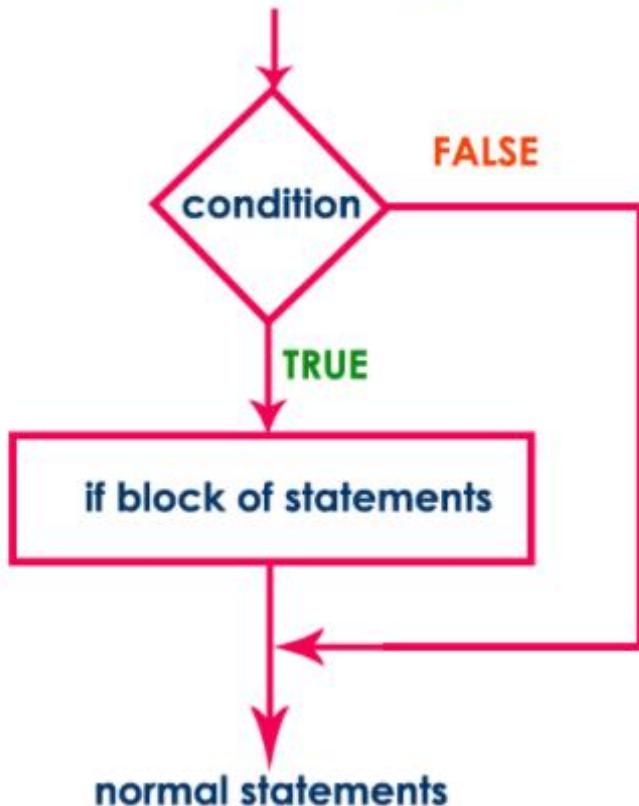
no  
5

# Simple if

## Syntax

```
if ( condition )
{
    ....
    block of statements;
    ....
}
```

## Execution flow diagram

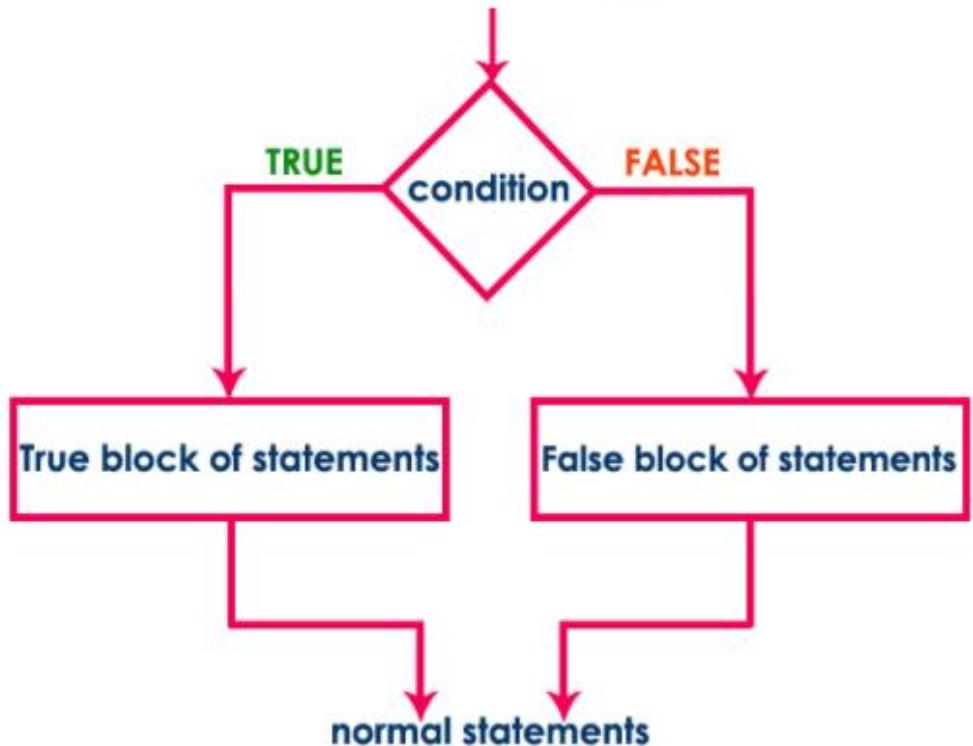


# if else

## Syntax

```
if ( condition )
{
    ....
    True block of statements;
    ....
}
else
{
    ....
    False block of statements;
    ....
}
```

## Execution flow diagram





```
#include<stdio.h>
int main()
{
    int age;
    if(age>=18)
        printf("Eligible for Vaccination");
    return 0;
}
```

## Simple if

```
#include<stdio.h>
int main()
{
    int age;
    if(age>=18)
        printf("Eligible for Vaccination");
    else
        printf("Not Eligible for Vaccination");
    return 0; }
```

## if else



## Syntax

# Nested if

```
if ( condition1 )
{
    if ( condition2 )
    {
        ....
        True block of statements 1;
    }
    ....
}
else
{
    False block of condition1;
}
```



```
#include<stdio.h>
int main()
{
    int age;
    if(age>=18)
    {
        if(age>=42)
        {
            printf("Eligible for Vaccination");
        }
    }
    else
        printf("Not Eligible for Vaccination");
    return 0;
}
```

## Nested if



## switch...case statement

- ❖ Gives ability to make decisions from fixed available choices.
- ❖ Rather making decision based on conditions.
- ❖ Using switch we can write a more clean and optimal code, that take decisions from available choices.

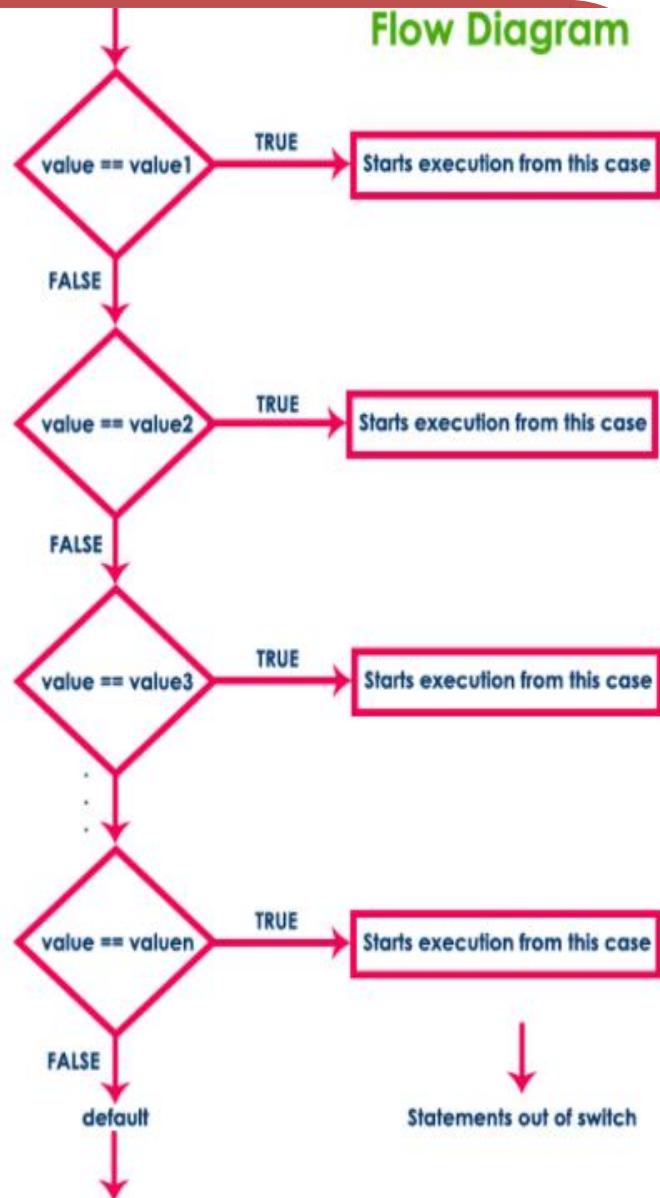


```
switch(expression)
{
    case 1: /* Statement/s */
        break;
    case 2: /* Statement/s */
        break;
    case n: /* Statement/s */
        break;
    default: /* Statement/s */
}
```

## Flow Diagram

### Syntax

```
switch ( expression or value )
{
    case value1: set of statements;
    ....
    case value2: set of statements;
    ....
    case value3: set of statements;
    ....
    case value4: set of statements;
    ....
    case value5: set of statements;
    ....
    .
    .
    .
    default: set of statements;
}
```





```
int num = 2;  
switch(num)  
{  
case 1: printf("I am One");  
        break;  
case 2: printf("I am Two");  
        break;  
case 3: printf("I am Three");  
        break;  
default: printf("definitely I am not 1, 2 and 3.");  
}
```

num  
2



- ❖ All conditions sequentially until condition matched.
  - ❖ It skips all subsequent condition check once condition got matched.
- 
- ❑ C compiler generate a lookup table based on the case values.
  - ❑ On execution, instead of matching switch(expression) for each case, it query the lookup table generated during compilation.
  - ❑ If the case exists in lookup table, then it transfers control to the matching case otherwise to default case (if mentioned).



```
#include <stdio.h>

int main() {
    char c;
    printf("Enter a character: ");
    scanf("%c", &c);
    if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z'))
        printf("%c is an alphabet.", c);
    else
        printf("%c is not an alphabet.", c);
    return 0;
}
```

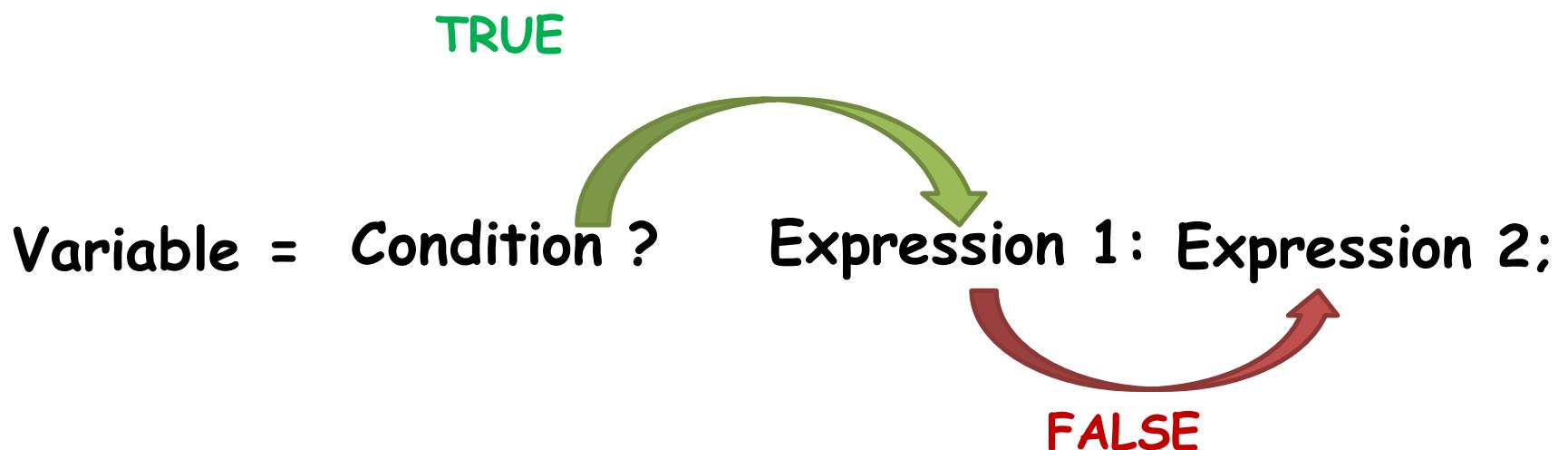
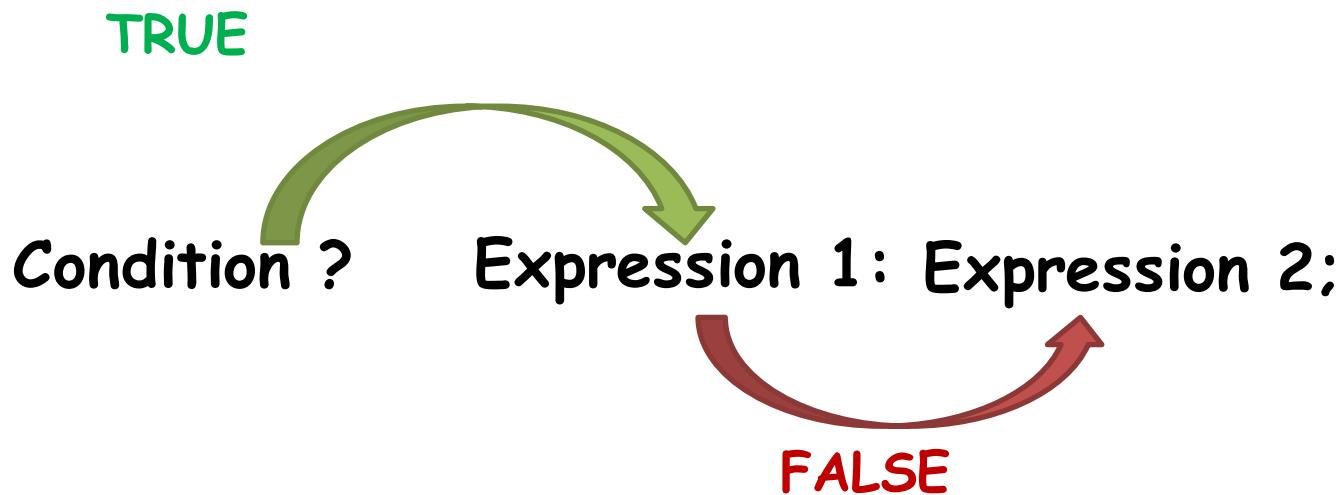


```
#include <stdio.h>

int main() {
    char c;
    int lowercase_vowel, uppercase_vowel;
    printf("Enter an alphabet: ");
    scanf("%c", &c);
    lowercase_vowel = (c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u');
    uppercase_vowel = (c == 'A' || c == 'E' || c == 'I' || c == 'O' || c == 'U');
    if (lowercase_vowel || uppercase_vowel)
        printf("%c is a vowel.", c);
    else
        printf("%c is a consonant.", c);
    return 0;
}
```



# Ternary Operator





# Ternary/ Conditional Operator

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

```
int main()
```

```
{
```

```
    int age;
```

```
    printf("Enter your age");
```

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

```
(age>=18)?(printf("eligible for voting")):(printf("not eligible for voting"));
```

```
    return 0;
```

```
}
```



# Predict the output

```
#include <stdio.h>
int main()
{
    int a=5,b;
    b=((a==5)?(3):(2));
    printf("The value of 'b' variable is : %d",b);
    return 0;
}
```



# Predict the output

```
#include <stdio.h>
int main()
{
    int a = 10, b = 20, c;
    c = (a < b) ? a : b;
    printf("%d", c);
    return 0;
}
```



# Predict the output

```
int a = 1, b = 2, ans;  
if (a == 1)  
{  
    if (b == 2)  
        ans = 3;  
    else  
        ans = 5;  
}  
else  
    ans = 0;  
printf ("%d\n", ans);
```

```
int a = 1, b = 2, ans;  
ans = (a == 1 ? (b == 2 ? 3 : 5) : 0);  
printf ("%d\n", ans);
```

# ARRAYS



*KLEF*

*(CTSD)*

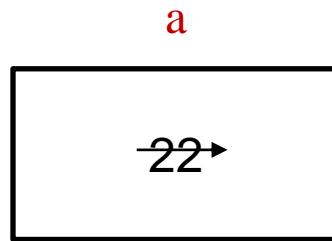
*BES-1*





# Integer Array

```
int main()
{
    int a;
    a=22;
    -----
    -----
    a=34;
    -----
    a=56;
    -----
    return 0;
}
```



→34→

56

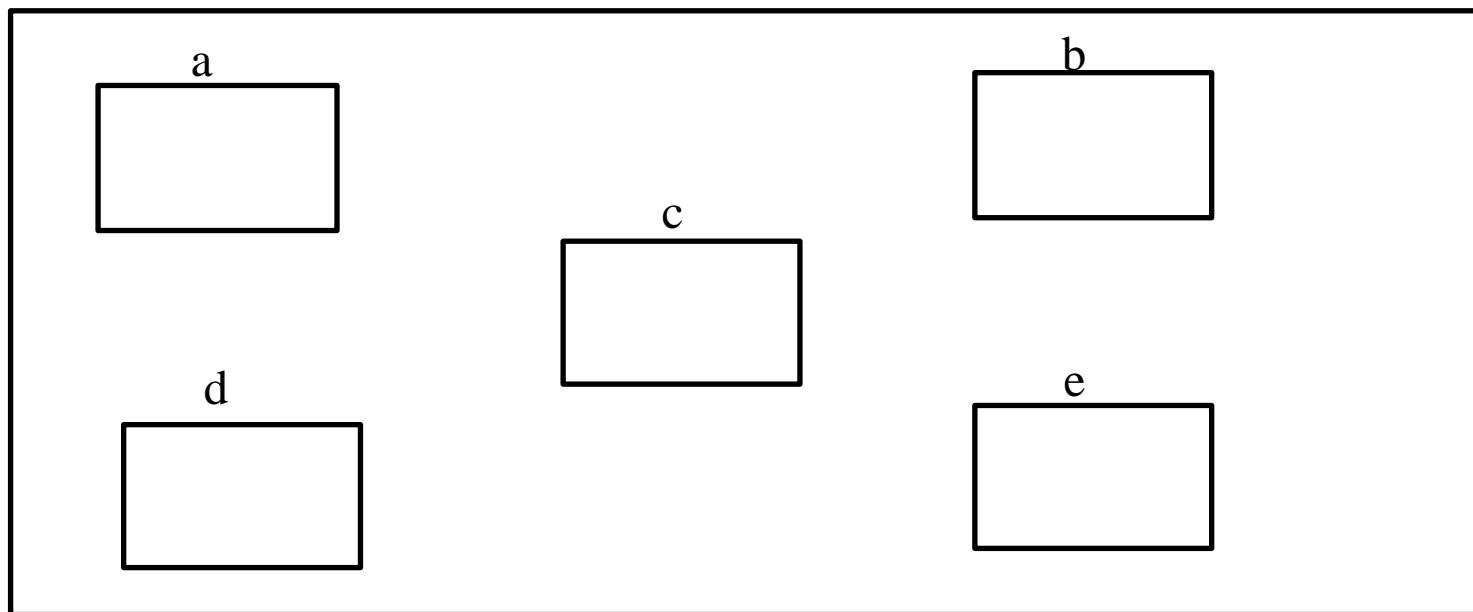
This shows that whenever we declare a variable, in that at a time we can store only one value .



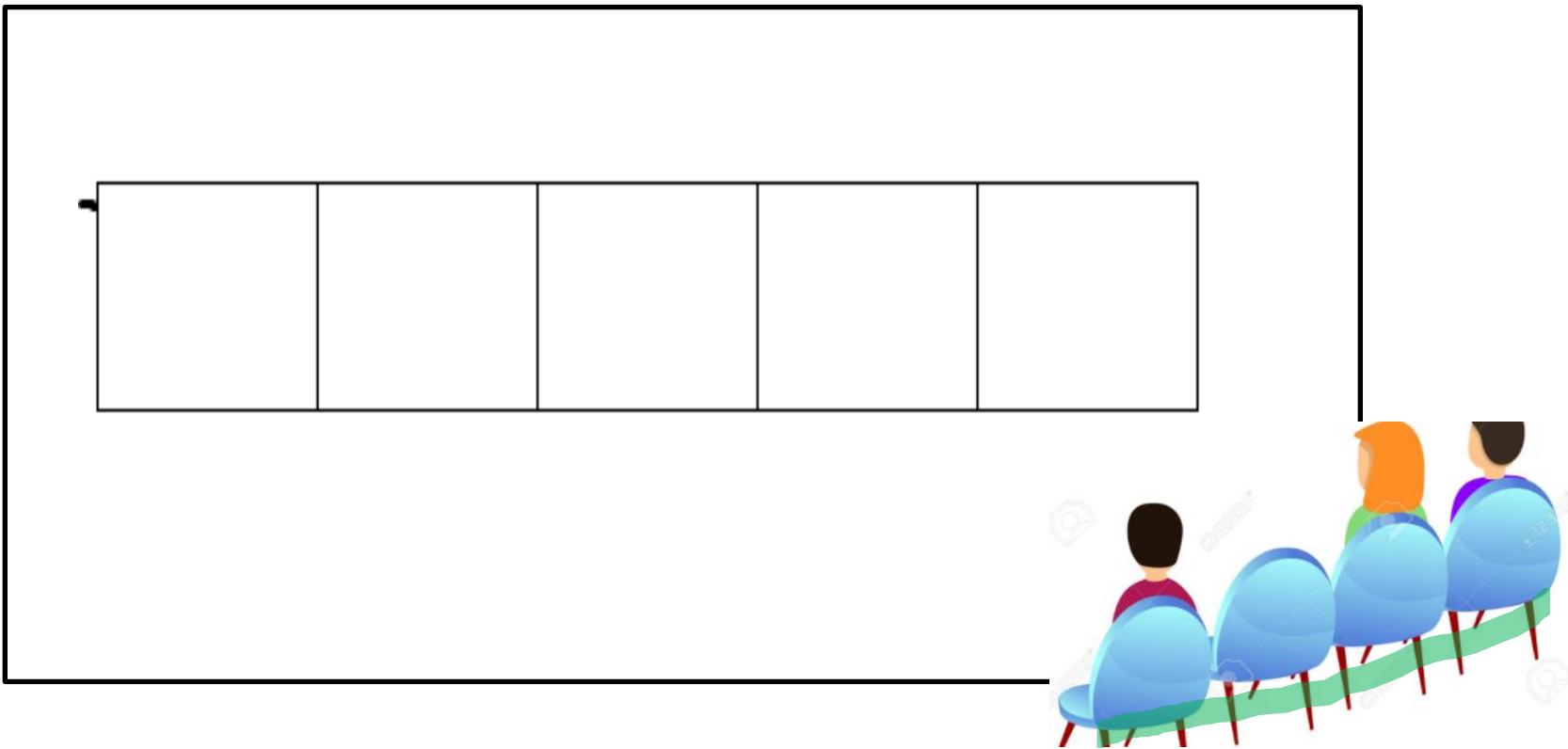
For example I am writing a program where I need to store 5 integer values. For this I must declare 5 individual variables like,

```
int a,b,c,d,e;
```

memory



Instead of occupying 5 individual locations I want to occupy continuous locations like  
**memory**





# General Concept of Array

What is an array?

An array is a collective name given to a group of related quantities belonging to same data type

Array declaration

```
datatype arrayname[size];
```



## What is an integer array?

An **integer** array is a collective name given to a group of related quantities belonging to **integer** data type

### Integer Array declaration

```
int arrayname[size];
```



# What is floating point array?

Similar to the int array, **float** array is a collective name given to a group of related quantities belonging to **floating point** data type.

## Floating point Array declaration

```
float arrayname[size];
```

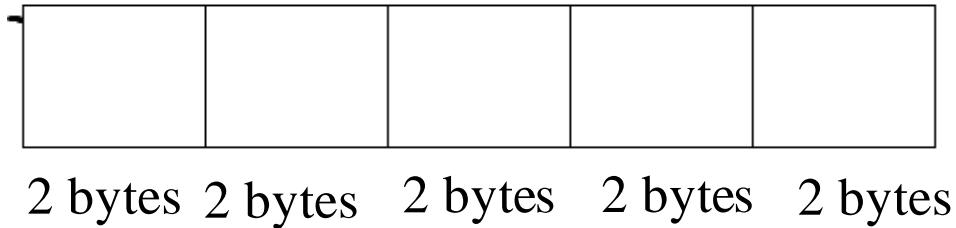


## Example

**int a [ 5 ];**

↓  
data  
type  
↓  
array  
name  
↓

size of  
the array





## The following are some examples of array declaration

```
int x[100];
```

// x is an array to store maximum 100 integers

```
float height[50];
```

// height is an array to store heights of maximum 50 persons

```
int fib[15];
```

// Fib is an array to store maximum 15 elements of fibinocci sequence.



# How to access array elements?

`int a[5];`

0

1

2

3

4

22	-34	65	45	36
----	-----	----	----	----

`a[0]`

`a[1]`

`a[2]`

`a[3]`

`a[4]`

`a[0]=22`

`a[1]=-34`

`a[3]=45`

`a[2]=65`

`a[4]=36`



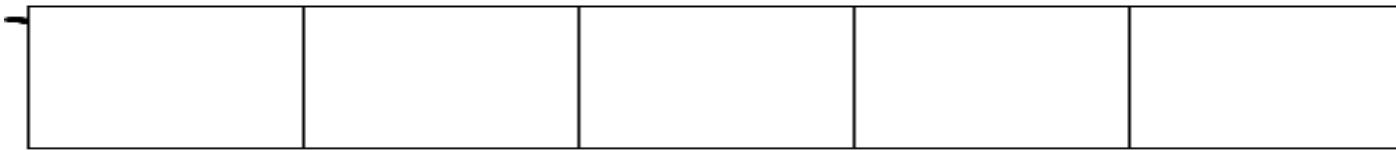
## Example

```
float a [ 5 ];
```

↓  
data  
type

array  
name

size of  
the array





## How to access array elements?

int a[5];

0	1	2	3	4
3.14	-0.465	8.5	4.5	33.6
a[0]	a[1]	a[2]	a[3]	a[4]

**a[0]=3.14**

**a[1]=-0.465**

**a[3]=4.5**

**a[2]=8.5**

**a[4]=33.6**



## ❖ Integer Array

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

**datatype arrayname[size];**

## ❖ Float Array

```
float a[5];
```

**datatype arrayname[size];**

```
#include<stdio.h>
int main()
{
    int a[2];
    a[0]=12;
    a[1]=-20;
    printf("%d%d",a[0],a[1]);
    return 0;
}
```



```
#include<stdio.h>
int main()
{
    int a[2];

    scanf("%d%d",&a[0],&a[1]);
    printf("%d%d",a[0],a[1]);
    return 0;
}
```

```
#include<stdio.h>
int main()
{
    float a[2];

    scanf("%f%f",&a[0],&a[1]);
    printf("%f%f",a[0],a[1]);
    return 0;
}
```



```
#include<stdio.h>
int main()
{
    int a,b;
    //create a and b
    a=10;
    b=20;
    //assigns value to a and b
    printf("%d%d",a,b);
    //prints the value of a and b
return 0;
}
```

## Assigning Values-Input

```
#include<stdio.h>
int main()
{
    int a,b;
    //create a and b
    scanf("%d%d",&a,&b);
    //takes values of a and b
    //from user
    printf("%d%d",a,b);
    //prints the value of a and b
return 0;
}
```

## Taking Values as Input at runtime



```
#include<stdio.h>
int main()
{
    int a[2];
    //create an array name as a
    a[0]=10;
    a[1]=20;
    //assigns value to a[0] and a[1]
    printf("%d%d",a[0],a[1]);
    //prints value of a[0] and a[1]
    return 0;
}
```

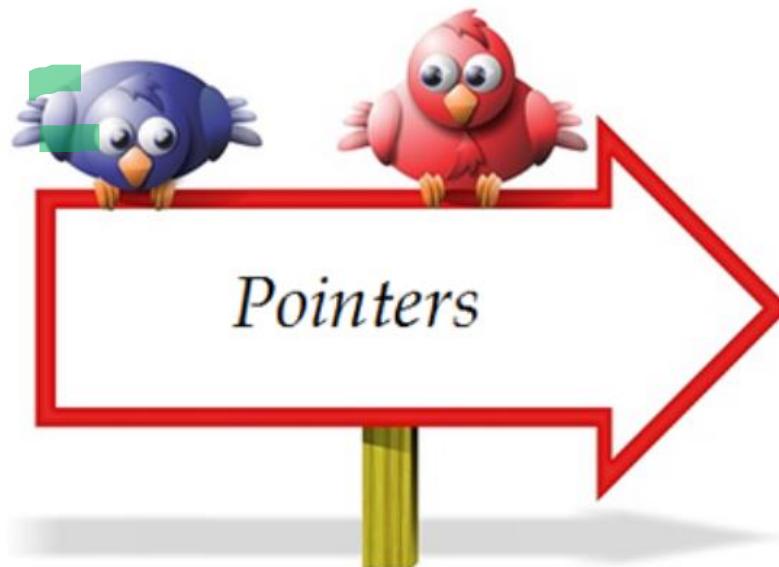
## Array Assigning

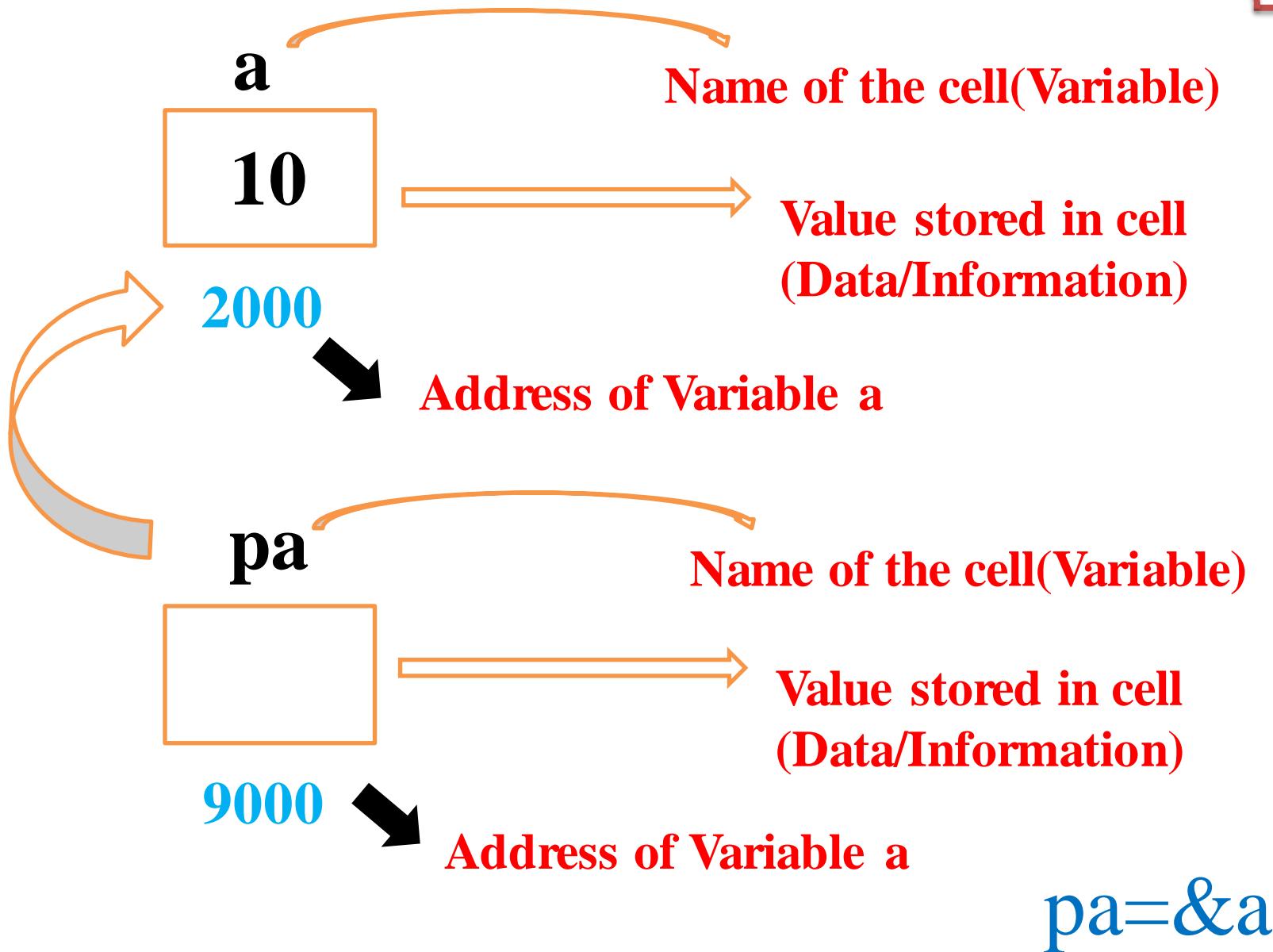
```
#include<stdio.h>
int main()
{
    int a[2];
    //create an array name as a
    scanf("%d%d",&a[0],&a[1]);
    //takes values of a[0] and a[1]
    //from user
    printf("%d%d",a[0],a[1]);
    //prints the value of a[0] and a[1]
    return 0;
}
```

## Array Reading Values-Taking input through keyboard

# Pointer

- ❖ Access to Variables by using the address of that variable.
- ❖ Variable that stores the address of another variable.







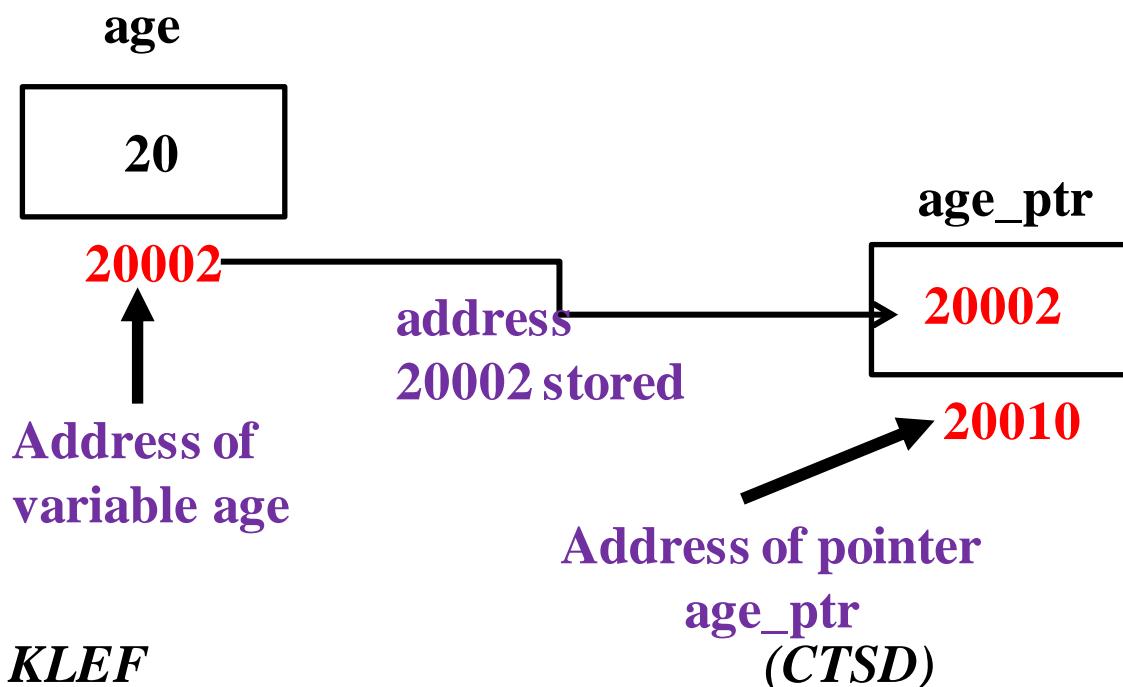
# Integer Pointer?

A special type of variable to store address of another variable. An integer pointer is declared by using **int** keyword and **\*** (asterisk) and only **points to integer** variable.

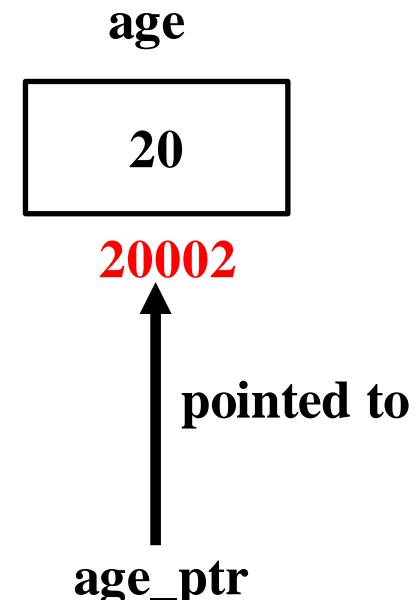
```
int age = 20;
```

```
int *age_ptr;
```

```
int age_ptr = &age; //address of age is  
assigned to pointer age_ptr.
```



*So pointer points variable like this.*





**&var** is read as “address of var”.

**\*ptr** is read as “value at ptr”.

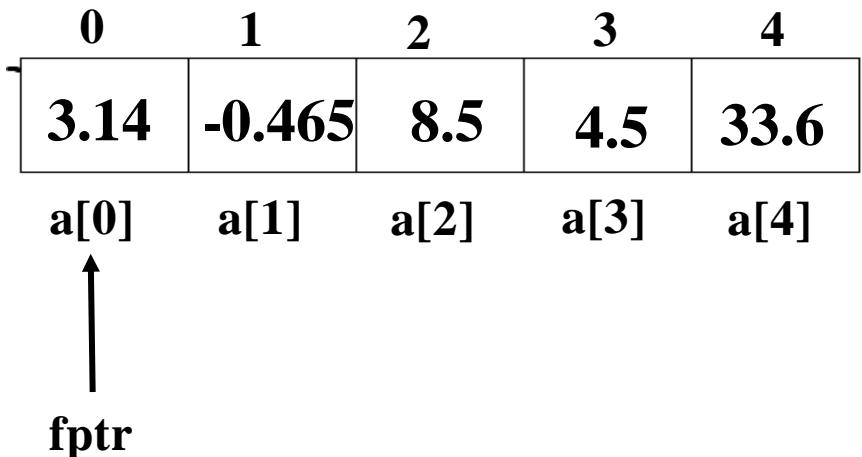
Address of first cell of an array is called base address of that array.

**Float array** element can be accessed using **float pointer** by assigning **base address** to float pointer and **adding cell index**.

```
float *pa=&a;
```

```
*(pa + 1); // equivalent to a[1]
```

```
pa[2]; // equivalent to a[2]
```





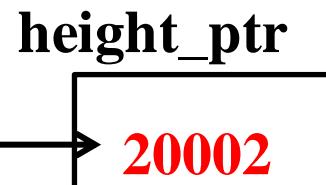
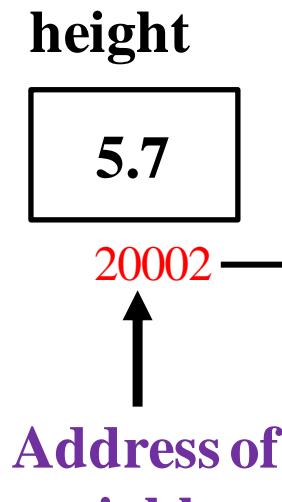
# Floating-Point Pointer?

A special type of variable to store address of another variable. A float pointer is declared by using **float** keyword and **\*** (asterisk) and only points to **float** variable.

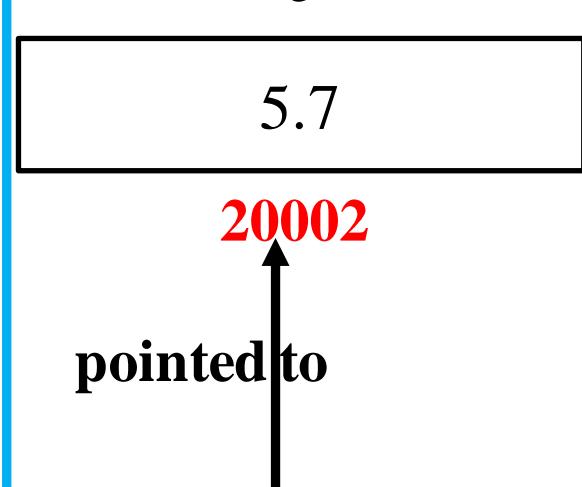
```
float height = 5.7;
```

```
float *height_ptr;
```

```
float height_ptr = &height;  
//address of height is assigned to pointer height_ptr.
```



*So pointer points to variable like this.*



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

```
int main()
```

```
{
```

```
    int a, b, *pa, *pb, sum;
```

```
    a=10;
```

```
    b=20;
```

```
    pa = &a;
```

```
    pb = &b;
```

```
    sum = *pa + *pb;
```

```
    printf("Sum of the numbers = %d\n", sum);
```

```
    return 0;
```

```
}
```

*KLEF*

a

b

10

20

2000

4056

pa

pb

2000

4056

7000

2030

sum

30

9012



## ❖ Integer Pointer

```
int a=5;
```

```
int *pa;
```

**datatype \*pointervariablename;**

## ❖ Floating Pointer

```
float a=7.4;
```

```
float *pa;
```

**datatype \*pointervariablename;**

**pa=&a;**

## ❖ Integer Array Pointer

```
int a[10];
```

```
int *pa;
```

## ❖ Float Array Pointer

```
float a[10];
```

```
float *pa;
```

**pa=a;**



```
#include <stdio.h>
int main()
{
    int a[2] ;  a[0]=10;a[1]=20;
    int *p = a;
    printf("a[0] = %d, a[1] = %d, *p = %d",
           *p, *(p+1), *p);
    return 0;
}
```

|



%**p** is used to print pointer addresses. If you still didn't get it, the format specifier cast the value **of** the respective variable into their format data type. %**d** cast it into an signed integer while %**u** cast it into an unsigned integer.

```
#include <stdio.h>
int main(void)
{
    int a=9,*pa;
    pa = &a;
    printf("%d",*pa);
    printf("\t%p",a);
    printf("\t%d",&a);
    return 0;
}
```



```
#include <stdio.h>
int main()
{
    int var = 5;
    int *p=&var;
    printf("var value: %d", var);
    printf("\naddress of var: %p", &var);
    printf("\nPointer value:%d",*p);
    //value of var is used by p
    printf("\npointing to address:\n%p",p);
    //address of var stored in p
    return 0;
}
```

# Assigning Values-Input

```
#include<stdio.h>
int main()
{
    int a,b;      //create a and b
    a=10;          b=20;
    int *pa,*pb;
    //create pointer variables to a and b
    pa=&a;          pb=&b;
    //stores the address of a to pa
    //and b to pb
    printf("%d%d",*pa,*pb);
    printf("%d%d",a,b);
    //prints the value of a and b
    //prints the value of *pa and *pb
    return 0;
}
```

# Taking Values as Input at runtime

```
#include<stdio.h>
int main()
{
    int a,b;      //create a and b
    scanf("%d%d",&a,&b);
    //takes values of a and b from user
    pa=&a;          pb=&b;
    //stores the address of a to pa
    //and b to pb
    printf("%d%d",*pa,*pb);
    printf("%d%d",a,b);
    //prints the value of a and b
    //prints the value of *pa and *pb
    return 0;
}
```





```
#include<stdio.h>
int main()
{
    int a[2];
    a[0]=10;
    a[1]=20;
    int *pa=a;
    printf("%d",*(pa));

    printf("%d",*(pa+1));
    return 0;
}
```

```
#include<stdio.h>
int main()
{
    int a[2];
    scanf("%d%d",a[0],a[1]);
    int *pa=a;
    printf("%d",*(pa));
    printf("%d",*(pa+1));
    return 0;
}
```



`&var` is read as “address of var”.

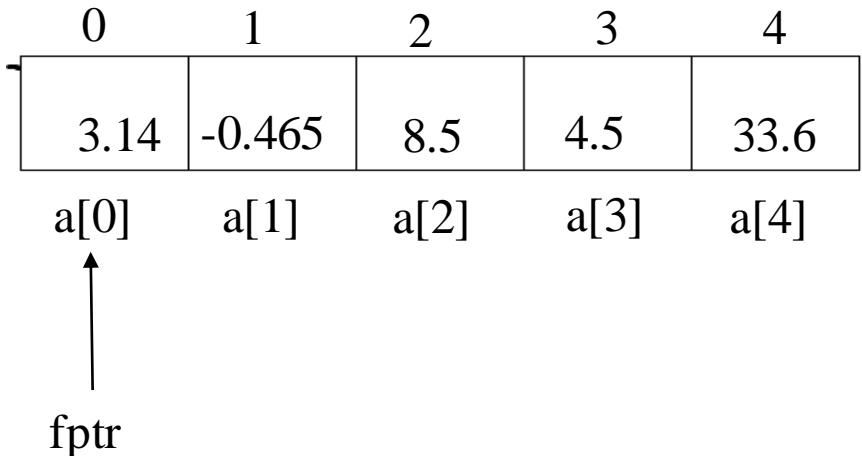
`*ptr` is read as “value at ptr”.

Address of first cell of an array is called base address of that array. **Float array** element can be accessed using **float pointer** by assigning **base address** to float pointer and **adding cell index**.

```
float *fptr = a;
```

```
*(fptr + 1); // equivalent to a[1]
```

```
fptr[2]; // equivalent to a[2]
```





Character holds an ASCII Value( 0 to 127)  
rather than character itself

Lowercase    97 to 122    Uppercase    65 to 90

```
#include<stdio.h>
int main()
{
    char ch;
    scanf("%c",&ch);  ch='p';
    if(((ch>=97)&&(ch<=122))||(ch>=65) && (ch<=90)))
        printf("%c is character",ch);
    else
        printf("not a character");
    return 0;
}
```



- ❖ Every variable in C programming has two properties: type and storage class.
- ❖ Type refers to the data type of a variable. And, storage class determines the scope, visibility and lifetime of a variable.
- ❖ There are 4 types of storage class:
  1. automatic
  2. external
  3. static
  4. register

# Storage Class and its Purpose



## auto

default storage class(block)

GarbageValue

## extern

global variable.(Program)

0

## Static

local variable which returns value even when control transfers to function call(block)

0

## register

store inside the register.(block)

GarbageValue



# Example of Automatic Storage Class

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

```
{
```

```
    auto int i=1;
```

```
{
```

```
    auto int i=2;
```

```
{
```

```
    auto int i=3;
    printf("\n%d",i);
```

```
}
```

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

```
    return 0;
```

```
}
```

This variable scope and lifetime is within main method

This variable scope and lifetime is within this block

Output: 3    2    1



# Example for static variable

```
#include<stdio.h>
void increment();
void main()
{
    increment();
    increment();
    increment();
}
void increment()
{
    static int i=1;
    printf("%d\t",i);
    i++;
}
```

Output:

1 2 3

This statement executes ONLY when the increment() function is called for the first time.

The variable i exists in memory till the program ends



# Dif. b/w auto and static storage class

```
#include<stdio.h>
void increment();
void main()
{
    increment();
    increment();
    increment();
}
void increment()
{
    auto int i=1;
    printf("%d\t",i);
    i=i+1;
}
```

Output:

1 1 1



# Predict the output

```
#include <stdio.h>
void func(void);
int main()
{
    func();
    func();
    return 0;
}
void func()
{
    static int i = 5;
    int j=5;
    i=i+1;
    j=j+1;
    printf("i is %d and j is %d\n",i, j);
}
```

Output:

i is 6 and j is 6  
i is 7 and j is 6

# Repeat a Statement

**Loop**



**Loops**





**Repeat a Statement**



**Loops**



**KLEF**

**(CTSD)**

**BES-1**



1

2

5

**for(initialization; ConditionChecking; Increment/Decrement)**

{

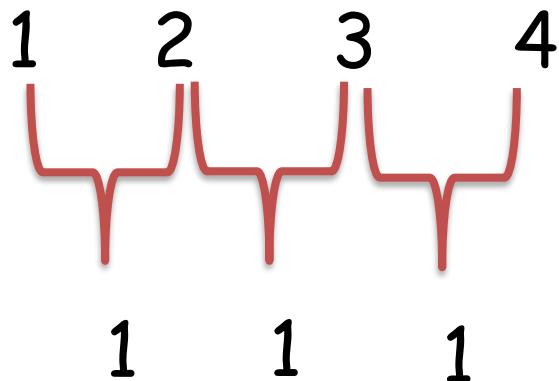
**set of statements;**  
}

3

4



Start

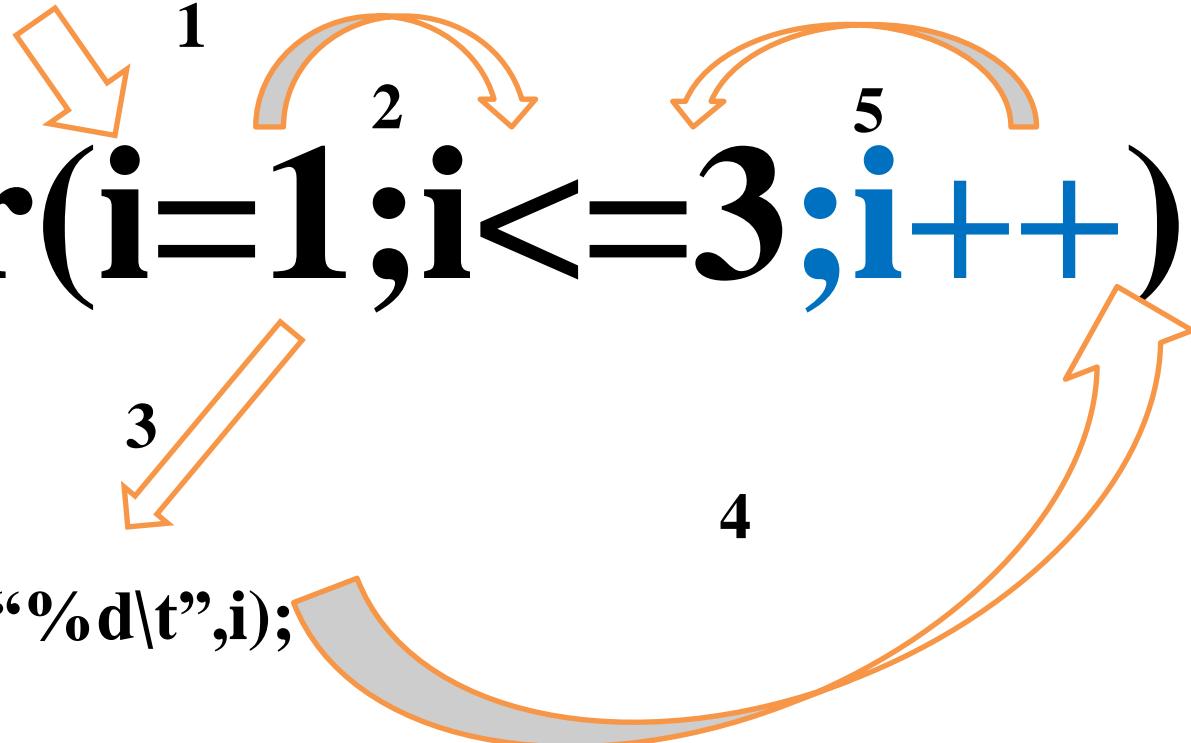


end

**for(i=1;i<=10;i++)**

i=1	$1 \leq 10$	1
i=2	$2 \leq 10$	2
i=3	$3 \leq 10$	3
i=4	$4 \leq 10$	4
i=5	$5 \leq 10$	5
i=6	$6 \leq 10$	6
i=7	$7 \leq 10$	7
i=8	$8 \leq 10$	8
i=9	$9 \leq 10$	9
i=10	$10 \leq 10$	10
i=11	$11 \leq 10$	

```
for(i=1;i<=3;i++)  
{  
    printf("%d\t",i);  
}
```





```
i=1;1
While(2condition)5
{
    statements;3 4
    increment/decrement;
}
```



i=1;<sup>1</sup>

do<sup>2</sup>

{

statements;

increment/decrement;

} While(condition);

5



## Break:

Force to Come out of the loop if the condition is true.

```
for(i=1;i<=5;i++)  
{  
    if(i==3)  
        break;  
    printf("%d\t",i);
```

### Output

1 2

## Continue:

Expect that Condition remaining statements are executed.

```
for(i=1;i<=5;i++)  
{  
    if(i==3)  
        continue;  
    printf("%d\t",i);  
}
```

### Output

1 2 4 5

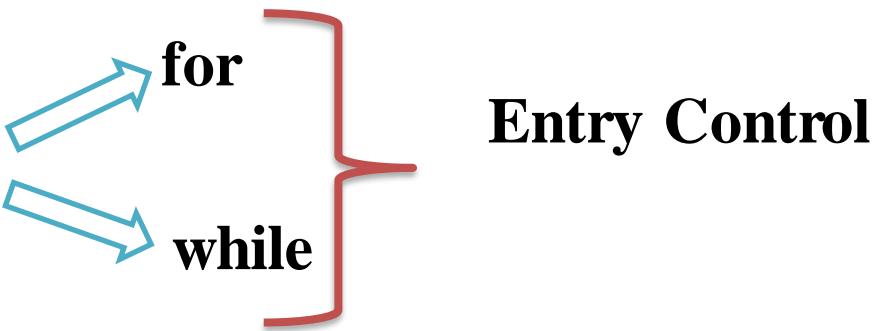


```
i=1;  
while(i<=5)  
{  
printf("%d\n", i);  
i++;  
}
```

```
int i=1;  
do  
{  
printf("%d ", i);  
i++;  
} while(i <= 10);
```

## Which loop to select

- Analyse the problem and check whether it requires a pretest or a post test loop.
- If pretest required



- If posttest required

→ do-while }      Exit Control



# File handling in C

- A file is a place on the disk where a group of related data is stored.
- FILE is a structure declared in stdio.h .
- We have to use file pointer, a pointer variable that points to a structure FILE.

|



# File management function

function	purpose
<b>fopen ()</b>	Creating a file or opening an existing file
<b>fclose ()</b>	Closing a file
<b>fprintf ()</b>	Writing a block of data to a file
<b>fscanf ()</b>	Reading a block data from a file

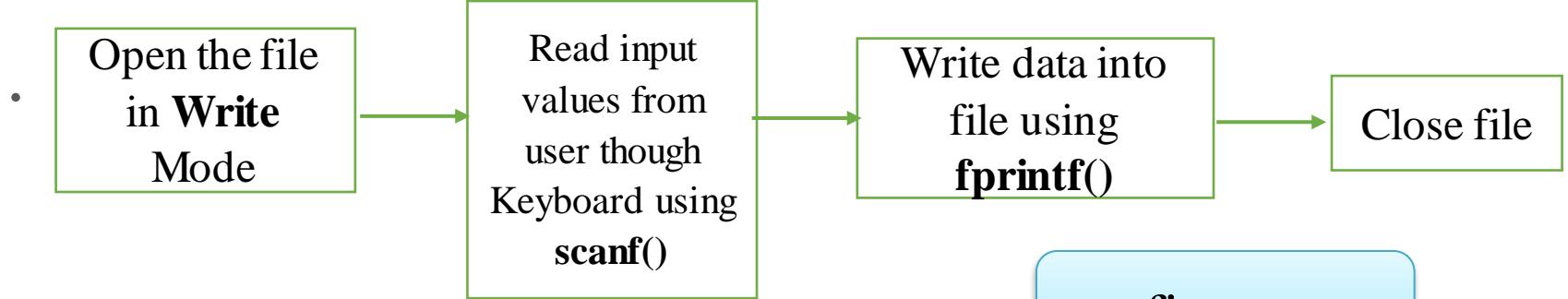


# Modes of files

Mode	Description
r	Open a file for reading. If a file is in reading mode, then no data is deleted if a file is already present on a system.
w	Open a file for writing. If a file is in writing mode, then a new file is created if a file doesn't exist at all. If a file is already present on a system, then all the data inside the file is truncated, and it is opened for writing purposes.
a	Open a file in append mode. If a file is in append mode, then the file is opened. The content within the file doesn't change.
r+	open for reading and writing from beginning
w+	open for reading and writing, overwriting a file
a+	open for reading and writing, appending to file



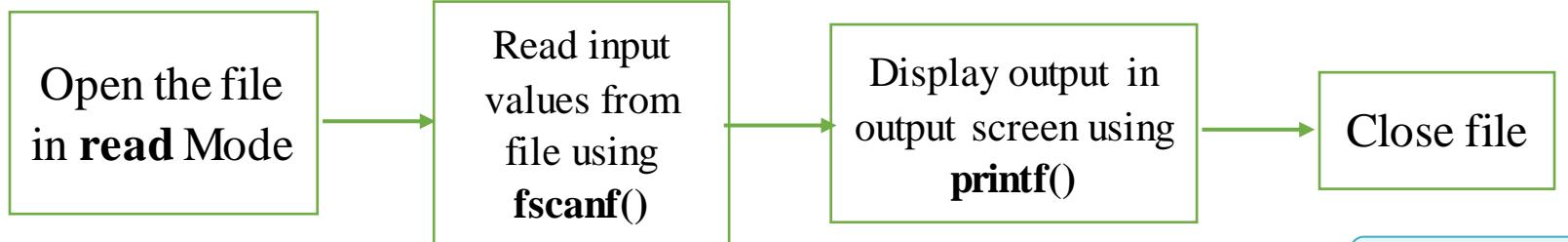
# Write operation to a file



first.txt

50

Enter num:



Output

Value of n=50



## File Operations

- ❖ **Create/Open**
- ❖ **Read data from file**
- ❖ **Write data into file**
- ❖ **Close the file**

## File Management Operations

- ❖ **fopen()**-----Create/Open
- ❖ **fclose()**-----closing the file
- ❖ **fprintf()**-----writing block of data to a file.
- ❖ **fscanf()**-----reading block of data from a file.

## To Create

- Create/open a file

Syntax:

**FILE \*filepointername;**

**Ex: FILE \*fp;**

We can open the file in two modes “read” and “write”.

Syntax:

**filepointername=fopen(“filepathname”,“modeofoperation”);**

**Ex: fp=fopen(“first.txt”,“w”);**

**fp=fopen(“first.txt”,“r”);**

## To Close

- Close a file

Syntax:

**fclose(filepointername);**

**Ex : fclose(fp);**

## To Print data into file

- **fprintf**

Syntax: **fprintf(filepointername,"typespecifier",variblename);**

**Ex: fprintf(fp,"%d",a);**

## To Read data from file

- **fscanf**

Syntax: **fscanf(filepointername,"typespecifier",&variblename);**

**Ex: fscanf(fp,"%d",&a);**



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

```
int main()
```

```
{
```

```
    int a;
```

```
    FILE *fp;
```

```
    fp=fopen("first.txt","w");
```

```
    printf("enter a value");
```

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

```
    fprintf(fp,"value in file is %d",a);
```

```
    fclose(fp);
```



```
    fp
```

```
        Value in  
        file is 47
```

```
fp=fopen("first.txt","r");
```

```
fscanf(fp,"%d",&a);
```

```
printf("a value taken from file accessed here is %d",a);
```

```
fclose(fp);
```

```
return 0;
```

```
}
```

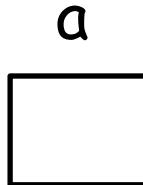


```
#include<stdio.h>
```

```
int main()
```

```
{
```

```
    int a;
```



```
    FILE *fp;
```

```
    fp=fopen("first.txt","w");
```

```
    printf("enter a value");
```

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

```
    fprintf(fp,"value in file %d",a);
```

```
    fclose(fp);
```

```
fp
```

Value in file is  
47

```
fp=fopen("first.txt","r");
```

```
fscanf(fp,"%d",&a);
```

```
printf("value taken from file is %d",a);
```

```
fclose(fp);
```

```
return 0;
```



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

```
int main()
```

```
{
```

```
    int a,b;
```

```
    FILE *fp;
```

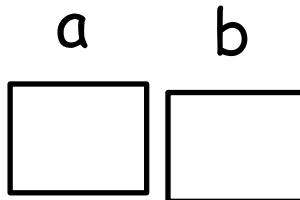
```
    fp=fopen("first.txt","w");
```

```
    printf("enter a and b values:");
```

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

```
    fprintf(fp,"%d\t%d",a,b);
```

```
    fclose(fp);
```



```
    fp
```

```
    10 20
```

```
    fp=fopen("first.txt","r");
```

```
    fscanf(fp,"%d%d",&a,&b);
```

```
    printf("from file %d",a+b);
```

```
    fclose(fp);
```

```
    return 0;
```



```
#include<stdio.h>
int main()
{
    int a[3],i;
    FILE *fp;
    fp=fopen("first.txt","w");
    for(i=0;i<3;i++)
        scanf("%d",&a[i]);
    for(i=0;i<3;i++)
        fprintf(fp,"%d\t",a[i]);
    fclose(fp);

    fp=fopen("first.txt","r");
    for(i=0;i<3;i++)
        fscanf(fp,"%d",&a[i]);
    printf("%d\t",a[i]);

    fclose(fp);
    return 0;
}
```



```
#include<stdio.h>
int main()
{
    float Salary,bonus;
    char gender;
    FILE *fp;
    fp=fopen("Salaray.txt","w");
    scanf("%c%f",&gender,&Salary);
    fprintf(fp,"%c %f",gender,Salary);
    fclose(fp);
    fp = fopen("Salary.txt","r");
    fscanf(fp,"%c%f",&gender,&Salary);
    if(gender=='f'|| gender == 'F')
        bonus = 0.1*Salary;
    else
        bonus = 0.05*Salary;
    fclose(fp);
    printf("The Bonus: %f",bonus);
    return 0;
}
```



Category	Operator	Associativity
Postfix	() [] -> . ++ - -	Left to right
Unary	+ - ! ~ ++ - - (type)* & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR		Left to right
Logical AND	&&	Left to right
Logical OR		Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %= >>= <<= &= ^=  =	Right to left
Comma	,	Left to right



```
int a = 20;           int b = 10;           int c = 15;           int d = 5;  
int e;
```

e = (a + b) \* c / d; // ( 30 \* 15 ) / 5

```
printf("Value of (a + b) * c / d is : %d\n", e );
```

e = ((a + b) \* c) / d; // (30 \* 15 ) / 5

```
printf("Value of ((a + b) * c) / d is : %d\n" , e );
```

e = (a + b) \* (c / d); // (30) \* (15/5)

```
printf("Value of (a + b) * (c / d) is : %d\n", e );
```

e = a + (b \* c) / d; // 20 + (150/5)

```
printf("Value of a + (b * c) / d is : %d\n" , e );
```



```
int a = 20;    int b = 10;        int c = 15;  
int d = 5;  
  
int e;
```

$$e = a+b*c/d$$

```
int a = 10;    int b = 4;        int c = 3;  
int d = 2;  
  
int e;
```

$$e = a+b*c/d$$



Assume that the value of  $a = 5$ ,  $b = 6$ ,  $c = 8$ ,  $d = 2$ , and  $e = 1$ . Using these values, let us try to evaluate and find the result of the following expression using the above tables.

```
a + b && b / c % d * e || 5 <= 0 != a + b - d
```



$5 + 6 \&\& 6 / 8 \% 2 * 1 || 5 <= 0 != 5 + 6 - 2$  (substitute values)

$5 + 6 \&\& 0 \% 2 * 1 || 5 <= 0 != 5 + 6 - 2$  (6 / 8 evaluated)

$5 + 6 \&\& 0 * 1 || 5 <= 0 != 5 + 6 - 2$  (0 % 2 evaluated)

$5 + 6 \&\& 0 || 5 <= 0 != 5 + 6 - 2$  (0 \* 1 evaluated)

$11 \&\& 0 || 5 <= 0 != 5 + 6 - 2$  (5 + 6 evaluated)

$11 \&\& 0 || 5 <= 0 != 11 - 2$  (5 + 6 evaluated)

$11 \&\& 0 || 5 <= 0 != 9$  (11 - 2 evaluated)

$11 \&\& 0 || 0 != 9$  (5 <= 0 evaluated)

$11 \&\& 0 || 1$  (0 != 9 evaluated)

$0 || 1$  (11 && 0 evaluated)



# Array Functions

```
#include<stdio.h>
void display(int ,int);
int main()
{
    int ageArray[] = {2, 8, 4, 12};

    // Passing second and third elements to display()
    display(ageArray[1], ageArray[2]);
    return 0;
}
void display(int age1, int age2)
{
    printf("%d\n", age1);
    printf("%d\n", age2);
}
```



## Passing Array Pointer to Function

```
#include<stdio.h>
void display(int *,int *);
int main()
{
    int ageArray[] = {2, 8, 4, 12};

    // Passing second and third elements to display()
    display(&ageArray[1], &ageArray[2]);
    return 0;
}
void display(int *age1, int *age2)
{
    printf("%d\n", *age1);
    printf("%d\n", *age2);
}
```



**Character**

**single letter**

**% c**

ch

a

**String**

**group of letters**

**% s**

ch

k l u \0

## String Declaration

```
char str1[]="klu";
```

```
char str1[5] = "klu";
```

\0 will automatically inserted at the end in this type of declaration

```
char str1[]={‘k’,’L’,’u’,’\0’};
```

```
char str1[5]={‘k’,’L’,’u’,’\0’};
```



# Character

```
#include <stdio.h>

int main() {
    char c;
    printf("Enter a character:");
    scanf("%c", &c);
    // %d displays the integer value of a character
    // %c displays the actual character
    printf("ASCII value of %c = %d", c, c);
    return 0;
}
```

C

a

a=97



# String

```
#include<stdio.h>
int main()
{
    char c[10];
    scanf("%s",c);
    printf("%s",c);
    return 0;
}
```

c	l	u	s	t	e	r	\0		
---	---	---	---	---	---	---	----	--	--



```
#include<stdio.h>
int main()
{
    char str[]={'k','l','u','\0'};
printf("%s",s);
return 0;
}
```

```
#include<stdio.h>
int main()
{
    char str[5]={'k','l','u','\0'};
printf("%s",s);
return 0;
}
```



```
#include<stdio.h>
int main()
{
    char str[5]={"KLU"};
    printf("%s",s);
    return 0;
}
```

```
#include<stdio.h>
int main()
{
    char str[]={"KLU"};
    printf("%s",s);
    return 0;
}
```



```
#include<stdio.h>
#include<string.h>
int main()
{
    char s1[15]="Hello";
    char s2[15]="cluster9";
strcpy(s1,s2);
    printf("%s\t%s",s1,s2);
    return 0;
}
```

```
#include<stdio.h>
#include<string.h>
int main()
{
    char s1[15];
    char s2[15];
strcpy(s1,"Hello");
strcpy(s2,"Cluster9");
    printf("%s\t%s",s1,s2);
    return 0;
}
```



# String Pointers

```
#include <stdio.h>

int main()
{
    char name[] = "Harry Potter";
    char *namePtr;
    namePtr = name;
    printf("%c", *namePtr);    // Output: H
    printf("%c", *(namePtr+1)); // Output: a
    printf("%c", *(namePtr+7)); // Output: o
```



- ❖ switch..case statement
- ❖ Gives ability to make decisions from fixed available choices.
- ❖ Rather making decision based on conditions.
- ❖ Using switch we can write a more clean and optimal code, that take decisions from available choices.



```
#include <stdio.h>

int main() {
    char c;
    printf("Enter a character: ");
    scanf("%c", &c);
    if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z'))
        printf("%c is an alphabet.", c);
    else
        printf("%c is not an alphabet.", c);
    return 0;
}
```



```
#include<stdio.h>
#include<string.h>
int main()
{
    char opt;  int a=7,b=5;  scanf("%c",&opt);
    switch(opt)
    {
        case '+':printf("%d",a+b);
                    break;
        case '-':printf("%d",a-b);
                    break;
        case '/':printf("%d",a/b);
                    break;
        case '*':printf("%d",a*b);
                    break;
        default:printf("Invalid");
                    break;
    }
}
```



~~switch(ch)~~

{

**case 'a': printf("Vowel");**

**break;**

**case 'e': printf("Vowel");**

**break;**

**case 'i': printf("Vowel");**

**break;**

**case 'o': printf("Vowel");**

**break;**

**default: printf("Invalid");**

**break;**



```
switch(ch)
{
case 'a':
case 'e':
case 'i':
case 'o':
case 'u':
case 'A':
case 'E':
case 'I':
case 'O':
case 'U': printf("Vowel");
            break;
}
```



// Program to calculate the sum of array elements by passing to a function

```
#include <stdio.h>

float calculateSum(float age[]);

int main() {
    float result, age[] = {23.4, 55, 22.6, 3, 40.5, 18};

    // age array is passed to calculateSum()
    result = calculateSum(age);
    printf("Result = %.2f", result);
    return 0;
}
```

```
float calculateSum(float age[]) {
    float sum = 0.0;

    for (int i = 0; i < 6; ++i) {
        sum += age[i];
    }

    return sum;
}
```

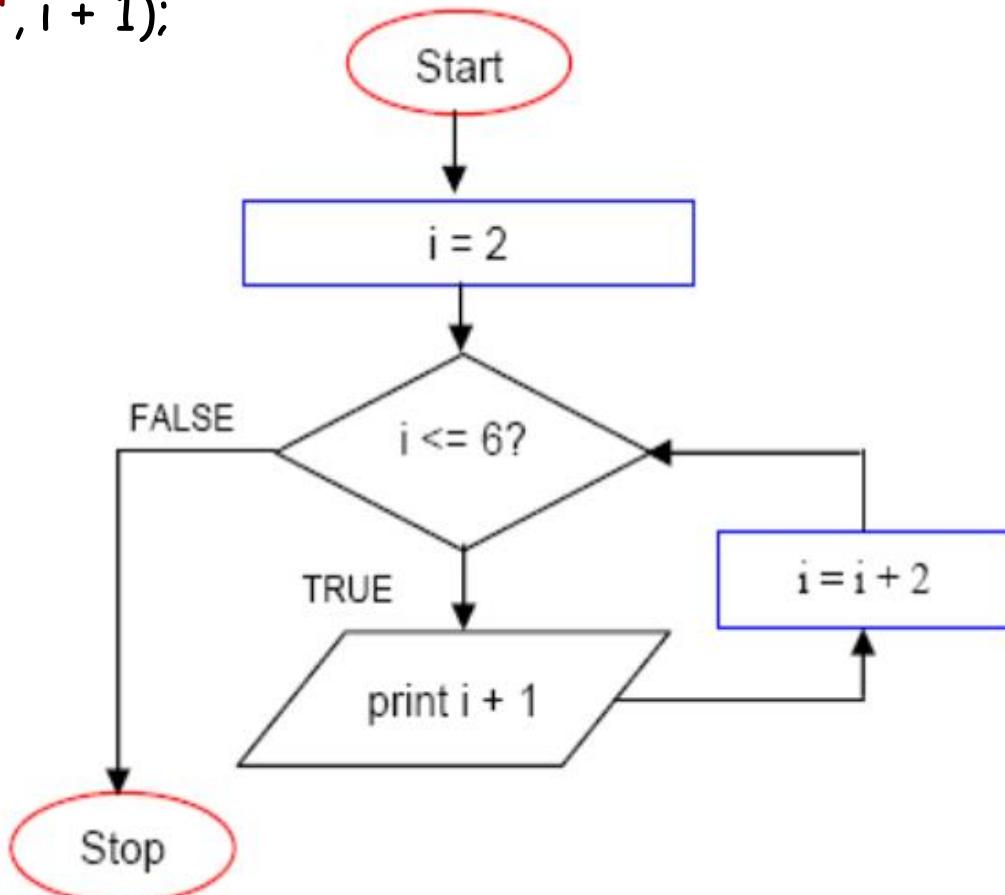


```
#include <stdio.h>

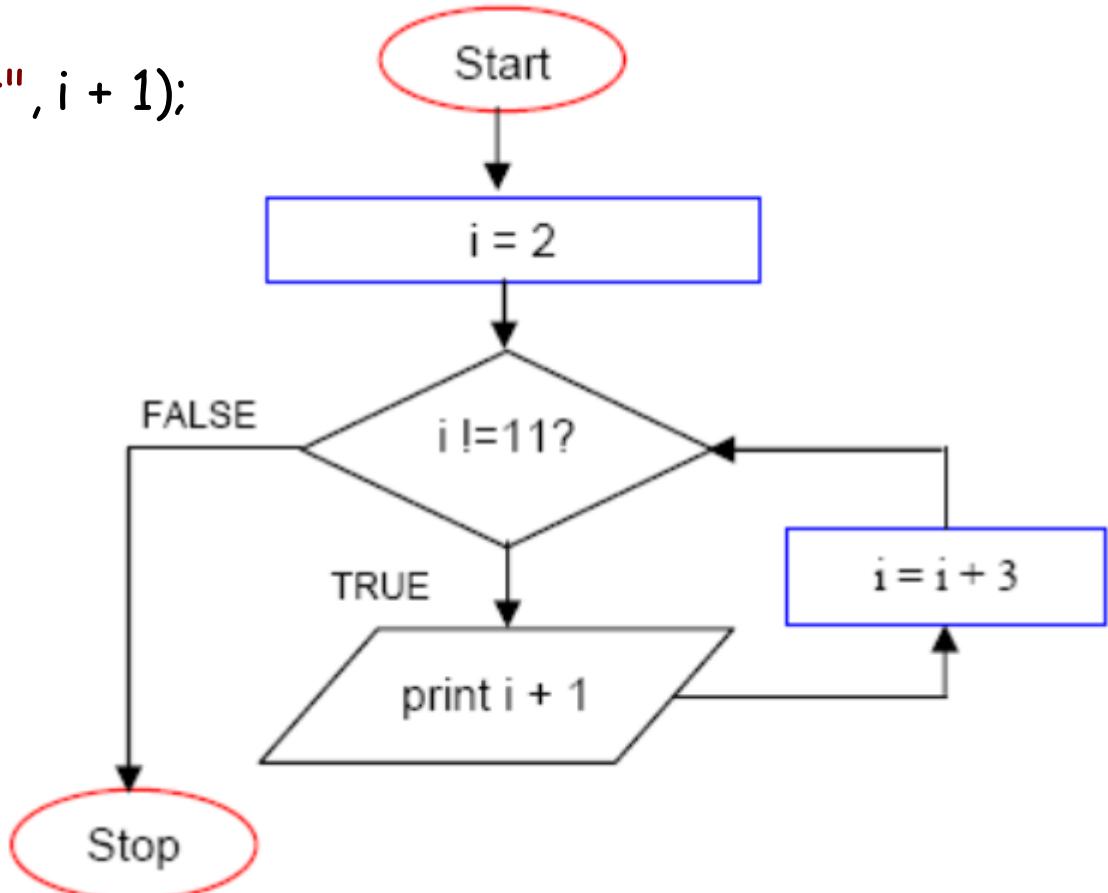
int main()
{
    char name[20] = "Harry Potter";
    printf("%c", *name); // Output: H
    printf("%c", *(name+1)); // Output: a
    printf("%c", *(name+7)); // Output: o

    char *namePtr;
    namePtr = name;
    printf("%c", *namePtr); // Output: H
    printf("%c", *(namePtr+1)); // Output: a
    printf("%c", *(namePtr+7)); // Output: o
}
```

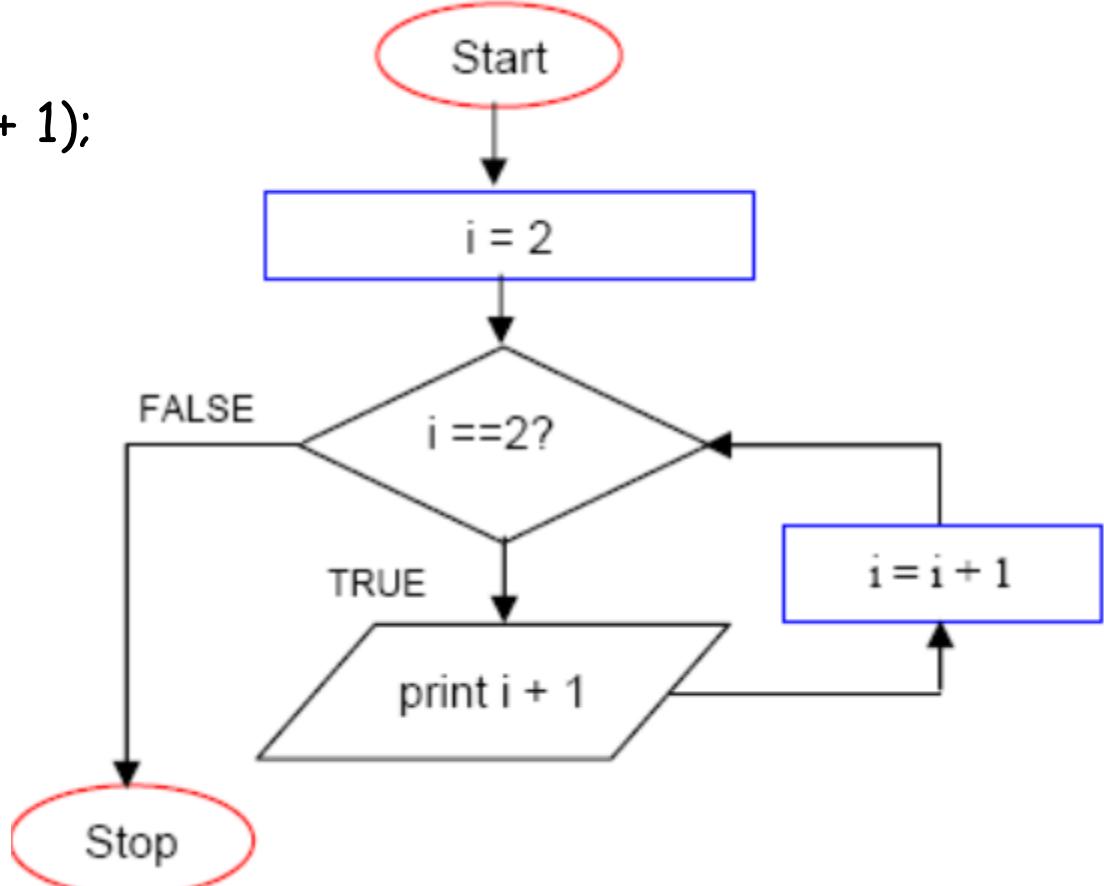
```
for(i = 2; i <= 6; i = i + 2)
{
    printf("%d\t", i + 1);
}
```



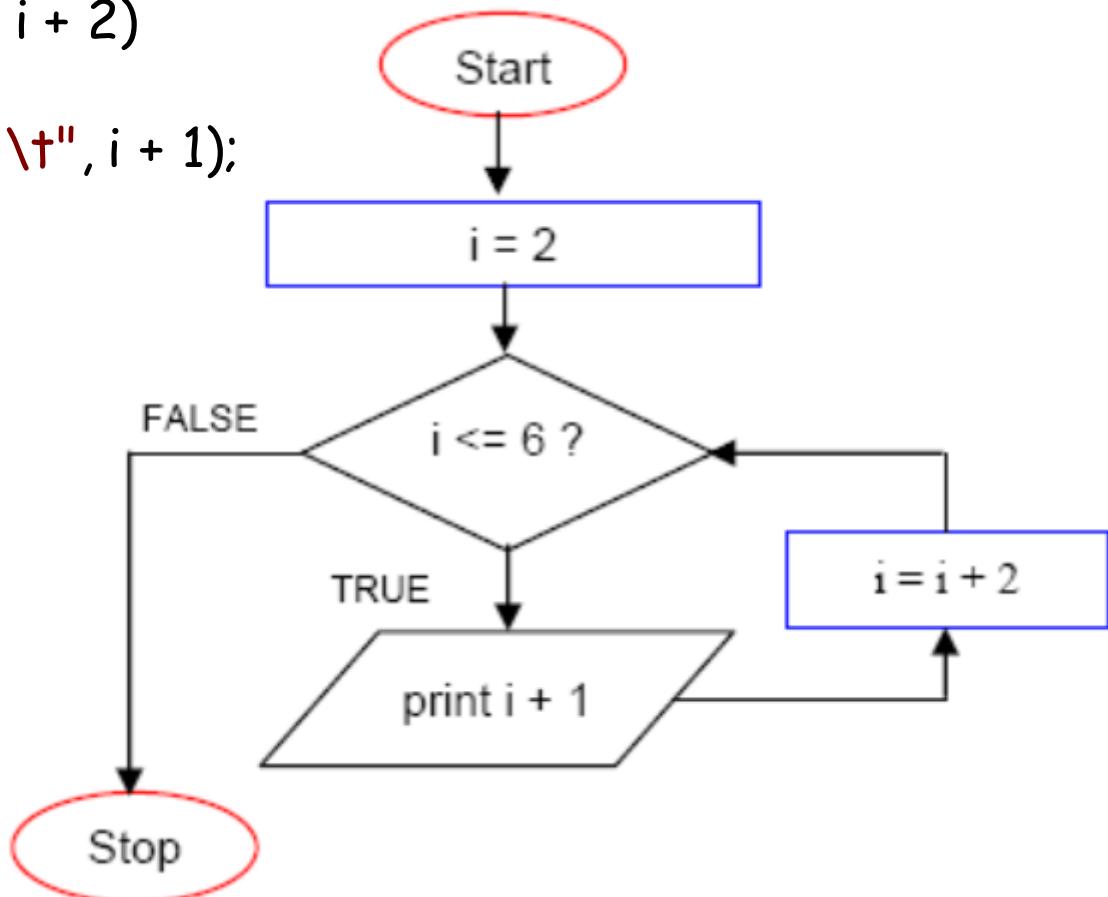
```
for(i = 2; i != 11; i = i + 3)
{
    printf("%d\t", i + 1);
}
```



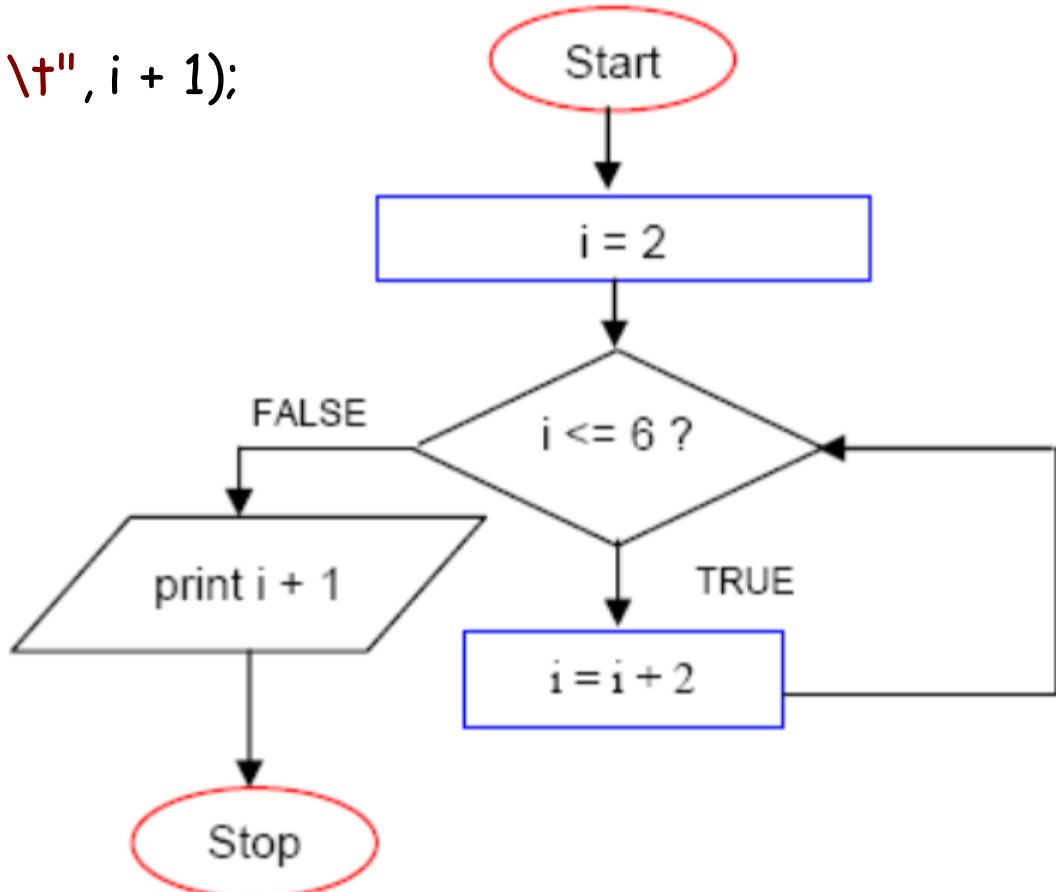
```
for(i = 2; i == 2; i = i + 1)
{
    printf("%d\t", i + 1);
}
```



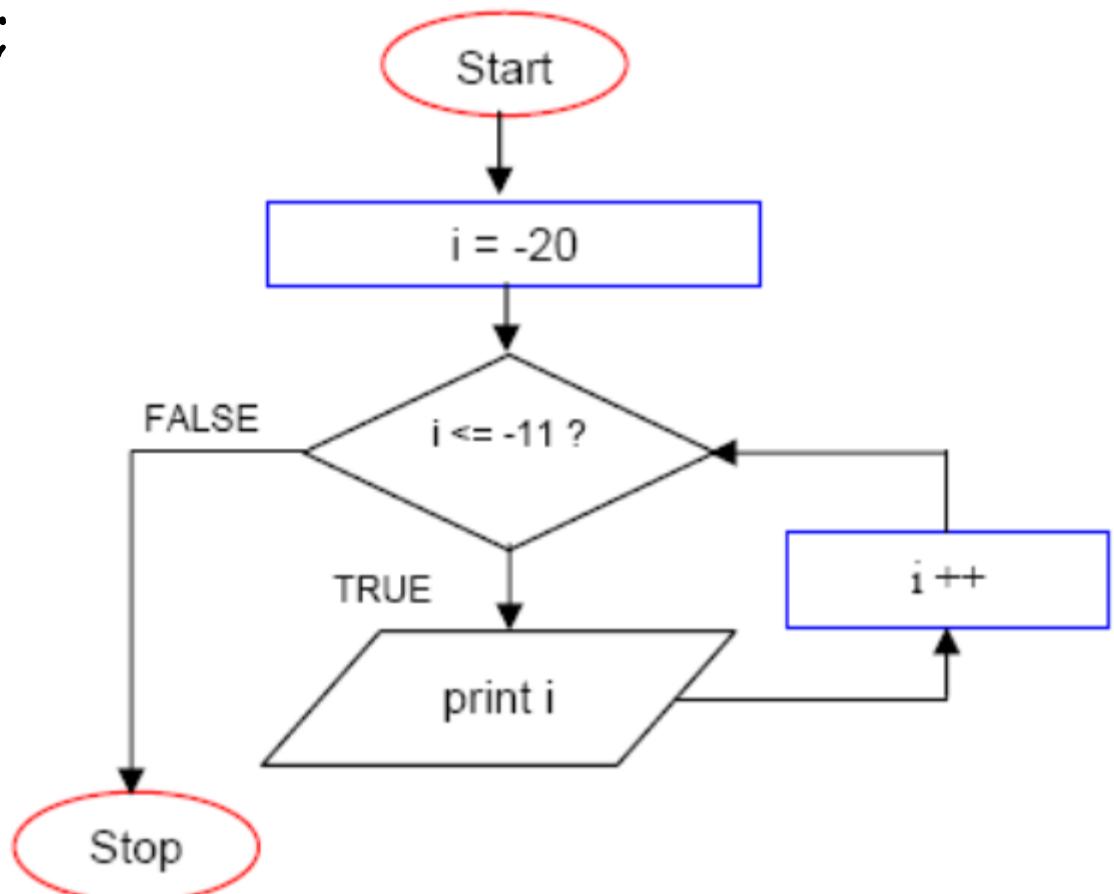
```
for(i = 2; i <= 6; i = i + 2)
{
    printf("%d\t", i + 1);
}
```



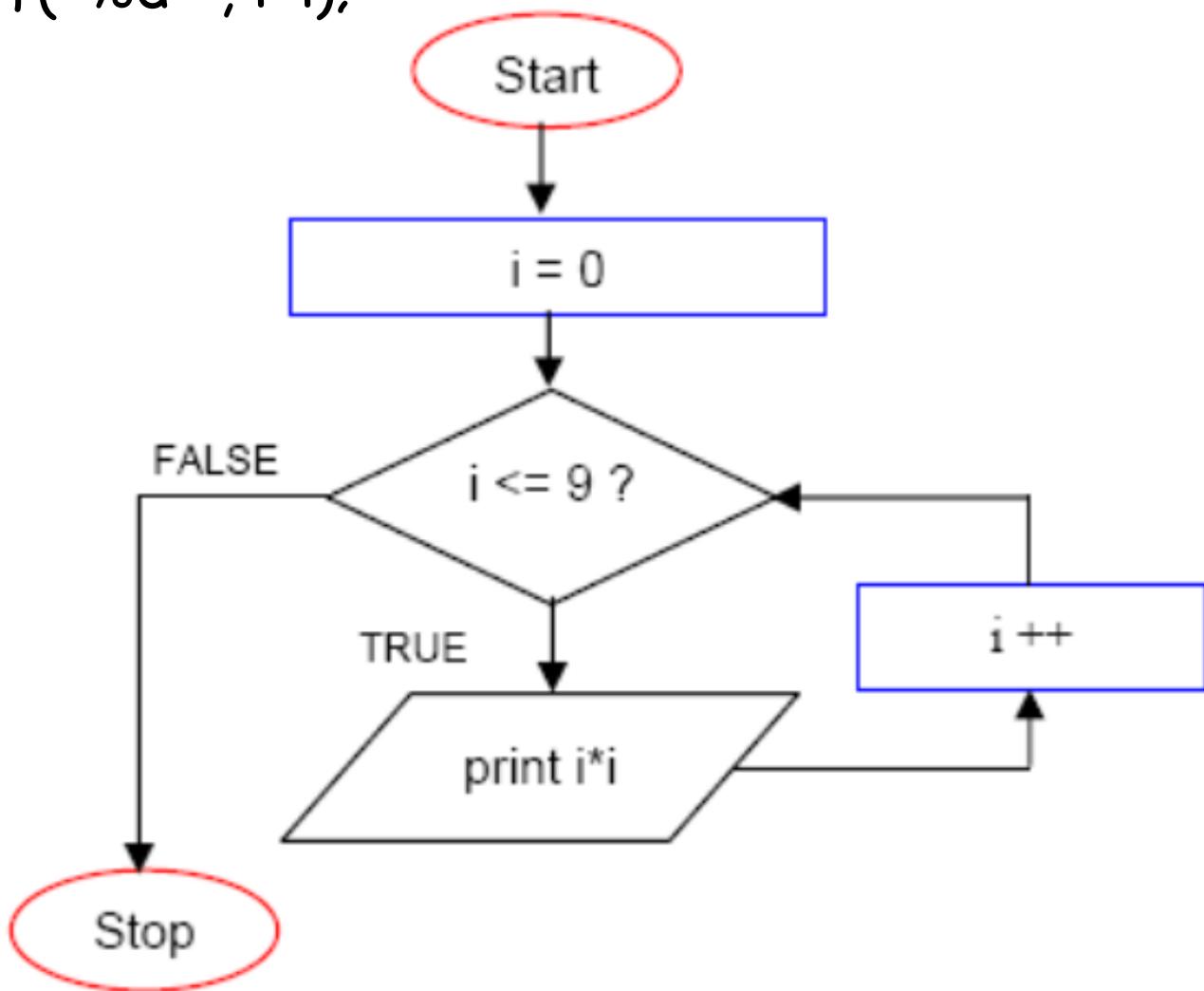
```
for(i = 2; i <= 6; i = i + 2); // note the semicolon here
{
    printf("%d\t", i + 1);
}
```



```
for(i = -20; i <= -11; i++)  
    printf("%d ", i);
```



```
for(i = 0; i <= 10; i++)  
    printf("%d ", i*i);
```



# NOTE



- ❖ All the topics what we have discussed in class are available here.
- ❖ Take time to practice and try do the programs from Work Book.
- ❖ Make sure u learn all the concepts and experiment on each and every concepts .
- ❖ If any minor mistakes are there please ignore.



# Syllabus-Briefly

- ❖ Algorithms & Flowcharts.
- ❖ Introduction to C Programming & Structured Programming.
- ❖ Conditional Statements. (if, if-else, nested-if, switch)
- ❖ Loops (for, while, do-while)
- ❖ Pointers & Arrays



# Comma (,) as Separator and Operator:

- **Comma (,) as separator**

While declaration multiple variables and providing multiple arguments in a function, comma works as a separator.

Example: int a,b,c;

- comma is a separator and tells to the compiler that these (a, b, and c) are three different variables.



## • Comma (,) as an operator

Sometimes we assign multiple values to a variable using comma, in that case comma is known as operator.

Example:

a = 10,20,30;    b = (10,20,30);

- In the first statement, value of a will be 10, because assignment operator (=) has more priority than comma (,), thus 10 will be assigned to the variable a.
- In the second statement, value of b will be 30, because 10, 20, 30 are enclosed in braces, and braces has more priority than assignment (=) operator. When multiple values are given with comma operator within the braces, then right most value is considered as result of the expression. Thus, 30 will be assigned to the variable b.



```
#include<stdio.h>
int main()
{
    int a,b=5,v,y;
    v=(a=10,y=5,a+b);
    printf("a= %d, b= %d\n v=%d\n y=%d",a,b,v,y);
    return 0;
}
```

a = 10  
b = 5  
v = 15  
y = 5



```
int main()
{
    int a,b;
    a = 10,20,30;
    b = (10,20,30);
    //printing the values
    printf("a= %d, b= %d\n",a,b);
    return 0;
}
```

a = 10  
b = 30



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

```
int main()
```

```
{
```

```
int a = 10,20,30;
```

```
int b;
```

```
    b = (10,20,30);
```

```
//printing the values
```

```
printf("a= %d, b= %d\n",a,b);
```

```
return 0;
```

```
}
```

Error because at the time of declaration multiple assignment is not allowed



```
#include<stdio.h>

int main()
{
    int i,j;
    i=(j=3,j+2);

    //printing the values
    printf("i= %d, j= %d\n",i,j);

    return 0;
}
```

i = 5  
j = 3



```
int main()
{
    int i=7,j=3,k;
    k=(i<<j);
    printf("%d\t%d\t%d",i,j,k);
    return 0;
}
```

i = 7  
j = 3  
k = 56



- int i = 7; // Decimal 7 is Binary  $(2^2 + (2^1) + (2^0)) = 0000\ 0111$
- int j = 3; // Decimal 3 is Binary  $(2^1 + (2^0)) = 0000\ 0011$
- k = (i << j); // Left shift operation multiplies the value by 2 to the power of j in decimal
  - // Equivalent to adding j zeros to the binary representation of i
    - //  $56 = 7 * 2^3$
    - //  $0011\ 1000 = 0000\ 0111 \ll 0000\ 0011$



***KLEF***

***(CTSD)***

***BES-1***

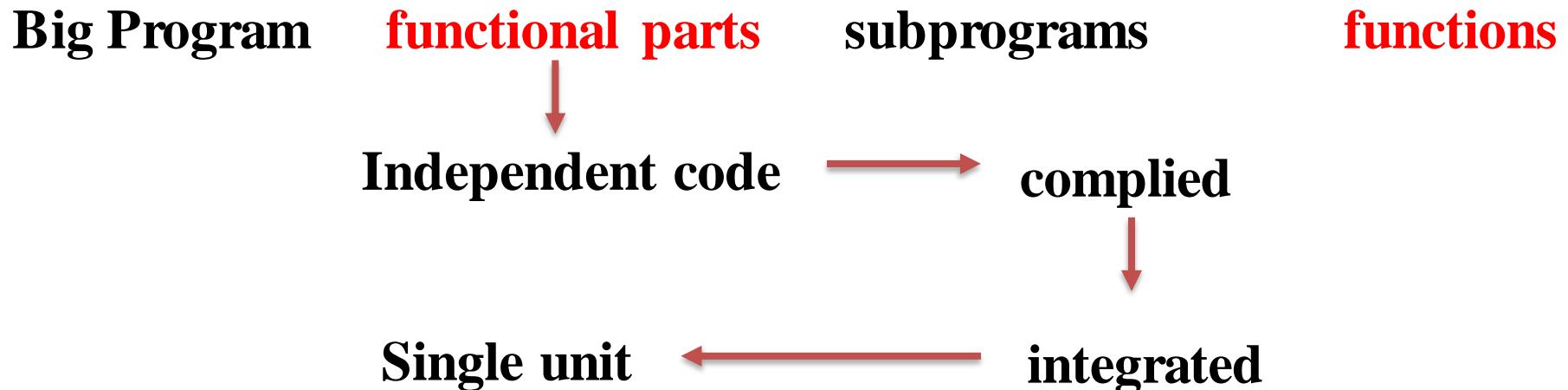


# CO3

- ❖ Recursion
- ❖ Character Arrays
- ❖ Strings & Library
- ❖ Command Line Arguments
- ❖ 1D Array
- ❖ Linear Search
- ❖ Binary Search
- ❖ DMA
- ❖ 2D Arrays
- ❖ Matrix Algebra



- ❖ Every C Program must have main function and any no.of other user defined functions.
- ❖ Program large complex
- ❖ Difficult understand debug test
- ❖ Difficult identify logicalerrors
- ❖ Repetitions of same set of statements a no of times within program.
- ❖ Difficult to update the program.





## Calling function

void **main()**

{

**functionname1();**  
    **printf("Its Monday");**

**Calling function**

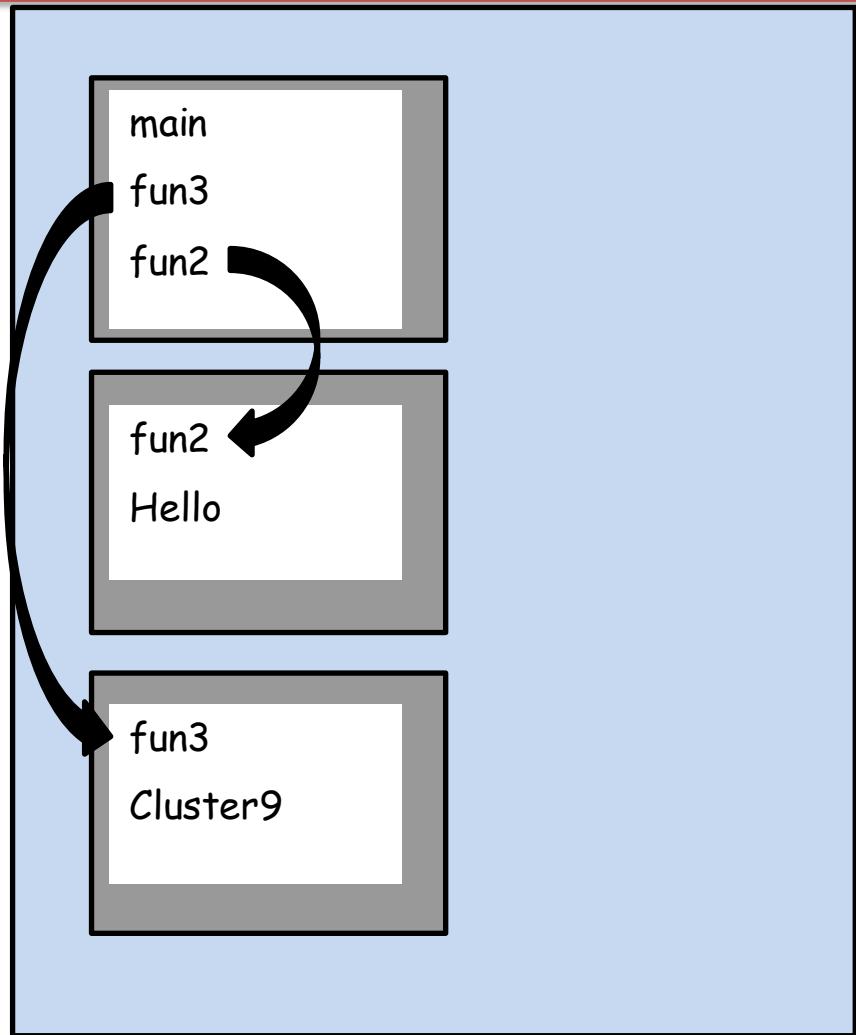
**functionname1();**      **Called function**

void **functionname1()**  
{

**printf("Hi Cluster9 Students");**

**Note:**

**A function can be called by any other function but main cannot be called by other function.**



## 1.) Without Arguments & Without ReturnValues

(No i/p & No o/p)  

## 2.) With Arguments &

Without ReturnVa

(i/p & No o/p)



## 3.) With Arguments &

With ReturnValues

(i/p & o/p)



## 4.) Without Arguments &

With ReturnValues

(No i/p & o/p)



# main()

```
{
    fun3();
}
void fun1()
{
    printf("Hi");
}
void fun2()
{
    fun1();
    printf("Cluster9");
}
void fun3()
{
    fun2();
    printf("listen");
}
```

# main()

fun3  
callsfun2()

fun2  
callsfun1()

✓ fun1()  
Hi



## Without Arguments &

### Without Return Value

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

```
void add();
```

```
void main()
```

```
{
```

```
    add();
```

```
}
```

```
void add()
```

```
{
```

```
    int a,b,s=0;
```

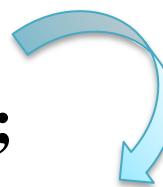
```
    a=10;
```

```
    b=20;
```

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

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

```
}
```



## With Arguments &

### Without Return Value

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

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

```
void main()
```

```
{
```

```
    int a,b;
```

```
    a=10;          b=20;
```

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

```
}
```

```
void add(int a1,int b1)
```

```
{
```

```
    int s;
```

```
    s=a1+b1;
```

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

```
}
```



## Without Arguments &

### With Return Value

```
#include<stdio.h>
int add();
void main()
{
    int result;
    result=add();
    printf("%d",result);
}

int add()
{
    int a,b,s=0;
    a=10;
    b=20;
    s=a+b;
    return s;
}
```

## With Arguments &

### With Return Value

```
#include<stdio.h>
int add(int a,int b);
void main()
{
    int s=0;
    int a=10,b=20;
    s=add(a,b);
    printf("%d",s);
}

int add(int a1,int b1)
{
    int r;
    r=a1+b1;
    return r;
}
```





## Call by Value and Call by Reference

On the basis of arguments there are two types of functions available in c Language.

### function

#### With Arguments

- ❖ Declare and defined with parameters list
- ❖ Values for parameters passed during function calls

**Ex:** int sum(int x,int y);  
      //declaration  
      sum(10,20);

**KLEF call**

#### Without Arguments

- ❖ No parameters List
- ❖ No value passed during function call

**Ex:** int show();  
      show();

**(CTSD)**

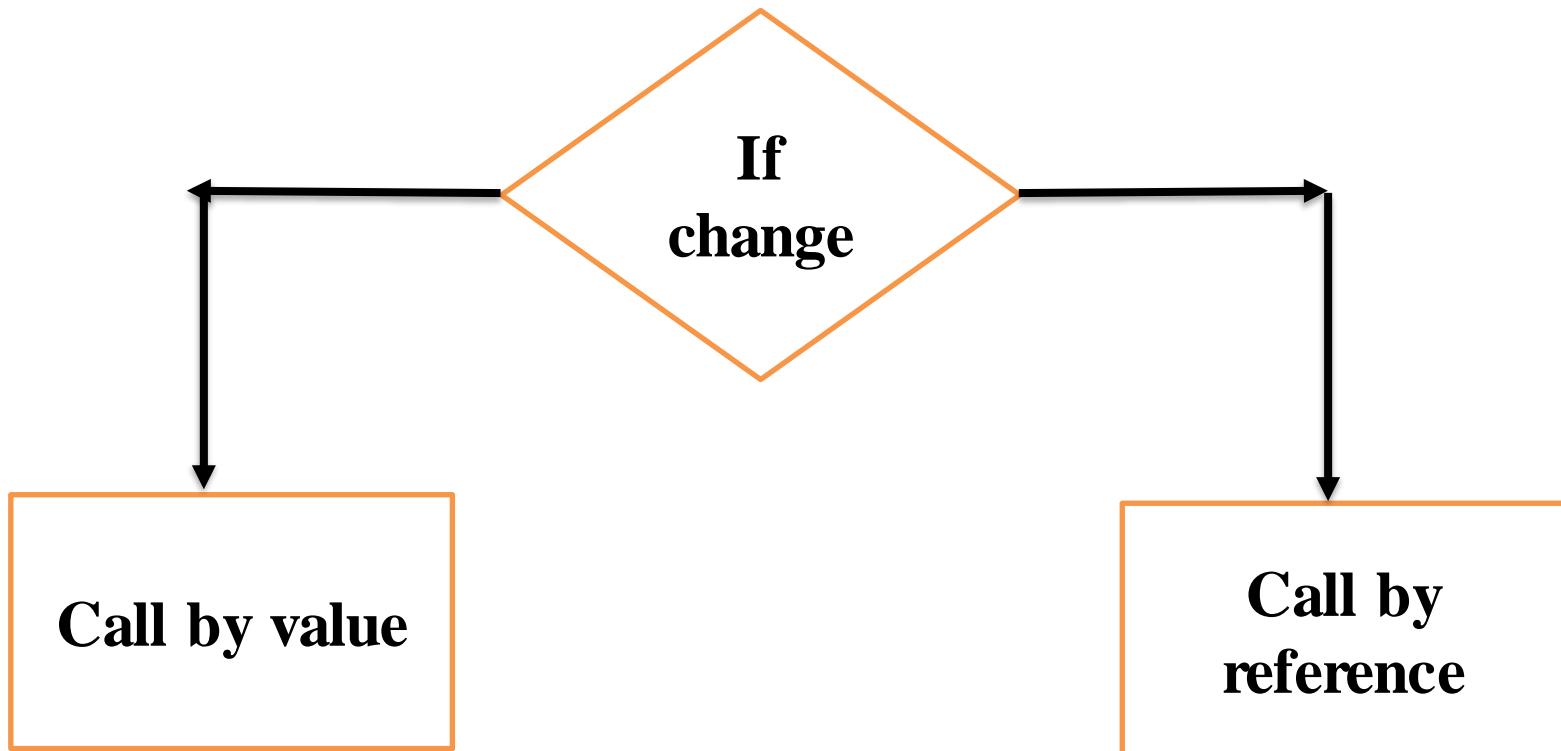
**BES-1**



2 ways to pass values or data to functions

call by value  
call by reference

Original value



## Call by value

Original value cannot be changed or modified.

```
void swap(int ,int);
```

```
void main()
{
    int a=10,b=20;
    swap(a,b);pass value to function
    printf("a=%d",a);
    printf("b=%d",b);
}
```

```
void swap(int a,int b)
{
    int temp;
    temp=a;    a=b;    b=temp;
    printf("a=%d",a);
    printf("b=%d",b);
}
```

## Call by reference

Original value is changed or modified.



```
void swap(int *,int *);
```

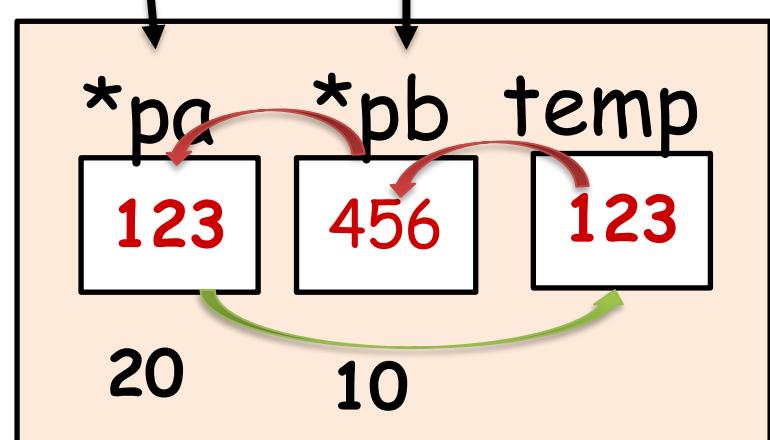
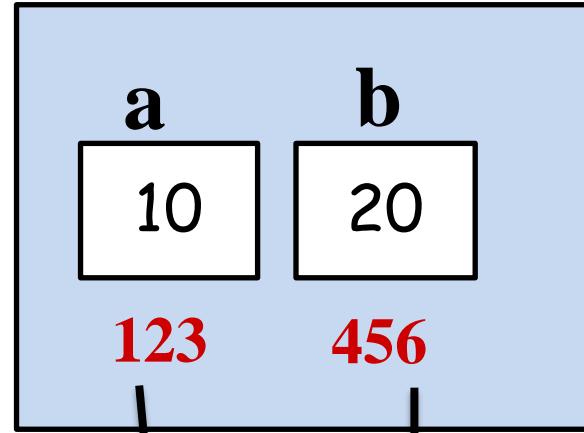
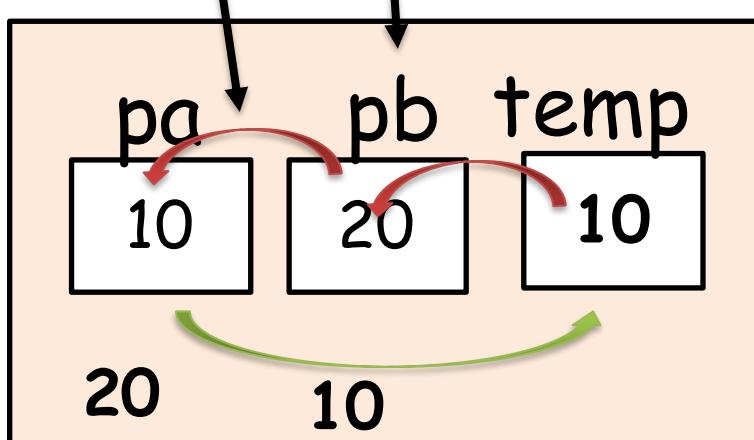
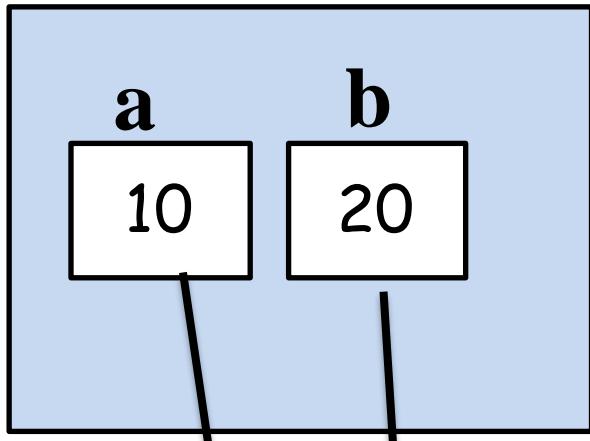
```
void main()
{
```

```
    int a=10,b=20;
    swap(&a,&b);pass address to function
    printf("a=%d",a);
    printf("b=%d",b);
}
```

```
void swap(int *pa,int *pb)
```

```
{
    int temp;    temp=*pa;
    *pa=*pb      *pb=temp;
    printf("a=%d",*pa);
    printf("b=%d",*pb);
}
```

main



$\text{temp} = *pa; \quad *pa = *pb; \quad *pb = \text{temp};$   
 $pa = \text{temp}; \quad pa = pb; \quad pb = \text{temp};$   
*(CTSD)*



# RECURSION



***KLEF***

***(CTSD)***

***BES-1***



# Recursive Functions

- Recursive function is closely related to mathematical induction
- Recursive function is a function that calls itself
- if a program allows you to call a function inside the same function, then it is called a *recursive call* of the function.
- Every Recursive function will have exit or stop condition – other wise the recursive function will keep on calling itself

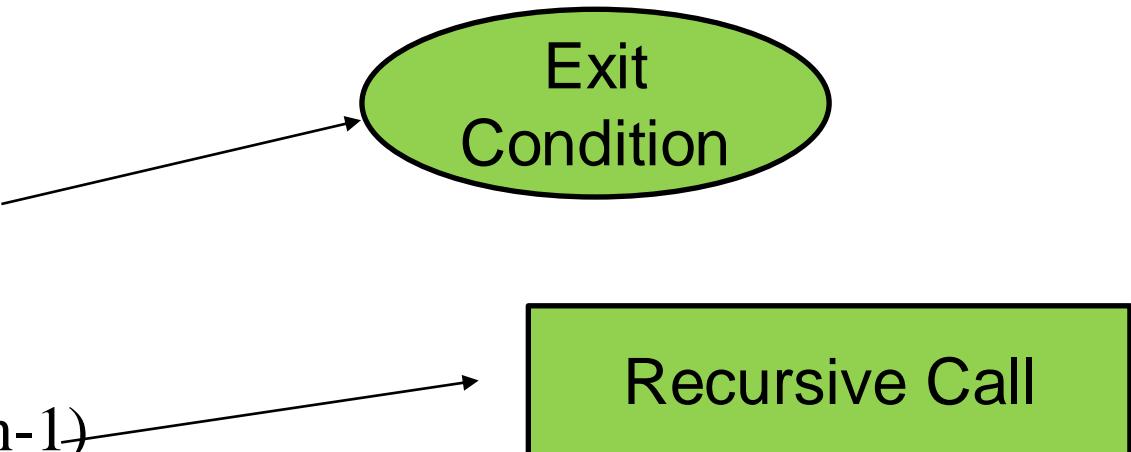


# Finding Factorial Recursively

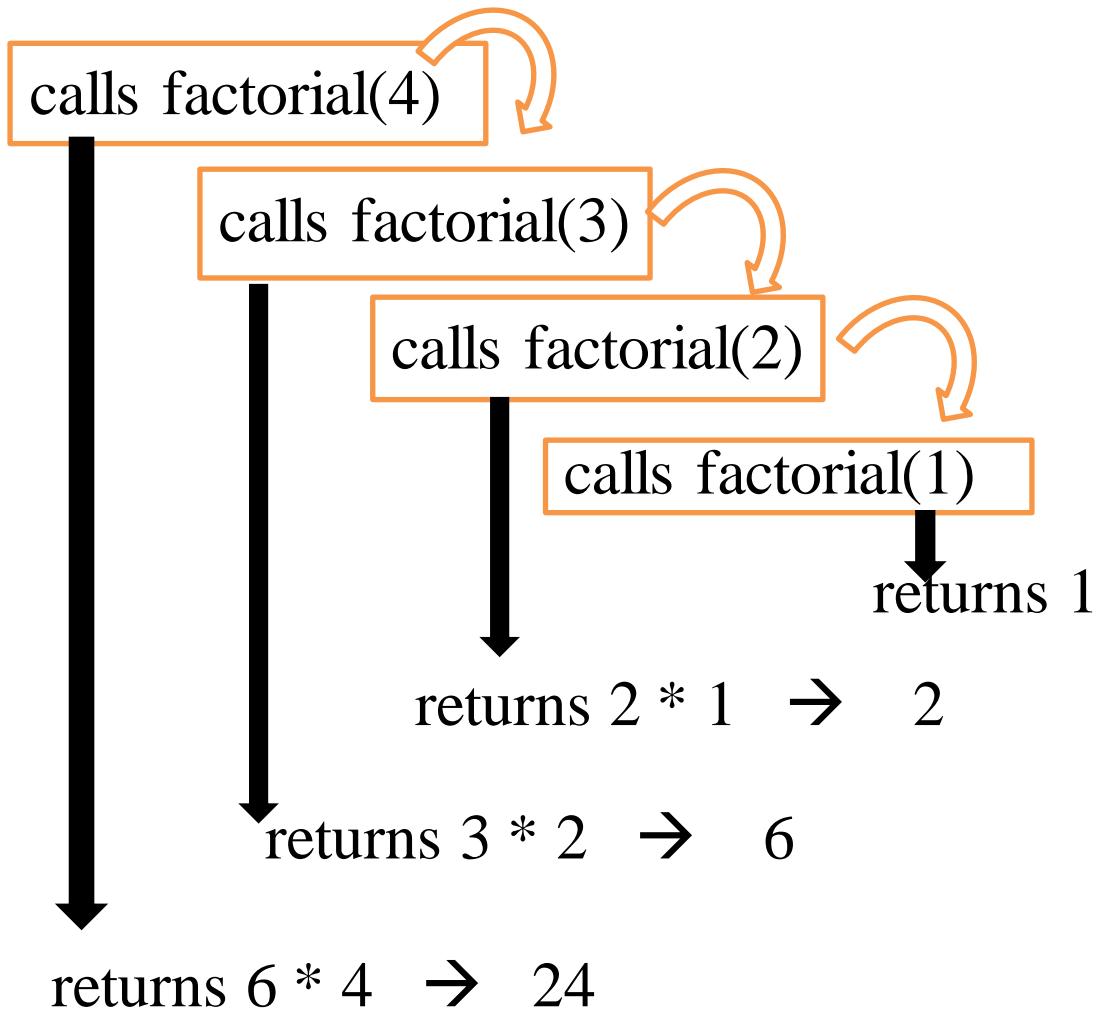
$$fact(n) = \begin{cases} 1 & \text{if } n == 0 \text{ or } n == 1 \\ n * fact(n - 1) & \text{if } n > 1 \end{cases}$$

**Recursive function :**

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



factorial(4)



factorial(4)

Step 0: Executes Factorial 4

Step 7:  
returns  $4 * 6 = 24$

Step 6:  
returns  $3 * 2 = 6$

Step 5:  
returns  $2 * 1 = 2$

Step 4:  
returns 1

return  $4 * \text{factorial}(3)$

Step 1: Executes Factorial 3

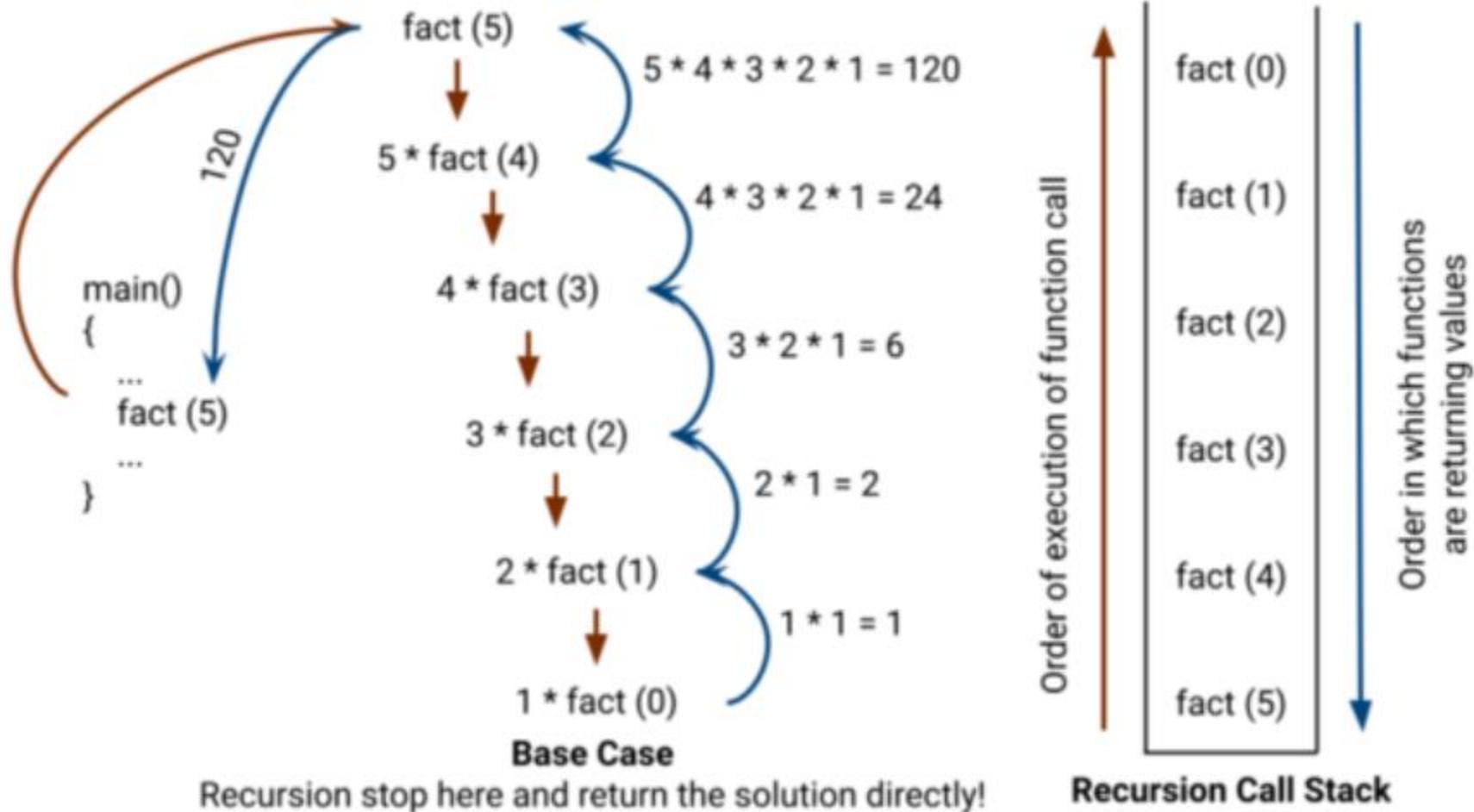
return  $3 * \text{factorial}(2)$

Step 2: Executes Factorial 2

return  $2 * \text{factorial}(1)$

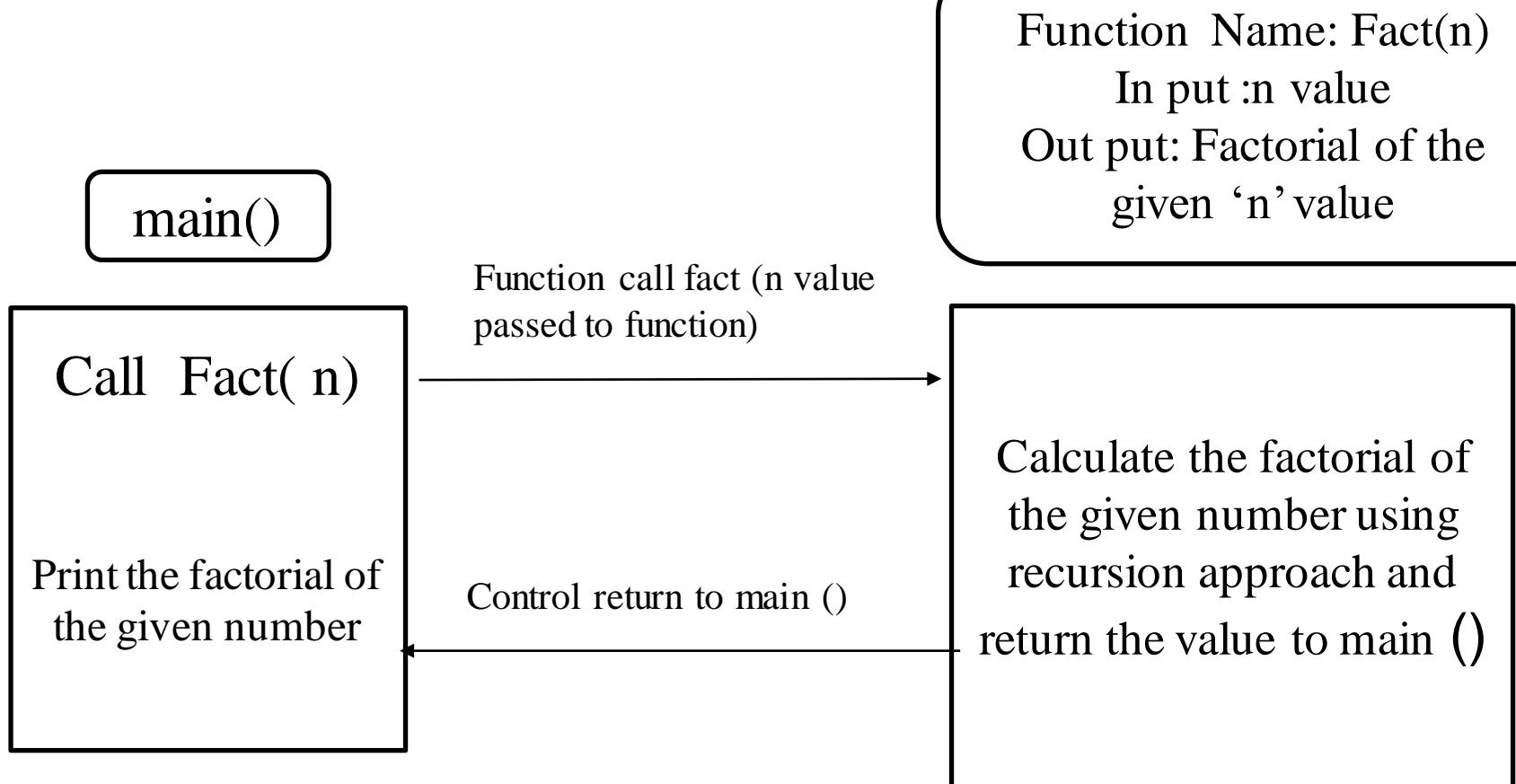
Step 3: Executes Factorial 1

return 1

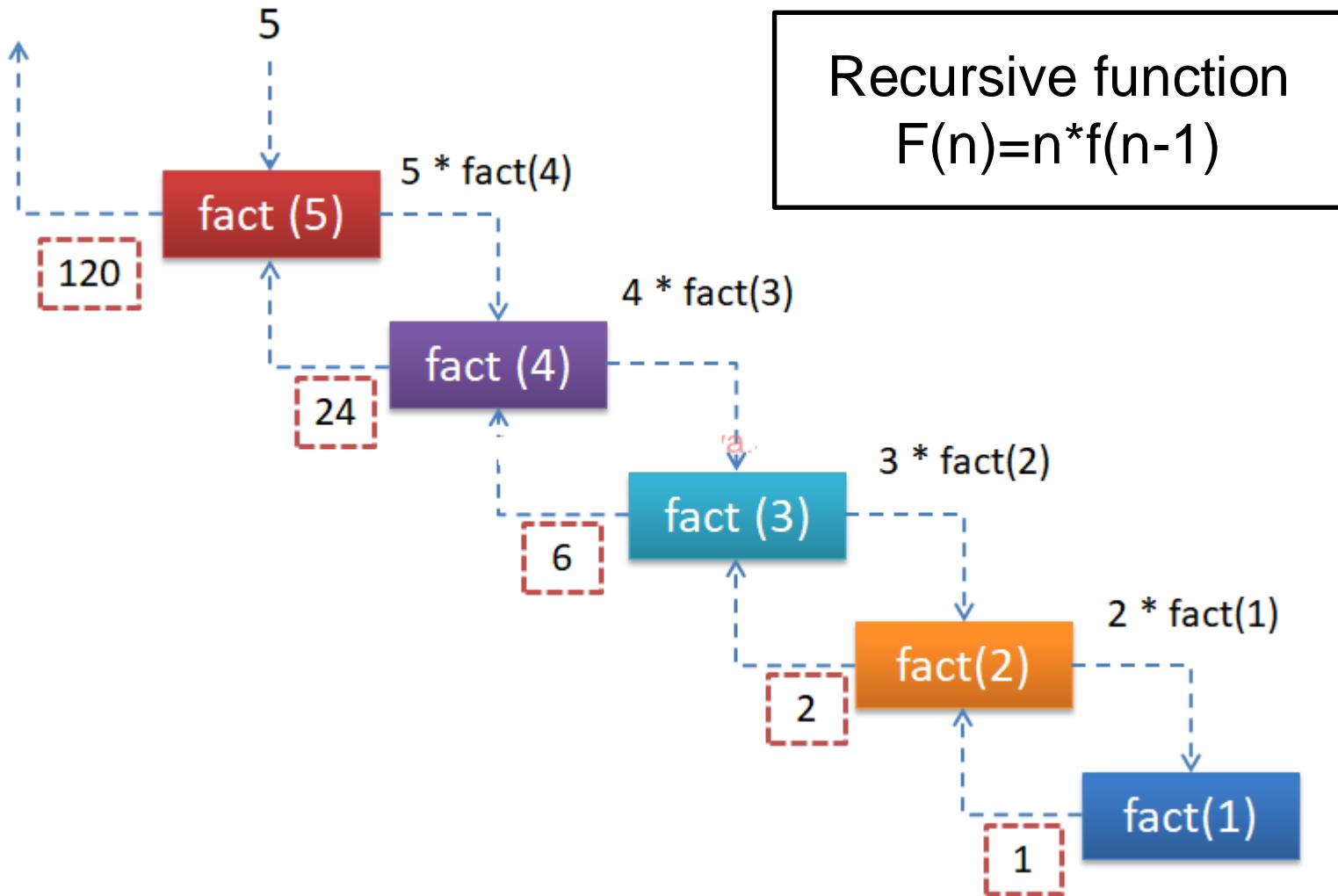




# Flow Of Execution



# Flow of Execution



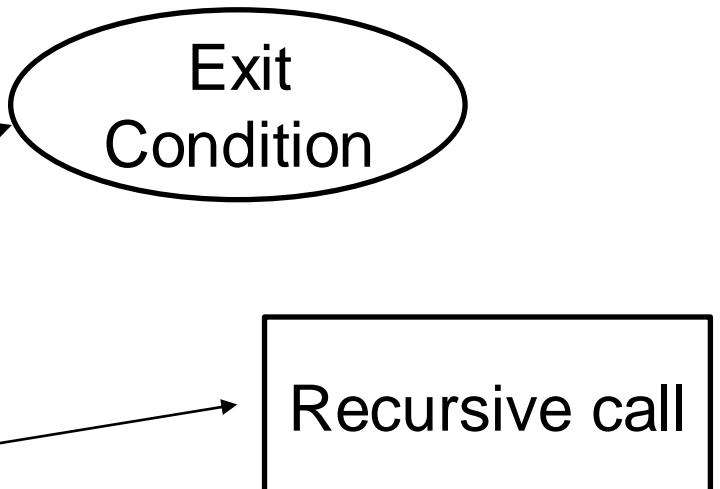


# Sum of Natural Numbers Recursively

$$sum(n) = \begin{cases} 1 & \text{if } n == 1 \\ n + sum(n - 1) & \text{if } n > 1 \end{cases}$$

Corresponding Recursive Function

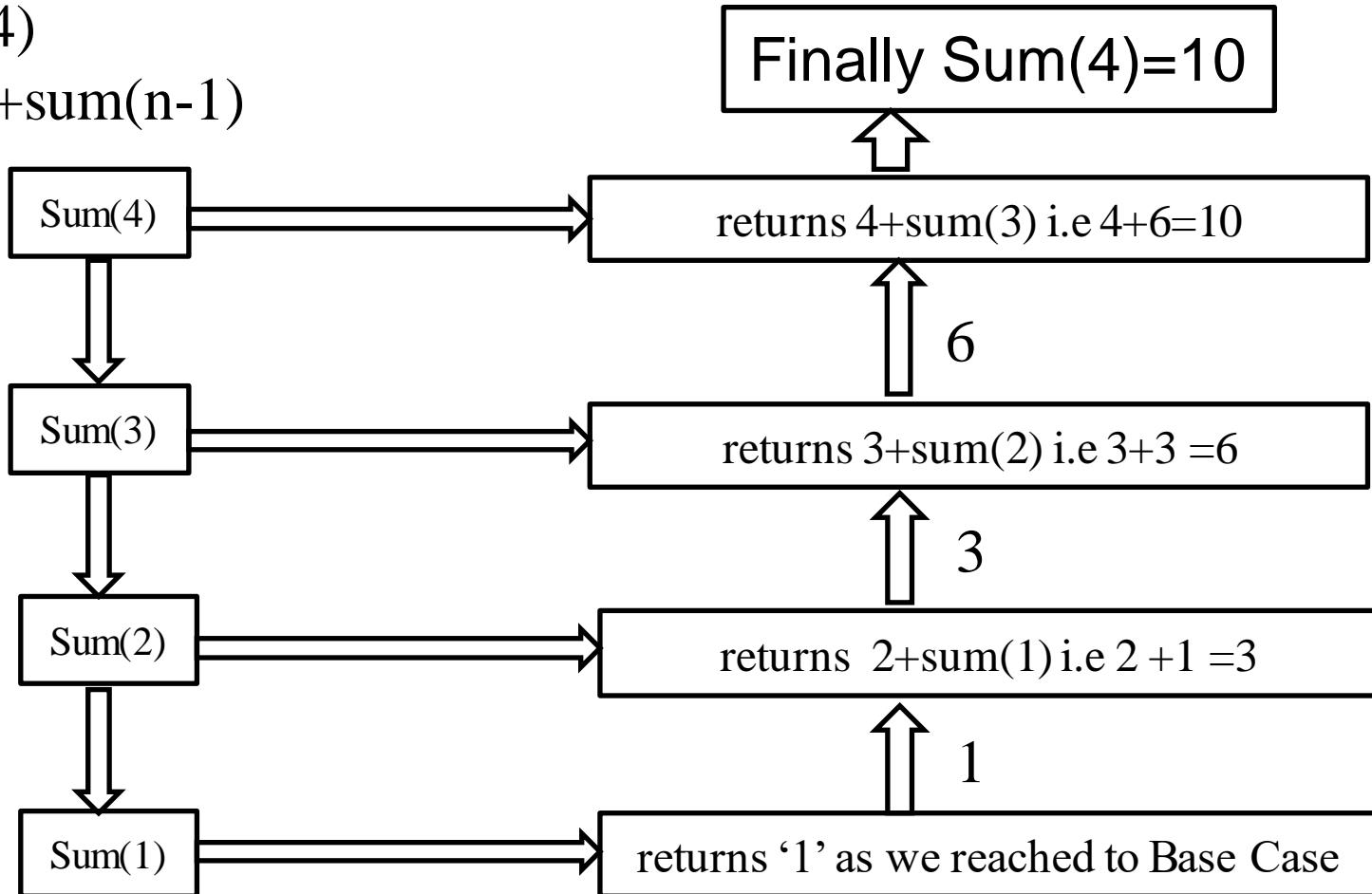
```
int sum(int n)
{
    if(n == 1)
        return 1;
    else
        return n + sum(n-1);
}
```



# Sum of n numbers recursively

Eg sum(4)

return  $n + \text{sum}(n-1)$





# Finding nth term in Fibonacci Series Recursively

- The Fibonacci numbers are the numbers in the following integer sequence.  
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, .....
- In mathematical terms, the sequence  $F_n$  of Fibonacci numbers is defined by the recurrence relation

$$F_n = F_{n-1} + F_{n-2}$$



# Finding nth term in Fibonacci Series Recursively

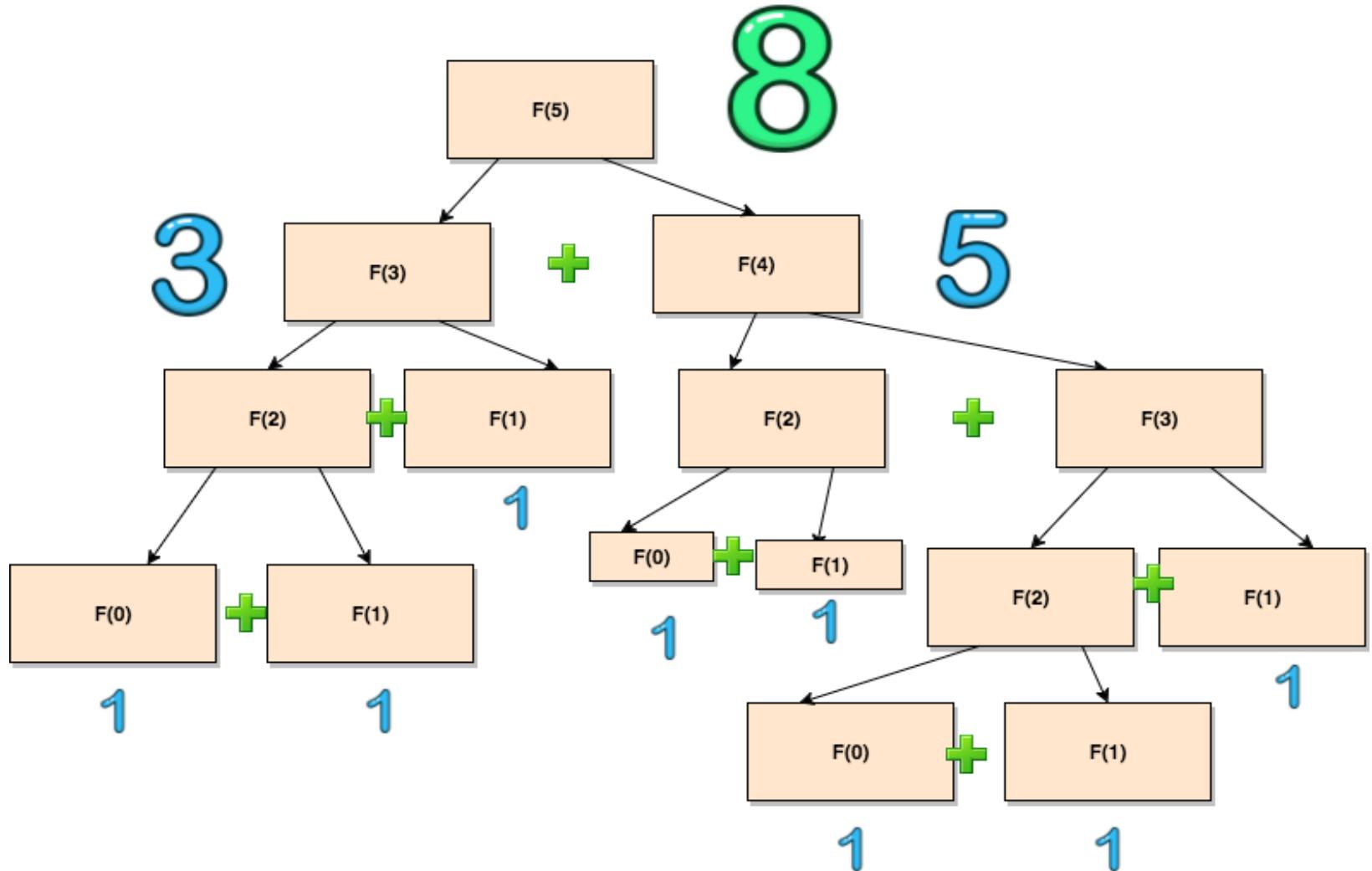
$$fib(n) = \begin{cases} 0 & \text{if } n == 0 \\ 1 & \text{if } n == 1 \\ fib(n - 1) + fib(n - 2) & \text{if } n > 1 \end{cases}$$

conditions of recursive function to find  $n^{\text{th}}$  Fibonacci term :

- If num == 0 then return 0. Since Fibonacci of 0<sup>th</sup> term is 0.
- If num == 1 then return 1. Since Fibonacci of 1<sup>st</sup> term is 1.
- If num > 1 then return fibo(num - 1) + fibo(n-2).

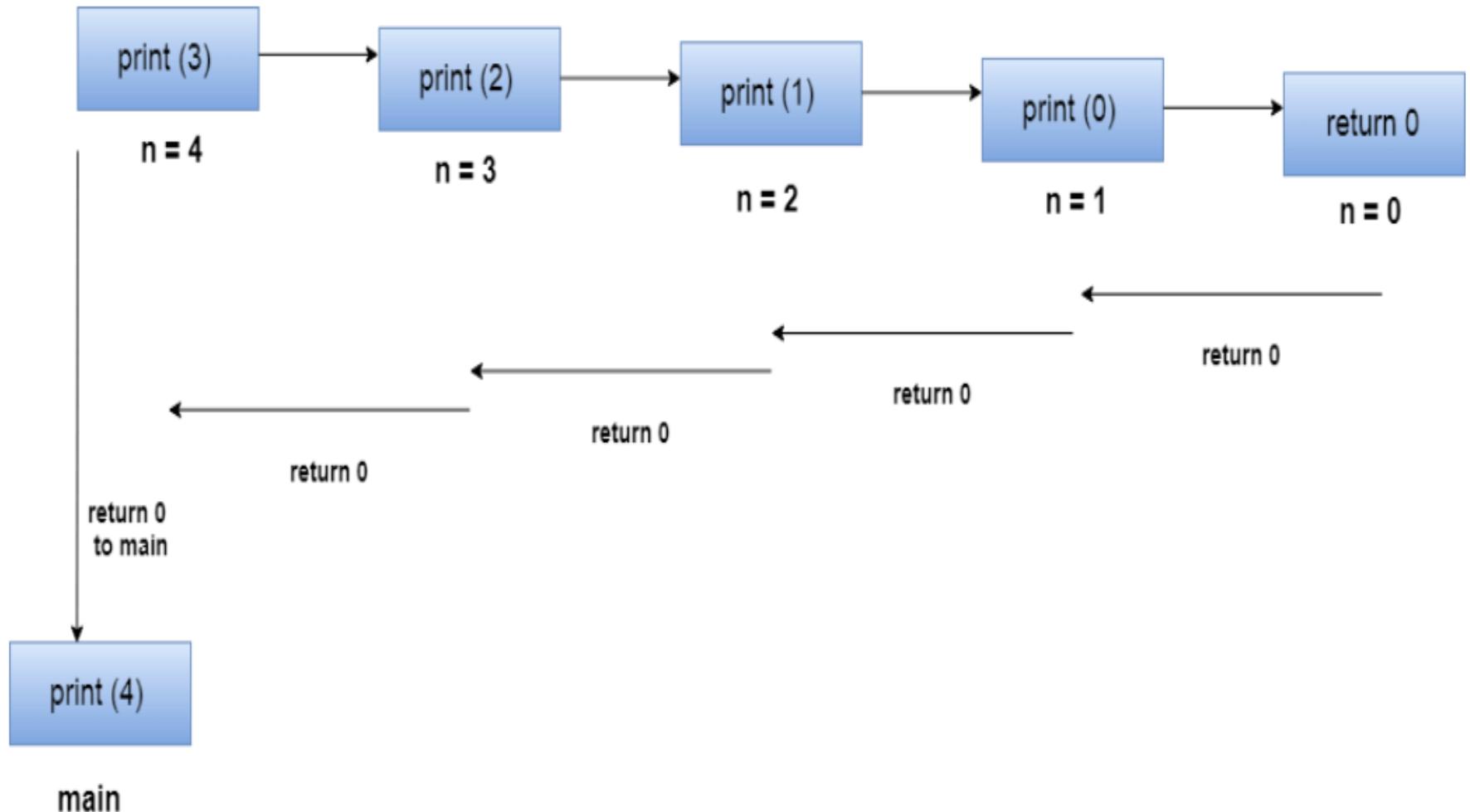
Since Fibonacci of a term is sum of previous two terms.

# 5th Fibonacci number :





```
int display (int n)
{
    if(n == 0)
        return 0; // terminating condition
    else
    {
        printf("%d",n);
        return display(n-1); // recursive call
    }
}
```





## Corresponding Recursive Function

```
int fib(int n)
{
    if( n ==0)
        return 0;
    if( n ==1)
        return 1;
    else
        return fib(n-1)*fib(n-2);
}
```

For example,  $n = 2$

$(\text{Fib}(n-2) + \text{Fib}(n - 1))$

$(\text{Fib}(2 - 2) + \text{Fib}(2 - 1))$

It means,  $(\text{Fib}(0) + \text{Fib}(1))$

$\text{return } (0 + 1)$

finally it returns 1



```
int fibonacci(int i)
{
    if(i == 0)
        { return 0; }
    if(i == 1)
        { return 1; }
    return fibonacci(i-1) +
        fibonacci(i-2);
}
```

```
int main()
{
    int i;
    for (i = 0; i < 10; i++) {
        printf("%d\t\n", fibonacci(i));
    }
    return 0;
}
```

```

int main() {
    ...
    result = sum(number);
    ...
}

int sum(int n) {
    if (n != 0)
        return n + sum(n-1);
    else
        return n;
}

int sum(int n) {
    if (n != 0)
        return n + sum(n-1);
    else
        return n;
}

int sum(int n) {
    if (n != 0)
        return n + sum(n-1);
    else
        return n;
}

int sum(int n) {
    if (n != 0)
        return n + sum(n-1);
    else
        return n;
}

```

3  
3  
2  
1  
0

$3+3 = 6$   
is returned

$2+1 = 3$   
is returned

$1+0 = 1$   
is returned

0  
is returned



```
#include<stdio.h>
long factorial(int n)
{
    if (n == 1)
        return 1;
    else
        return(n * factorial(n-1));
}
void main()
{
    int number;
    long fact;
    printf("Enter a number: ");
    scanf("%d", &number);
    fact = factorial(number);
    printf("Factorial of %d is %ld\n"
           , number, fact);
    return 0;
}
```



```
//Recursive Implementation
int fact(int n)
{
    if (n == 0)
        return 1

    return n * fact(n - 1)
}
```

```
//Iterative Implementation
int fact(int n)
{
    int output = 1
    for (int i = 2; i <= n; i = i + 1)
        output = output*i

    return output
}
```



# Character

```
#include <stdio.h>

int main() {
    char c;
    printf("Enter a character:");
    scanf("%c", &c);
    // %d displays the integer value of a character
    // %c displays the actual character
    printf("ASCII value of %c = %d", c, c);
    return 0;
}
```

C

a

a=97



# String

```
#include<stdio.h>
int main()
{
    char c[10];
    scanf("%s",c);
    printf("%s",c);
    return 0;
}
```

c	l	u	s	t	e	r	\0		
---	---	---	---	---	---	---	----	--	--



```
#include<stdio.h>
int main()
{
    char str[]={'k','l','u','\0'};
printf("%s",s);
return 0;
}
```

```
#include<stdio.h>
int main()
{
    char str[5]={'k','l','u','\0'};
printf("%s",s);
return 0;
}
```



```
#include<stdio.h>
int main()
{
    char str[5]={"KLU"};
    printf("%s",s);
    return 0;
}
```

```
#include<stdio.h>
int main()
{
    char str[]={"KLU"};
    printf("%s",s);
    return 0;
}
```



# strlen

```
#include<stdio.h>
int main()
{
    char str[5]={'k','l','u','\0'};
printf("%d",strlen(str));
return 0;
}
```

```
#include<stdio.h>
int main()
{
    char
str[5]={'k','l','u','\0'};
    int i;
    for(i=0;str[i]!='\0';i++);
        printf("%d",i);
    return 0;
}
```

```
#include<stdio.h>
int main()
{
    char str[5]={'k','l','u','\0'};
    int i,c=0;
    for(i=0;str[i]!='\0';i++)
        c++;
    printf("%d",c);
    return 0;
}
```

# strcpy



```
#include<stdio.h>
int main()
{
    char s1[20];           char s2[20];
    int i;                 gets(s1);           gets(s2);
    //scanf("%s%s",s1,s2);
    for(i=0;s1[i]!='\0';i++)
    {
        s2[i]=s1[i];
    }
    s2[i]='\0';
    printf("s1=%s\ns2=%s",s1,s2);
    return 0;
}
```



## strcpy

```
#include<stdio.h>      #include<string.h>
int main()
{
    char s1[10];          char s2[10];
    //scanf("%s%s",s1,s2);
    gets(s1);            gets(s2);
    strcpy(s2,s1);
    printf("s1=%s \ns2=%s",s1,s2);
    return 0;
}
```



int main()    strcat

```
{  
char s1[20];        char s2[20]; int i;  
gets(s1);    gets(s2);  
for(i=0;s1[i]!='\0';i++);  
for(j=0;s2[j]!='\0';j++,i++)  
{  
    s1[i]=s2[j];  
}  
s1[i]='\0';  
printf("s1=%s\ns2=%s",s1,s2);  
return 0;
```



```
#include<stdio.h>
#include<string.h>
int main()
{
    char s1[20];
    char s2[20];
    //scanf("%s%s",s1,s2);
    gets(s1);
    gets(s2);
    strcat(s1,s2);
    printf("s1=%s\ns2=%s",s1,s2);
    return 0;
}
```



## String Pointers

```
#include <stdio.h>

int main()
{
    char name[] = "Harry Potter";
    char *namePtr;
    namePtr = name;
    printf("%c", *namePtr);    // Output: H
    printf("%c", *(namePtr+1)); // Output: a
    printf("%c", *(namePtr+7)); // Output: o
}
```



No.	Function	Description
1)	<u>strlen(string_name)</u>	returns the length of string name.
2)	<u>strcpy(destination, source)</u>	copies the contents of source string to destination string.
3)	<u>strcat(first_string, second_string)</u>	concats or joins first string with second string. The result of the string is stored in first string.
4)	<u>strcmp(first_string, second_string)</u>	compares the first string with second string. If both strings are same, it returns 0.
5)	<u>strrev(string)</u>	returns reverse string.
6)	<u>strlwr(string)</u>	returns string characters in lowercase.
7)	<u>strupr(string)</u>	returns string characters in uppercase.



```
int main()
{
    char str[10];int i;
    gets(str);
    for(i=0;str[i]!='\0';i++)
    {
        if(str[i]>='a'&&str[i]<='z')
            str[i]=str[i]-32;
    }
    printf("Your String:%s",str);
    return 0;
}
```

## Lower Case to Upper Case



```
int main()
{
    char str[10];int i;
    gets(str);
    char *s=str;
    while(*s)
    {
        if((*s>='a')&& (*s<='z'))
            *s=*s-32;
        s++;
    }
    printf("Your String:%s",str);
    return 0;
}
```



```
#include<stdio.h>
int main()
{
    char str[10];int i;
    gets(str);
    char *s=str;
    while(*s)
    {
        *s=(*s>='a'&&*s<='z')?*s-32:*s;
        s++;
    }
    printf("String:%s",str);
    return 0;
```



```
#include<stdio.h>
int main()
{
    char str[10];int i;
    gets(str);
    char *s=str;
    while(*s)
    {
        *s=(*s>='a'&&*s<='z')?*s-32:*s;
        s++;
    }
    printf("String:%s",str);
    return 0;
```



```
#include<stdio.h>
int main()
{
    char str[50];int i=0,word=1;
    gets(str);
    while(str[i]!='\0')
    {
        if(str[i]==' '||str[i]=='\n'||str[i]=='\t')
            word++;
        i++;
    }
    printf("words:%d",word);
return 0;
}
```



```
#include<stdio.h>
#include<string.h>
int main()
{
char s1[15]="KL University";
char s2[15]="KL UNIVERSITY";
if(strcmp(s1,s2)==0)
    printf("equal");
else
printf("Not Equal");
return 0;
}
```



```
#include<stdio.h>
#include<string.h>
int main()
{
    char s1[15] = "Hello";
    char s2[15] = "cluster9";
    strcat(s1,s2);
    printf("%s",s1);
    printf("\n%s",s2);
    return 0;
}
```



```
#include<stdio.h>
#include<string.h>
int main()
{
    char s1[15] = "Hello";
    char s2[15] = "Students";
    strcpy(s1,s2);
    printf("%s\t%s",s1,s2);
    return 0;
}
```

S1=Students  
S2= Students

```
#include<stdio.h>
#include<string.h>
int main()
{
    char s1[15];
    char s2[15];
    strcpy(s1,"Hello");
    strcpy(s2,"Students");
    printf("%s\t%s",s1,s2);
    return 0;
}
```

S1>Hello  
S2= Students



```
#include<stdio.h>
#include<string.h>
int main()
{
    char s1[15] = "Hello";
    char s2[15] = "Students";
    for(i=0;s1[i]!='\0';i++)
    {
        s2[i]=s1[i];
    }
    s2[i]='\0';
    printf("%s",s2); printf("%s",s1);
    return 0;
}
```



```
#include<stdio.h>
#include<string.h>
int main()
{
    char s1[15]="Hello";
    char s2[15]="cluster9";
    strcpy(s1,s2);
    printf("%s\t%s",s1,s2);
    return 0;
}
```

```
#include<stdio.h>
#include<string.h>
int main()
{
    char s1[15];
    char s2[15];
    strcpy(s1,"Hello");
    strcpy(s2,"Cluster9");
    printf("%s\t%s",s1,s2);
    return 0;
}
```



```
char language[5][10] =  
{"Java", "Python", "C++", "HTML", "SQL"};
```

```
char largestcity[6][15] =  
{"Tokyo", "Delhi", "Shanghai", "Mumbai",  
"Beijing", "Dhaka"};
```



```
char language[5][10] =  
{  
    {'J','a','v','a','\0'},  
    {'P','y','t','h','o','n','\0'},  
    {'C','+','+','\0'},  
    {'H','T','M','L','\0'},  
    {'S','Q','L','\0'}  
};
```



J	a	v	a	\o					
P	y	t	h	o	n	\o			
C	+	+	\o						
H	T	M	L	\o					
s	Q	L	\o						



- Note1:- the number of characters (column-size) must be declared at the time of the initialization of the two-dimensional array of strings.
- **// it is valid**
- `char language[ ][10] = {"Java", "Python", "C++", "HTML", "SQL"};`



But that the following declarations  
are invalid.

// invalid

```
char language[ ][ ] = {"Java",
"Python", "C++", "HTML", "SQL"};
```

// invalid

```
char language[5][ ] = {"Java",
"Python", "C++", "HTML", "SQL"};
```



Note2:- Once we initialize the array of String then we can't directly assign a new String.

```
char language[5][10] = {"Java", "Python", "C++",
"HTML", "SQL"};
```

// now, we can't directly assign a new String

```
language[0] = "Kotlin"; // invalid
```

// we must copy the String

```
strcpy(language[0], "Kotlin"); // valid
```

// Or,

```
scanf(language[0], "Kotlin"); // valid
```



# Structures

User defined datatypes combine data of different types together

Data              Stored

Records

Define a Structure

```
struct structure-tag  
{  
    member variable1;  
    member variable2;  
};
```

Example:

```
struct student  
{  
    char name[25];  
    int age;  
    char branch[5];  
    char status;  
};
```



## Assign Values to Structure Members.

### 1.) Using Dot(.) Operator

**variablename.membername=value;**

### 2.) All members assigned in one statement

**struct structname variablename={ value of member1,  
value of member2,-----};**

```
struct student  
{
```

```
    char name[25];  
    int age;  
    char branch[10];  
    char status;
```

```
};
```

```
struct student s1,s2;
```

```
struct student  
{  
    char name[25];  
    int age;  
    char branch[10];  
    char status;  
}s1,s2;
```



```
#include<stdio.h>
#include<string.h>
struct student
{
    char name[25];
    int age;
    char branch[5];
    char status;
};
```

```
int main()
{
    struct student s1;
    strcpy(s1.name,"Rahul");
    s1.age=17;
    strcpy(s1.branch,"CSE");
    s1.status='h';

    printf("Name=%s",s1.name);
    printf("Age=%d",s1.age);
    printf("Branch=%s",s1.branch);
    printf("Status=%c",s1.status);
    return 0;
}
```



```
#include<stdio.h>
#include<string.h>
struct student
{
    char name[25];
    int age;
    char branch[5];
    char status;
};
```

```
int main()
{
    struct student s1;
    strcpy(s1.name,"Rahul");
    s1.age=17;
    strcpy(s1.branch,"CSE");
    s1.status='h';

    printf("Name=%s",s1.name);
    printf("Age=%d",s1.age);
    printf("Branch=%s",s1.branch);
    printf("Status=%c",s1.gender);
    return 0;
}
```



## Structure Pointers

```
#include <stdio.h>
struct person
{
    int age;
    float weight;
};
```

```
int main()
{
    struct person *personPtr, person1;
    personPtr = &person1;
    printf("Enter age: ");
    scanf("%d", &personPtr->age);
    printf("Enter weight: ");
    scanf("%f", &personPtr->weight);
    printf("Displaying:\n");
    printf("Age: %d\n", personPtr->age);
    printf("weight: %f", personPtr->weight);
    return 0;
}
```



**struct Books**

{

char title[50];

char author[50];

char subject[100];

int book\_id;

} book;

**struct Velocity**

{

char name[50];

int distance;

**struct timetaken**

{

int hours;

int mins;

int secs;

} tt;

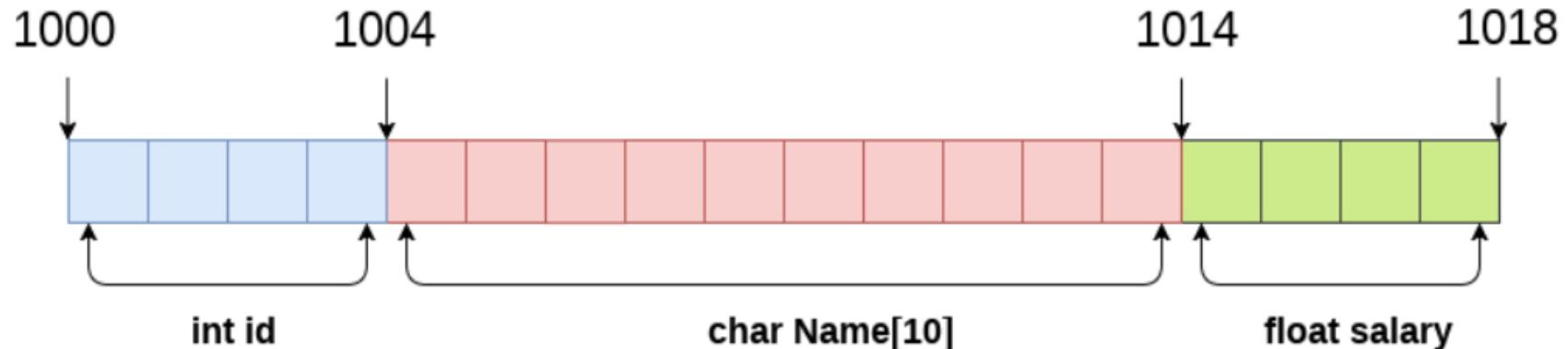
}vel;



```
// struct with typedef person  
typedef struct Person {  
    // code  
} person;
```

```
// equivalent to struct Person p1  
person p1;
```

```
struct employee
{ int id;
char name[20];
float salary;
};
```



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

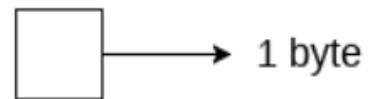
**sizeof (emp) = 4 + 10 + 4 = 18 bytes**

**where;**

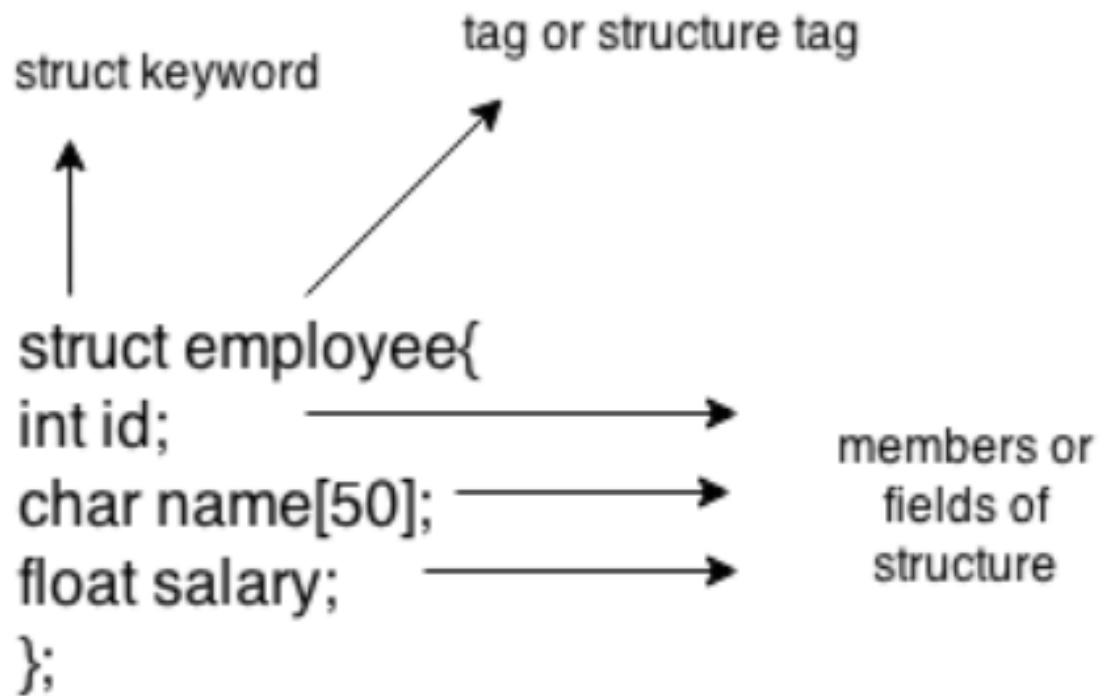
**sizeof (int) = 4 byte**

**sizeof (char) = 1 byte**

**sizeof (float) = 4 byte**



**struct** is the keyword; **employee** is the name of the structure; **id**, **name**, and **salary** are the members or fields of the structure.





// Program to calculate the sum of array elements by passing to a function

```
#include <stdio.h>

float calculateSum(float age[]);

int main() {
    float result, age[] = {23.4, 55, 22.6, 3, 40.5, 18};

    // age array is passed to calculateSum()
    result = calculateSum(age);
    printf("Result = %.2f", result);
    return 0;
}
```

```
float calculateSum(float age[]) {
    float sum = 0.0;

    for (int i = 0; i < 6; ++i) {
        sum += age[i];
    }

    return sum;
}
```



**typedef <existing\_name> <alias\_name>**

**typedef int unit;**

**unit a, b;**

**struct student**

**{**

**char name[20];**

**int age;**

**};**

**typedef struct student stud;**

**stud s1, s2;**



```
struct employee  
{ int id;  
    char name[50]; struct employee e1, e2;  
    float salary;  
};
```

```
struct employee  
{ int id;  
    char name[50];  
    float salary;  
}e1,e2;
```

Accessing members of the structure

There are two ways to access structure members:

1. By . (member or dot operator)
2. By -> (structure pointer operator)



```
#include <stdio.h>

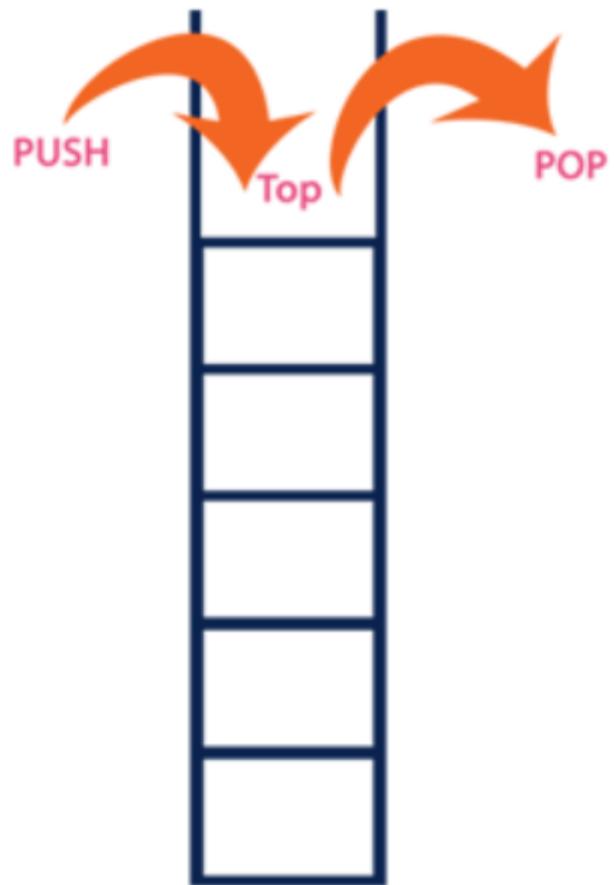
int main()
{
    char name[20] = "Harry Potter";
    printf("%c", *name); // Output: H
    printf("%c", *(name+1)); // Output: a
    printf("%c", *(name+7)); // Output: o

    char *namePtr;
    namePtr = name;
    printf("%c", *namePtr); // Output: H
    printf("%c", *(namePtr+1)); // Output: a
    printf("%c", *(namePtr+7)); // Output: o
}
```



## Stack ADT

- ❖ Stack is a linear data structure in which the insertion and deletion operations are performed at only one end.
- ❖ In a stack, adding and removing of elements are performed at a single position which is known as "top".
- ❖ That means, a new element is added at top of the stack and an element is removed from the top of the stack.
- ❖ In stack, the insertion and deletion operations are performed based on LIFO (Last In First Out) principle.





## Stack Operations

- **Push:** Adding a new item to the Stack Data Structure, in other words pushing new item to Stack DS.

If Stack is full, then it is said to be in an **overflow condition**

- **Pop:** Removing an item from the stack, i.e. popping an item out.

If a stack is empty then it is said to be in an **underflow condition**

- **Peek:** This basically returns the topmost item in the stack, in other words, peek that what is the topmost item.

- **IsEmpty:** This returns True If the stack is empty else returns False



## Representation of stack.

Stack as a data structure can be represented in two ways.

- Stack as an Array.
- Stack as a struct
- Stack as a Linked List.



```
void push(int stk[],int val)
{
    if (top == (max-1))
        printf("\nStack is full / Over flow");
    else
    {
        top++;
        stk[top]=val;
        printf("\nThe number is inserted");
    }
}
```



```
int pop(int stk[])
{
    int val;
    if (top== -1)
    {
        printf("\nThe stack is empty / Under flow");
        return (-1);
    }
    else
    {
        val=stk[top];
        top--;
        return(val);
    }
}
```



```
void display(int stk[])
{
    int i;
    if(top == -1)
        printf("\nThe stack is empty");
    else
    {
        for(i=top; i>=0; i--)
            printf("\n%d",stk[i]);
    }
}
```



An expression is a collection of operators and operands that represents a specific value.

## Expression Types

Based on the operator position, expressions are divided into THREE types. They are as follows...

1. Infix Expression
2. Postfix Expression
3. Prefix Expression

**INFIX**

Operand1      Operator      Operand2

(a+b)

Operator      Operand1      Operand2

+ab

**PREFIX**

**POSTFIX**

Operand1      Operand2      Operator

ab+



( A + B ) \* ( C - D )

A B + C D - \*

A-B\*D+E

ABD\*-E+

K + L - M\*N + (O^P) \* W/U/V \* T + Q

KL+MN\*-OP^W\*U/V/T\*+Q+

A+(B\*(C-D)/E)

ABCD-\*E/+

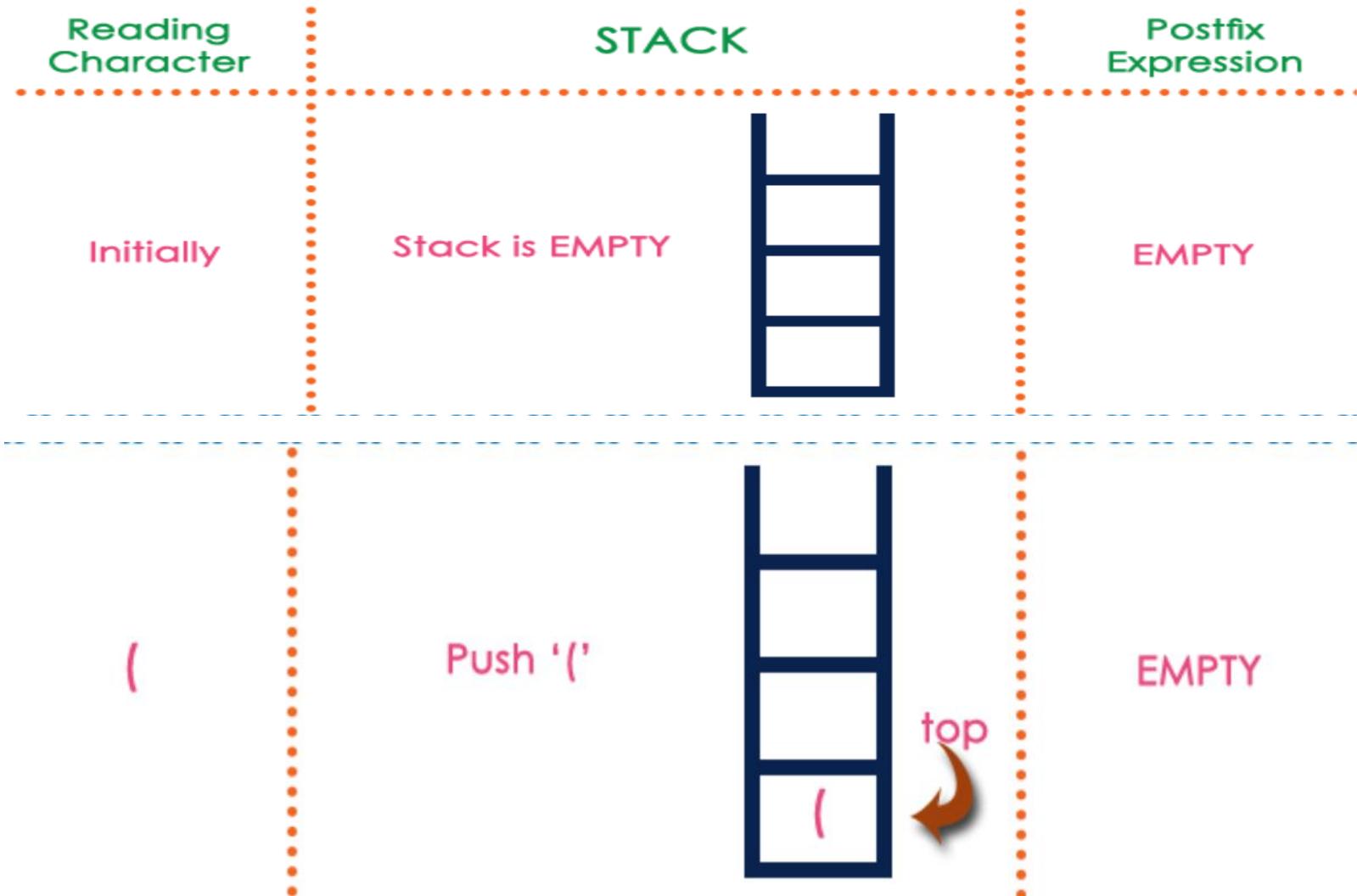


A+ (B\*C-(D/E^F)\*G)\*H

ABC\*D E F^/ G\*-H\*+

$$(A + B)^* (C - D)$$

The given infix expression can be converted into postfix expression using Stack data Structure as follows...



A

No operation  
Since 'A' is OPERAND



A

+

'+' has low priority  
than '(' so,  
PUSH '+'



A

B

No operation  
Since 'B' is OPERAND



A B

)

POP all elements till  
we reach '('

POP '+'  
POP '('



A B +

\*

Stack is EMPTY  
&  
'\*' is Operator  
PUSH '\*'



A B +

()

PUSH '('



A B +

C

No operation  
Since 'C' is OPERAND



A B + C

-

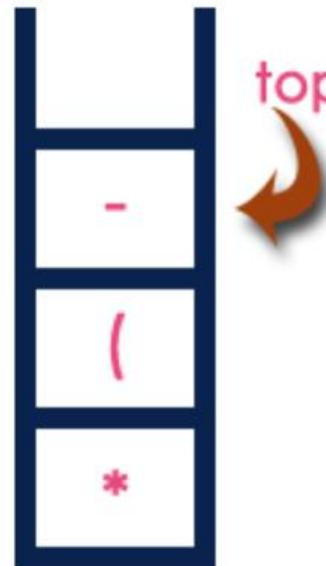
'-' has low priority  
than '(' so,  
PUSH '-'



A B + C

D

No operation  
Since 'D' is OPERAND



A B + C D

)

POP all elements till  
we reach '('

POP '-'  
POP '('



A B + C D -



\$

POP all elements till  
Stack becomes Empty

A B + C D - \*

The final Postfix  
Expression is as follows...

A B + C D - \*

- **Postfix Expression Evaluation**
- A postfix expression is a collection of operators and operands in which the operator is placed after the operands. That means, in a postfix expression the operator follows the operands.

Postfix Expression has following general structure...



Infix Expression

**(5 + 3) \* (8 - 2)**

Postfix Expression

**5 3 + 8 2 - \***

Above Postfix Expression can be evaluated by using Stack Data Structure as follows...

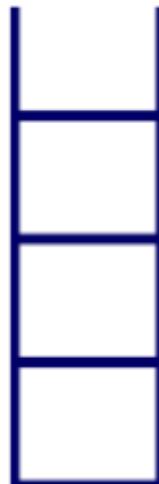
Reading  
Symbol

Stack  
Operations

Evaluated  
Part of Expression

Initially

Stack is Empty



Nothing

5

push(5)



Nothing

3

push(3)



Nothing

+

```
value1 = pop()  
value2 = pop()  
result = value2 + value1  
push(result)
```

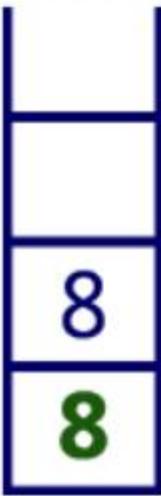


```
value1 = pop(); // 3  
value2 = pop(); // 5  
result = 5 + 3; // 8  
Push( 8 )
```

**(5 + 3)**

8

```
push(8)
```



**(5 + 3)**

2

`push(2)`



$(5 + 3)$

-

```

value1 = pop()
value2 = pop()
result = value2 - value1
push(result)

```



```

value1 = pop() // 2
value2 = pop() // 8
result = 8 - 2; // 6
Push( 6 )

```

$(8 - 2)$

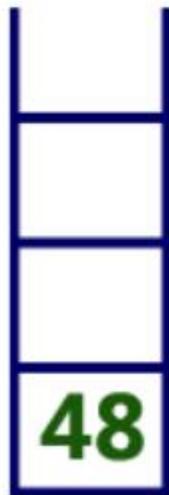
$(5 + 3), (8 - 2)$

\*

\$

End of Expression

```
value1 = pop()
value2 = pop()
result = value2 * value1
push(result)
```



result = pop()



```
value1 = pop() // 6
value2 = pop() // 8
result = 8 * 6; // 48
Push( 48 )
```

**(6 \* 8)**

**(5 + 3) \* (8 - 2)**

Display (result)

**48**

As final result



Infix Expression  $(5 + 3) * (8 - 2) = 48$

Postfix Expression  $5\ 3\ +\ 8\ 2\ -\ *$  value is  $48$



2 3 4 \* +

14

3 4 \* 2 5 \* +

22

#define directive allows definition of Macros within your source code.

Macro definitions allows constant values to be declared for the use throughout your code.

## Infix Expression : A+B\*(C^D-E)

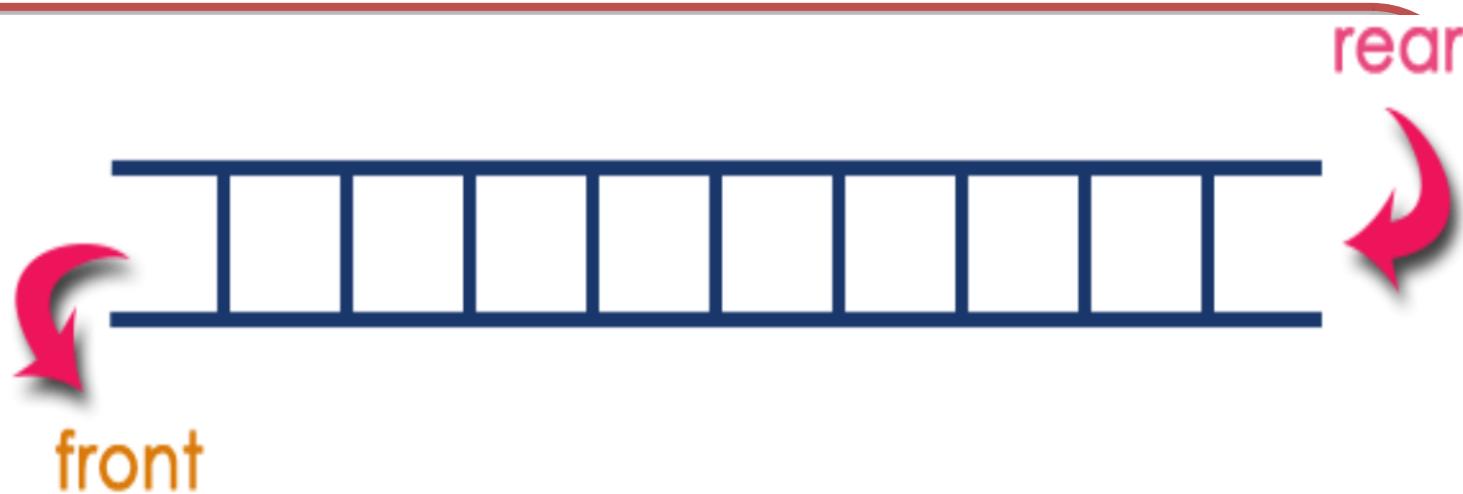
Token	Action	Result	Stack	Notes
A	Add A to the result	A		
+	Push + to stack	A	+	
B	Add B to the result	AB	+	
*	Push * to stack	AB	* +	* has higher precedence than +
(	Push ( to stack	AB	( * +	
C	Add C to the result	ABC	( * +	
^	Push ^ to stack	ABC	^ ( * +	
D	Add D to the result	ABCD	^ ( * +	
-	Pop ^ from stack and add to result	ABCD^	( * +	- has lower precedence than ^
	Push - to stack	ABCD^	- ( * +	
E	Add E to the result	ABCD^E	- ( * +	
)	Pop - from stack and add to result	ABCD^E-	( * +	Do process until ( is popped from stack
	Pop ( from stack	ABCD^E-	* +	
	Pop * from stack and add to result	ABCD^E-*	+	Given expression is iterated, do Process till stack is not Empty, It will give the final result
	Pop + from stack and add to result	ABCD^E-*+		

**Postfix Expression : ABCD^E-\*+**



## Queue ADT

- ❖ Queue is a linear data structure in which the insertion and deletion operations are performed at two different ends.
- ❖ In a queue data structure, adding and removing elements are performed at two different positions.
- ❖ The insertion is performed at one end and deletion is performed at another end.
- ❖ In a queue data structure, the insertion operation is performed at a position which is known as 'rear' and the deletion operation is performed at a position which is known as 'front'.
- ❖ In queue data structure, the insertion and deletion operations are performed based on **FIFO (First In First Out)** principle.



In a queue data structure, the insertion operation is performed using a function called "**enQueue()**" and deletion operation is performed using a function called "**deQueue()**".

**"Queue data structure is a collection of similar data items in which insertion and deletion operations are performed based on FIFO principle".**



## Example

Queue after inserting 25, 30, 51, 60 and 85.

After Inserting five elements...





## Operations on a Queue

The following operations are performed on a queue data structure...

**enQueue(value) - (To insert an element into the queue)**

**deQueue() - (To delete an element from the queue)**

**display() - (To display the elements of the queue)**

Queue data structure can be implemented in two ways.

They are as follows...

**1. Using Array**

**2. Using Linked List**

When a queue is implemented using an array, that queue can organize an only limited number of elements. When a queue is implemented using a linked list, that queue can organize an unlimited number of elements.



```
void enQueue(int value){  
    if(rear == SIZE-1)  
        printf("\nQueue is Full!!! ");  
    else  
    {  
        if(front == -1)  
            front = 0;  
        rear++;  
        queue[rear] = value;  
        printf("\nInsertion success!!!");  
    }  
}
```



```
void deQueue(){  
    if(front == -1)  
        printf("\nQueue is Empty!!! Deletion is not possible!!!!");  
    else{  
        printf("\nDeleted : %d", queue[front]);  
        front++;  
        if(front > rear)  
            front = rear = -1;  
    }  
}
```



```
void display()
{
    if(rear == -1)
        printf("\nQueue is Empty!!!");
    else{
        int i;
        printf("\nQueue elements are:\n");
        for(i=front; i<=rear; i++)
            printf("%d\t",queue[i]);
    }
}
```

# Enqueue & Dequeue Operation in Queues

a[0]	a[1]	a[2]	a[3]	a[4]
empty	empty	empty	empty	empty

Front = -1  
Rear = -1

front				
10	empty	empty	empty	empty
rear				

Enqueue always happens  
at rear of the queue

← Enqueue 10

front				
10	20	empty	empty	empty
rear				

← Enqueue 20

front				
10	20	30	empty	empty
rear				

← Enqueue 30

front				
10	20	30	40	empty
rear				

← Enqueue 40

Dequeue always happens  
at front of the queue

front				
10	20	30	40	empty
rear				

Dequeue ←

front				
empty	20	30	40	empty
rear				

Dequeue ←

front				
empty	empty	30	40	empty
rear				



# Linked List

- When we want to work with an **unknown** no of data values
- We use a linked list **data structure** to **organize** that data.
- LL is a linear data structure that contains a sequence of elements such that each element's links to its next element in the sequence.
- Each element is a linked list is called "**Node**".

SLL

Sequence of elements

Every element has link to its next element.



# Linked List

## Types of Linked List

### ➤ Single Linked List

Item navigation is forward only.

### ➤ Doubly Linked List

Items can be navigated forward and backward.

### ➤ Circular Linked List

Last item contains link of the first element as next and the first element has a link to the last element as previous.



Individual element

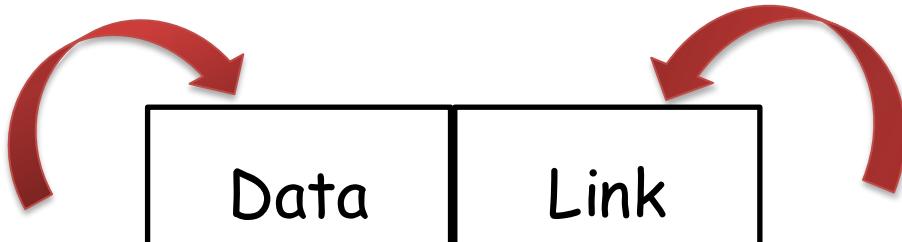
Node

Data Field

Stores value of node

Link Field

Address of next node in sequence



Stores Address of  
next node

Stores Actual Values



## Basic Operations

### ➤ **Insertion**

Adds an element of list.

### ➤ **Deletion**

Deletes an element of the list.

### ➤ **Display**

Displays the complete list.

### ➤ **Search**

Searches an element using the given key.

### ➤ **Delete**

Deletes an element using the given key.



Before we implement actual operations, first we need to setup empty list. First perform the following steps before implementing actual operations.

**Step 1:** Include all the **header files** which are used in the program.

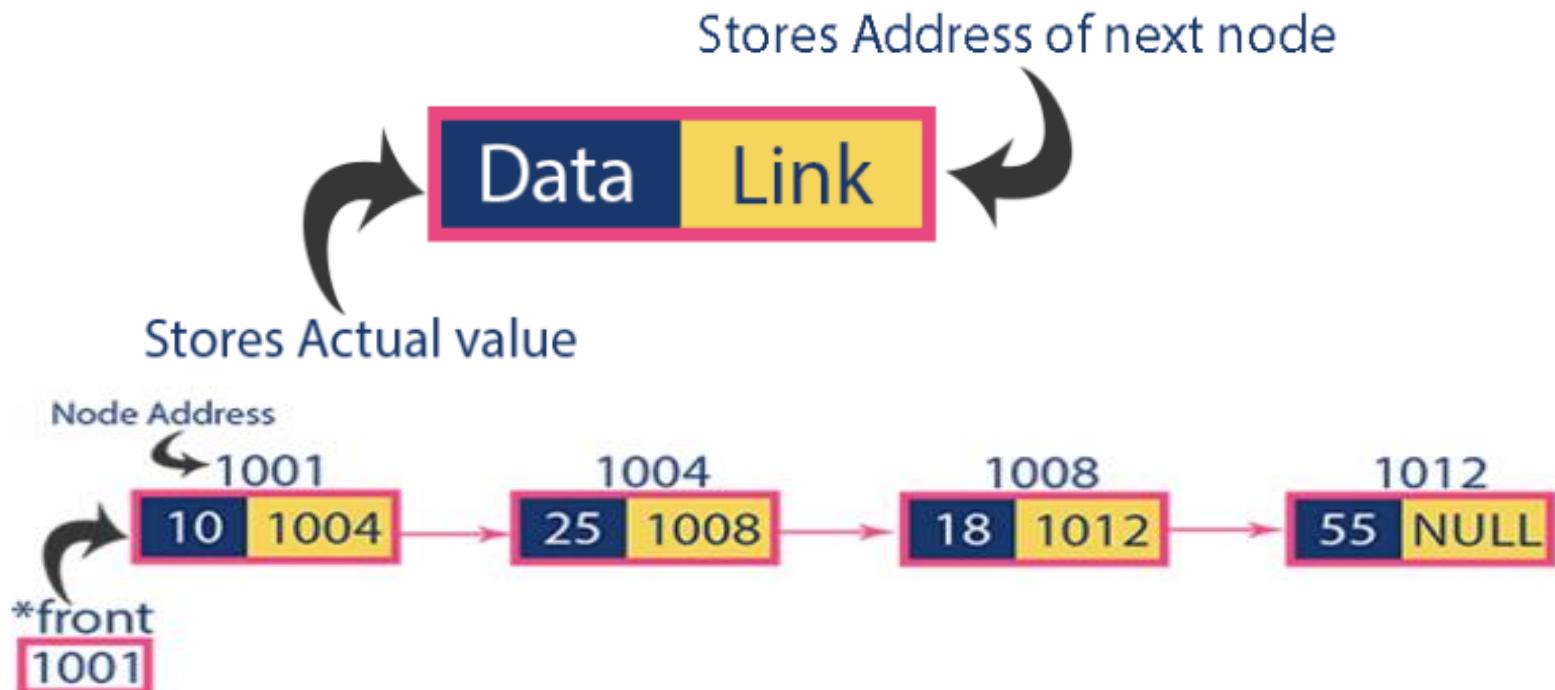
**Step 2:** Declare all the **user defined functions**.

**Step 3:** Define a **Node** structure with two members **data** and **next**

**Step 4:** Define a Node pointer '**start**' and set it to **NULL**.

**Step 4:** Implement the **main** method by displaying operations menu and make suitable function calls in the main method to perform user selected operation.

- Single linked list is a sequence of elements in which every element has link to its next element in the sequence.
- In any single linked list, the individual element is called as "Node". Every "Node" contains two fields, **data** and **next**. The **data** field is used to store actual value of that node and next field is used to store the address of the next node in the sequence.



# Insertion



In a single linked list, the insertion operation can be performed in three ways. They are as follows...

Inserting At Beginning of the list

Inserting At End of the list

Inserting At Specific location in the list

## Inserting At Beginning of the list

**Step 1:** Create a **temp** with given value.

**Step 2:** Check whether list is Empty (**start == NULL**)

**Step 3:** If it is Empty then,

set **temp→next = NULL** and **start = temp**.

**Step 4:** If it is Not Empty then,

set **temp→next = start** and **start = temp**.





```
void insertAtBeginning(int value)
{
```

```
    struct Node *temp;
    temp = (struct Node*)malloc(sizeof(struct Node));
    temp->data = value;
    temp->next = NULL;
    if(start == NULL)
    {
        start = temp;
    }
    else
    {
        temp->next = start;
        start = temp;
    }
    printf("\nOne node inserted!!!\n");
}
```

# Inserting At End of the list



**Step 1:** Create a **temp** with given value and  
**temp → next as NULL.**

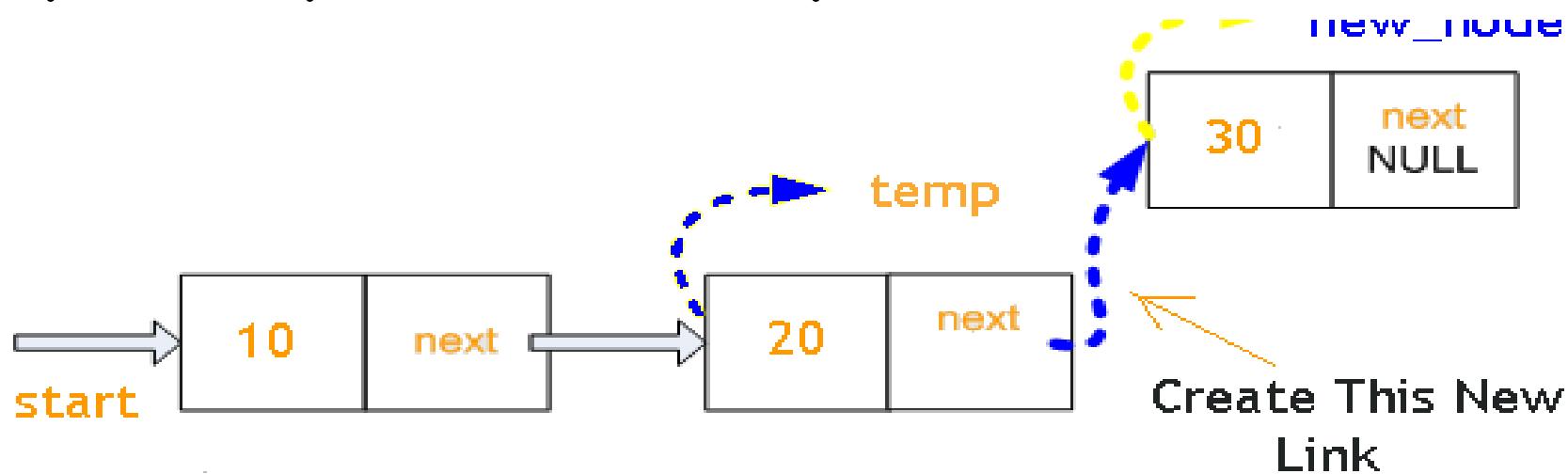
**Step 2:** Check whether list is Empty (**start == NULL**).

**Step 3:** If it is **Empty** then, set **start = temp**.

**Step 4:** If it is **Not Empty** then, define a node  
pointer **ptr** and initialize with **start**.

**Step 5:** Keep moving the **ptr** to its next node until it  
reaches to the last node in the list  
(until **ptr → next** is equal to **NULL**).

**Step 6:** Set **ptr → next = temp**.





```
void insertAtEnd(int value)
{
    struct Node *temp;
    temp = (struct Node*)malloc(sizeof(struct Node));
    temp->data = value;
    temp->next = NULL;
    if(start == NULL)
        start = temp;
    else
    {
        struct Node *ptr = start;
        while(ptr->next != NULL)
            ptr = ptr->next;
        ptr->next = temp;
    }
    printf("\nOne node inserted!!!\n");
}
```



Step 1: Create a **temp** with given value.

Step 2: Check whether list is **Empty** (**start == NULL**)

Step 3: If it is **Empty** then, set **temp → next = NULL** and **start = temp**.

Step 4: If it is **Not Empty**, define node pointer **ptr** and set with **start**.

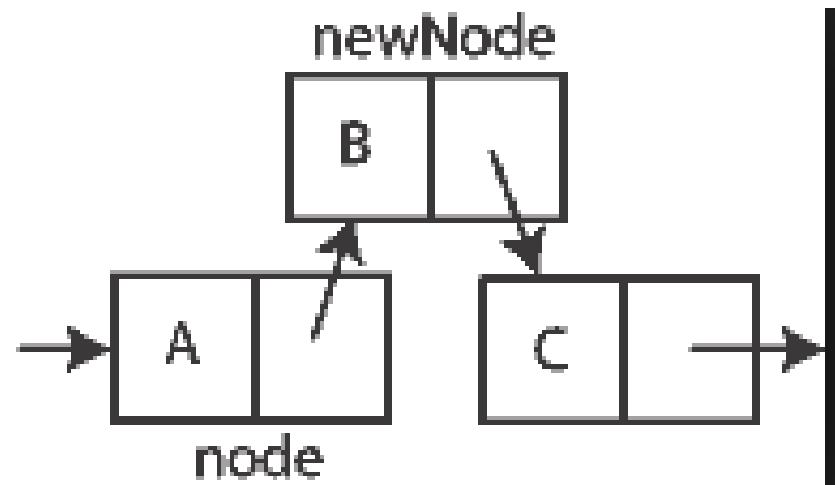
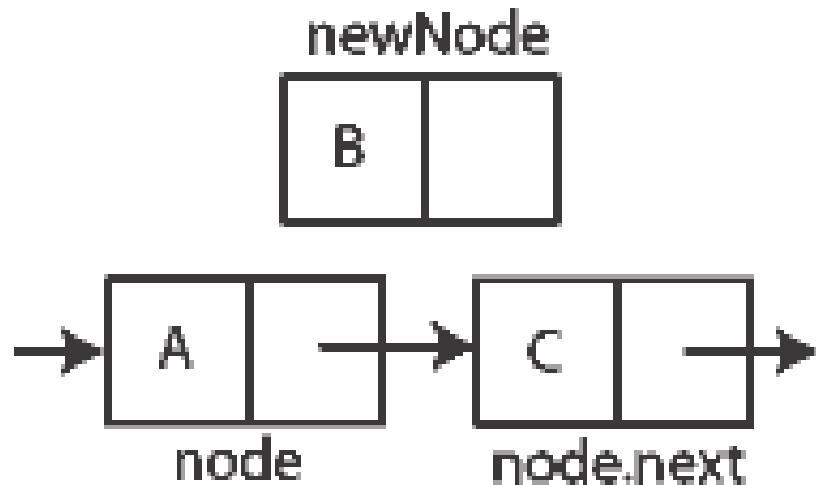
Step 5: Keep moving the **ptr** to its next node until it finds node which we want to insert the **temp** (until **ptr → data = location**, )

Step 6: Every time check whether **ptr** is reached to last node or not. If it is reached to last node then display '**Given node is not found in the list!!!**'

**Insertion not possible!!!**' and terminate the function.

Otherwise move the **ptr** to next node.

Step 7: Finally, Set '**temp → next = ptr → next**' and '**ptr → next = temp**'



## void insertBetween(int value, int loc1, int loc2)



```
{  
    struct Node *temp;  
    temp = (struct Node*)malloc(sizeof(struct Node));  
    temp->data = value;  
    temp->next = NULL;  
    if(start == NULL)  
    {  
        start = temp;  
    }  
    else  
    {  
        struct Node *ptr = start;  
        while(ptr->data != loc1 && ptr->next->data != loc2)  
        {  
            ptr = ptr->next;  
            temp->next = ptr->next;  
            ptr->next = temp;  
        }  
        printf("\nOne node inserted!!!\n");  
    }  
}
```

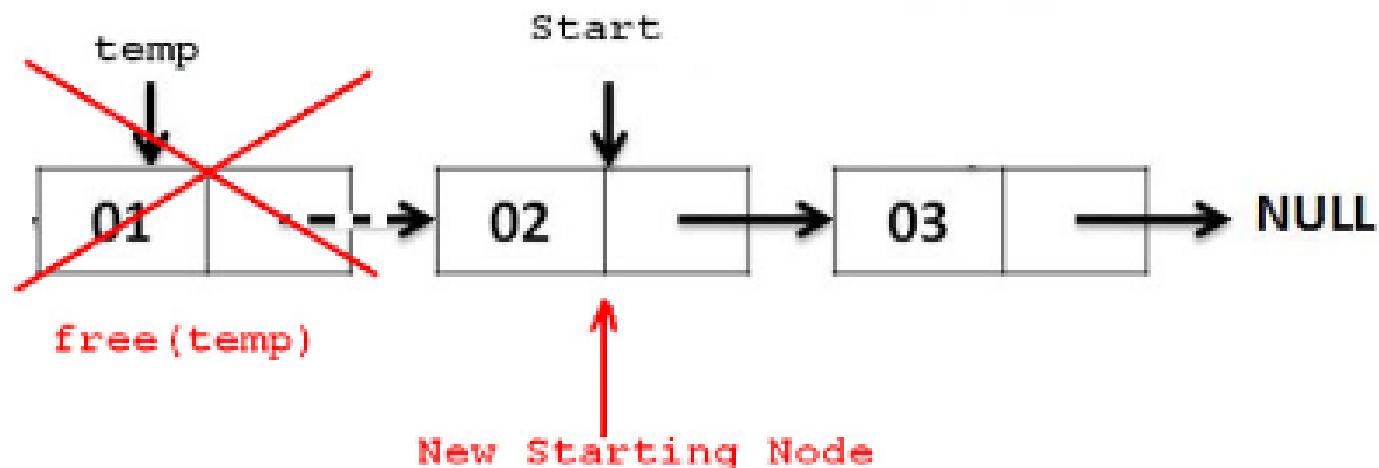
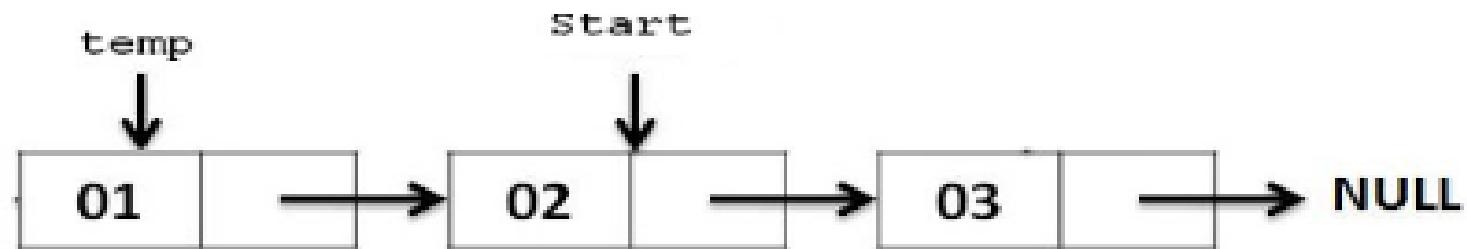
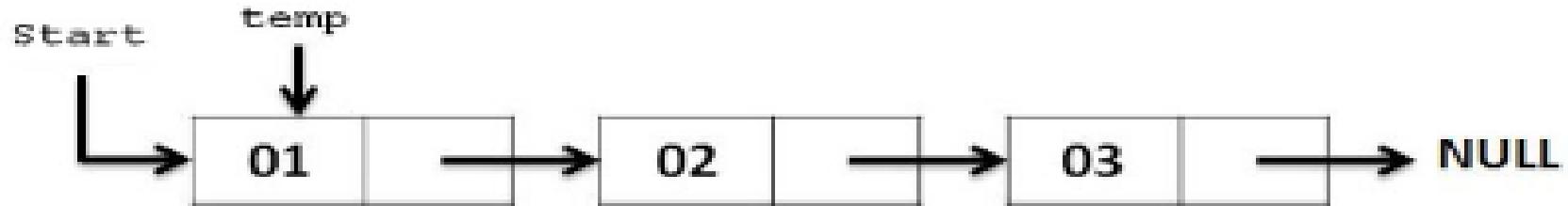
# DELETION FROM BEGIN



**Step 1 : Store Current Start in Another Temporary Pointer**

**Step 2 : Move Start Pointer One position Ahead**

**Step 3 : Delete temp i.e Previous Starting Node as we have Updated Version of Start Pointer**





## void removeBeginning()

{

if(start == NULL)  
printf("\n\nList is Empty!!!");

else

{

struct Node \*ptr =start;

if(start->next == NULL)

{

start = NULL;

free(ptr);

}

else

{

start = ptr->next;

free(ptr);

printf("\nOne node deleted!!!\n\n");

}

}

}

KLEF

(CTSD)

BES-1

## DELETE AT END

**Step 1:** Check whether list is Empty (`start == NULL`)

**Step 2:** If it is Empty then, display 'List is Empty!!! Deletion is not possible' and terminate the function.

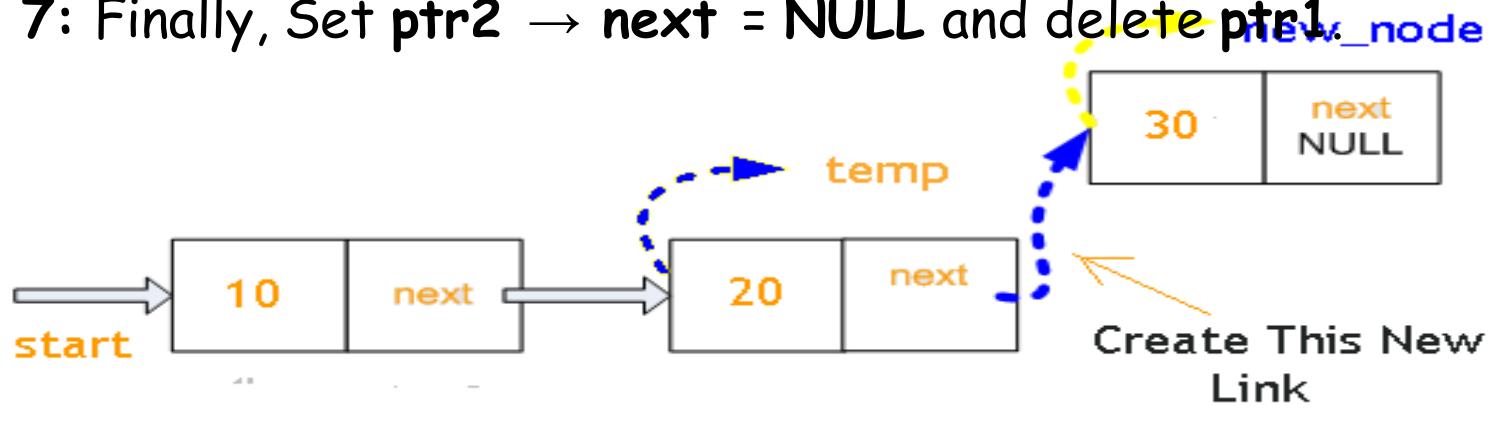
**Step 3:** If it is Not Empty then, define two Node pointers '`ptr1`' and '`ptr2`' and initialize '`ptr1`' with `start`.

**Step 4:** Check whether list has only one Node (`ptr1 -> next == NULL`)

**Step 5:** If it is TRUE. Then, set `start = NULL` and delete `ptr1`. And terminate the function. (Setting Empty list condition)

**Step 6:** If it is FALSE. Then, set '`ptr2 = ptr1`' and move `ptr1` to its next node. Repeat the same until it reaches to the last node in the list. (until `ptr1 -> next == NULL`)

**Step 7:** Finally, Set `ptr2 -> next = new_node`





## void removeEnd()

```
{  
    if(start == NULL)  
    {  
        printf("\nList is Empty!!!\n"); }  
    else  
    {
```

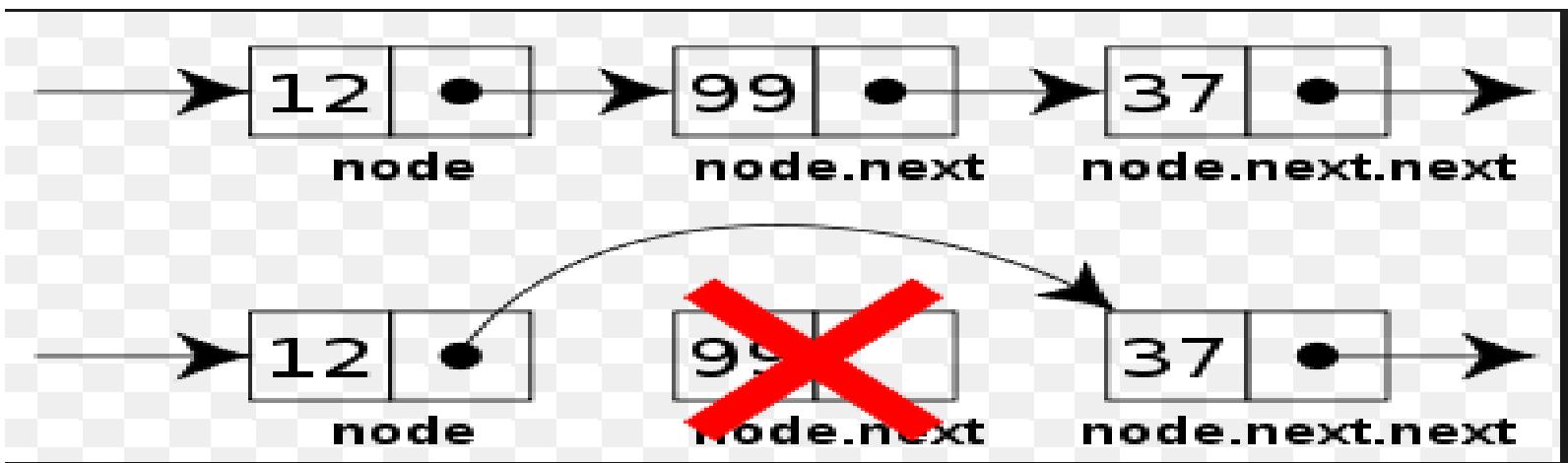
```
        struct Node *ptr1 = start,*ptr2;  
        if(start->next == NULL)  
            start = NULL;  
        else  
        {  
            while(ptr1->next != NULL)  
            {  
                ptr2 = ptr1;  
                ptr1 = ptr1->next;  
            }  
            ptr2->next = NULL;  
        } free(ptr1);  
        printf("\nOne node deleted!!!\n\n");
```

```
}
```



## void removeSpecific(int delValue)

```
{  
    struct Node *ptr1 = start, *ptr2;  
    while(ptr1->data != delValue)  
    {  
        if(ptr1->next == NULL)  
        {  
            printf("\nGiven node not found in the list!!!");  
        } ptr2 = ptr1;  
        ptr1 = ptr1->next;  
    } ptr2->next = ptr1->next;  
    free(ptr1);  
    printf("\nOne node deleted!!!\n\n");  
}
```





```
void display()
{
```

```
    if(start == NULL)
    {
        printf("\nList is Empty\n");
    }
    else
    {
        struct Node *ptr = start;
        printf("\n\nList elements are - \n");
        while(ptr->next != NULL)
        {
            printf("%d --->", ptr->data);
            ptr = ptr->next;
        }
        printf("%d --->NULL", ptr->data);
    }
}
```

}



```
void display()
{
```

```
    if(start == NULL)
    {
        printf("\nList is Empty\n");
    }
    else
    {
        struct Node *ptr = start;
        printf("\n\nList elements are - \n");
        while(ptr->next != NULL)
        {
            printf("%d --->", ptr->data);
            ptr = ptr->next;
        }
        printf("%d --->NULL", ptr->data);
    }
}
```

}



```
void display()
{
```

```
    if(start == NULL)
    {
        printf("\nList is Empty\n");
    }
    else
    {
        struct Node *ptr = start;
        printf("\n\nList elements are - \n");
        while(ptr->next != NULL)
        {
            printf("%d --->",ptr->data);
            ptr = ptr->next;
        }
        printf("%d --->NULL",ptr->data);
    }
}
```

}



```
void display()
{
```

```
    if(start == NULL)
    {
        printf("\nList is Empty\n");
    }
    else
    {
        struct Node *ptr = start;
        printf("\n\nList elements are - \n");
        while(ptr->next != NULL)
        {
            printf("%d --->",ptr->data);
            ptr = ptr->next;
        }
        printf("%d --->NULL",ptr->data);
    }
}
```

}



## Tree - Terminology

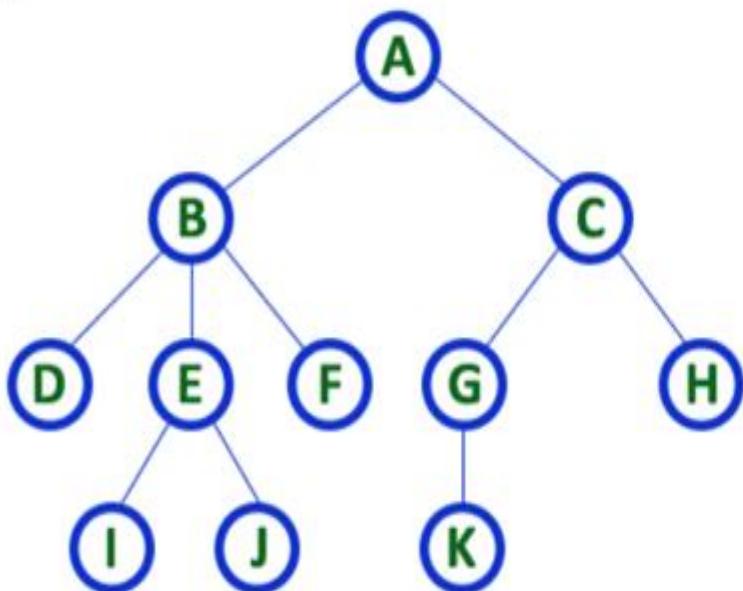
- ❖ In linear data structure data is organized in sequential order
- ❖ non-linear data structure data is organized in random order. Tree is a very popular non-linear data structure used in wide range of applications.
- ❖ A tree data structure can be defined as follows...

Tree is a non-linear data structure which organizes data in hierarchical structure and this is a recursive definition.

Tree data structure is a collection of data (Node) which is organized in hierarchical structure recursively

- ❖ In tree data structure, every individual element is called as **Node**
- ❖ In a tree data structure, if we have **N** number of nodes then we can have a maximum of **N-1** number of links.

### Example

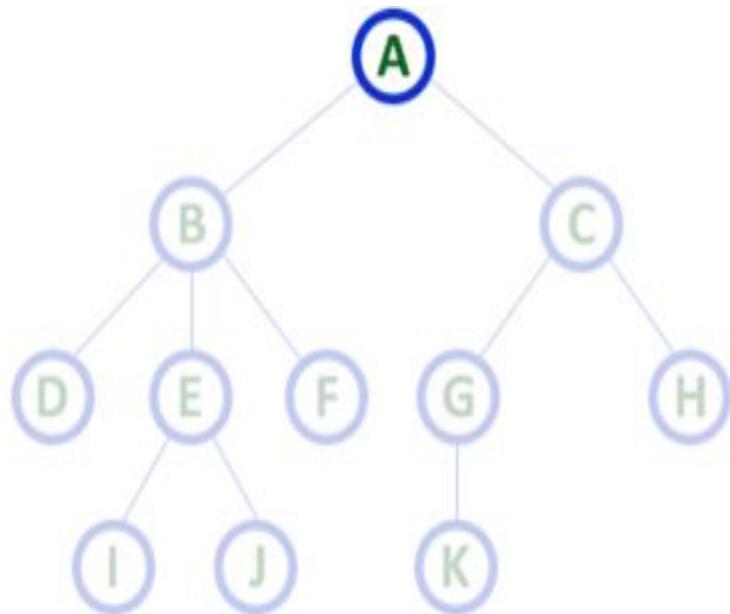


**TREE with 11 nodes and 10 edges**

- In any tree with '**N**' nodes there will be maximum of '**N-1**' edges
- In a tree every individual element is called as '**NODE**'

# Root

- ❖ In a tree data structure, the first node is called as **Root Node**.
- ❖ Every tree must have root node.
- ❖ Root node is the origin of tree data structure.
- ❖ In any tree, there must be only one root node.

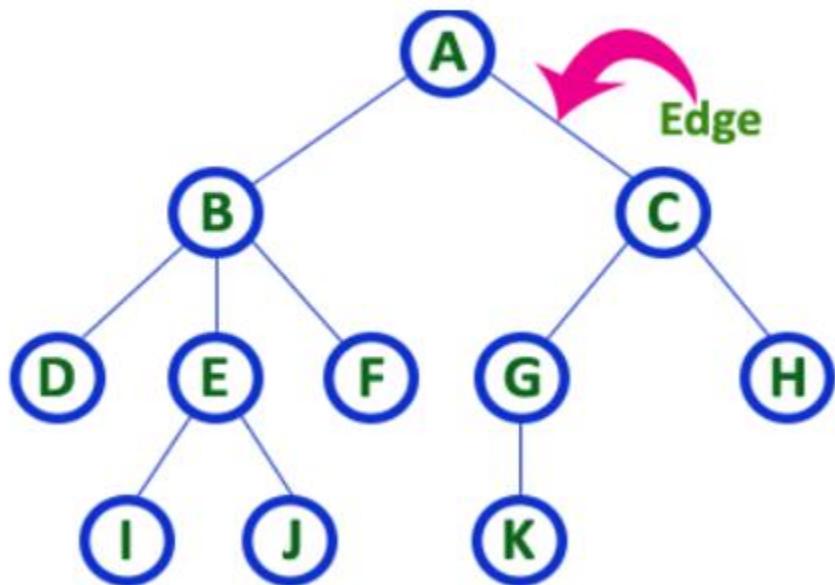


Here 'A' is the 'root' node

- In any tree the first node is called as **ROOT node**

# Edge

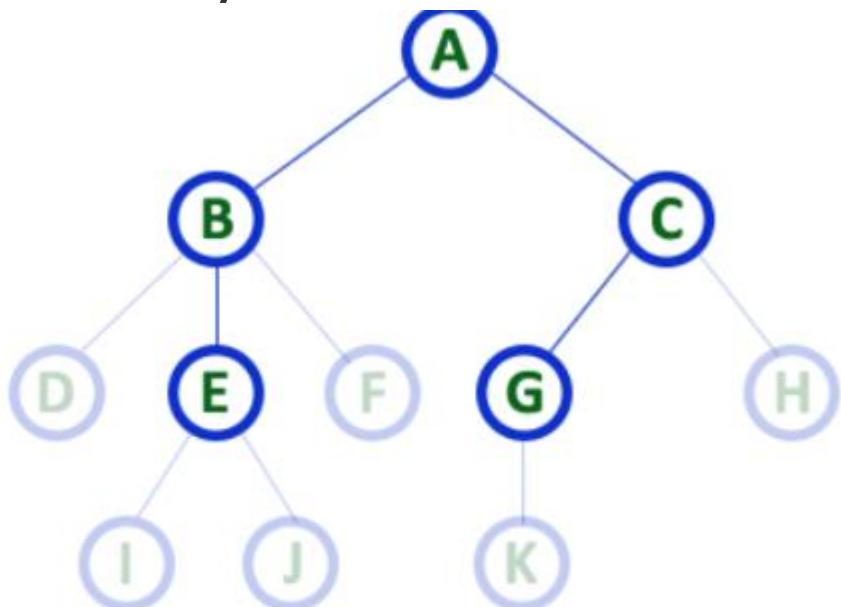
- ❖ In a tree data structure, the connecting link between any two nodes is called as EDGE.
- ❖ In a tree with 'N' number of nodes there will be a maximum of 'N-1' number of edges.



- In any tree, 'Edge' is a connecting link between two nodes.

# Parent

- ❖ In a tree data structure, the node which is predecessor of any node is called as **PARENT NODE**.
- ❖ In simple words, the node which has branch from it to any other node is called as parent node.
- ❖ Parent node can also be defined as "**The node which has child / children**".

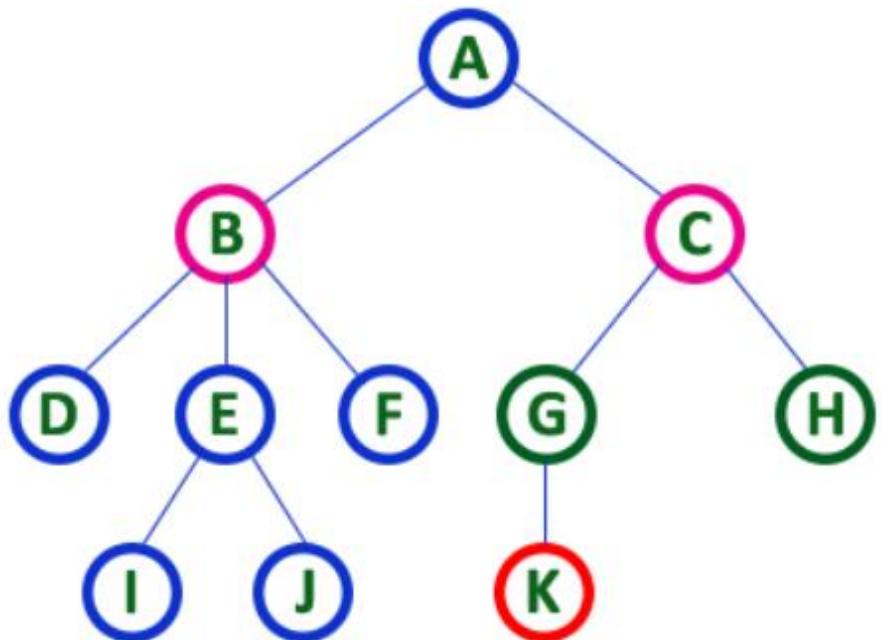


Here A, B, C, E & G are **Parent nodes**

- In any tree the node which has child / children is called '**Parent**'
- A node which is predecessor of any other node is called '**Parent**'

## Child

- ❖ Node which is descendant of any node is called as **CHILD Node**.
- ❖ In simple words, the node which has a link from its parent node is called as child node.
- ❖ In a tree, any parent node can have any number of child nodes.
- ❖ In a tree, all the nodes except root are child nodes.



Here B & C are **Children of A**

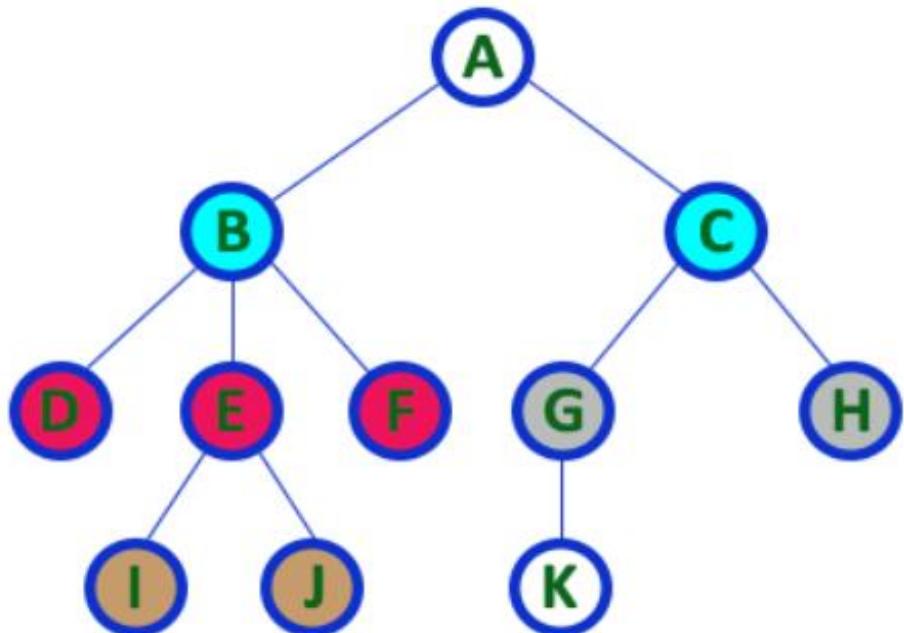
Here G & H are **Children of C**

Here K is **Child of G**

- **descendant of any node is called as CHILD Node**

# Siblings

- ❖ Nodes which belong to same Parent are called as **SIBLINGS**.
- ❖ In simple words, the nodes with same parent are called as Sibling nodes.



Here **B & C** are Siblings

Here **D E & F** are Siblings

Here **G & H** are Siblings

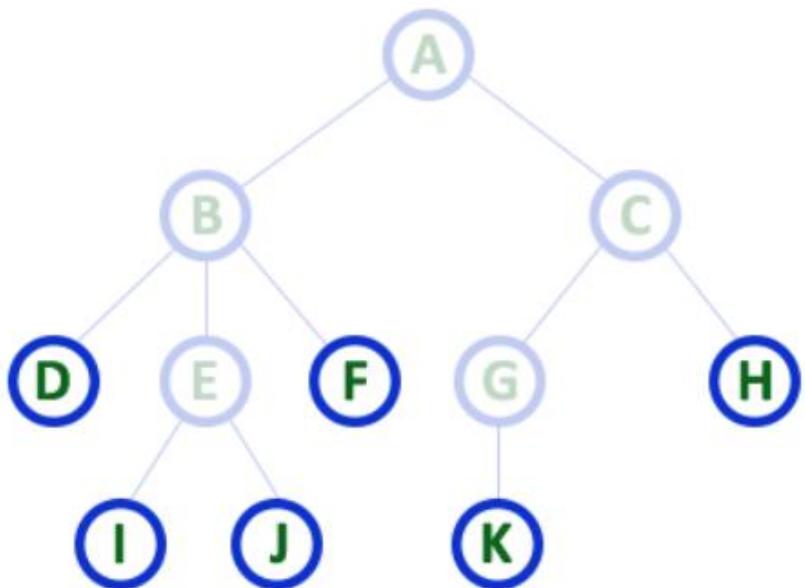
Here **I & J** are Siblings

- In any tree the nodes which has same Parent are called '**Siblings**'

- The children of a Parent are called '**Siblings**'

# Leaf

- ❖ Node which does not have a child is called as **LEAF Node**.
- ❖ In simple words, a leaf is a node with no child.
- ❖ Leaf nodes are also called as **External Nodes**
- ❖ External node is also a node with no child.
- ❖ Leaf node is also called as 'Terminal' node.

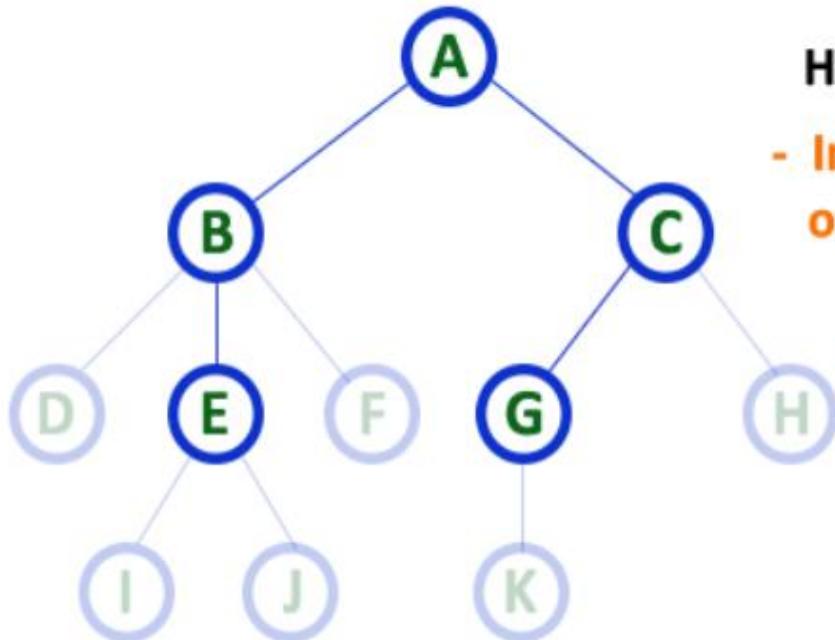


Here D, I, J, F, K & H are **Leaf nodes**

- In any tree the node which does not have children is called '**Leaf**'
- A node without successors is called a '**leaf**' node

# Internal Nodes

- ❖ Node which has atleast one child is called as **INTERNAL Node**.
- ❖ In simple words, an internal node is a node with atleast one child. Nodes other than leaf nodes are called as **Internal Nodes**.
- ❖ **The root node is also said to be Internal Node** if the tree has more than one node.
- ❖ **Internal nodes are also called as 'Non-Terminal' nodes.**

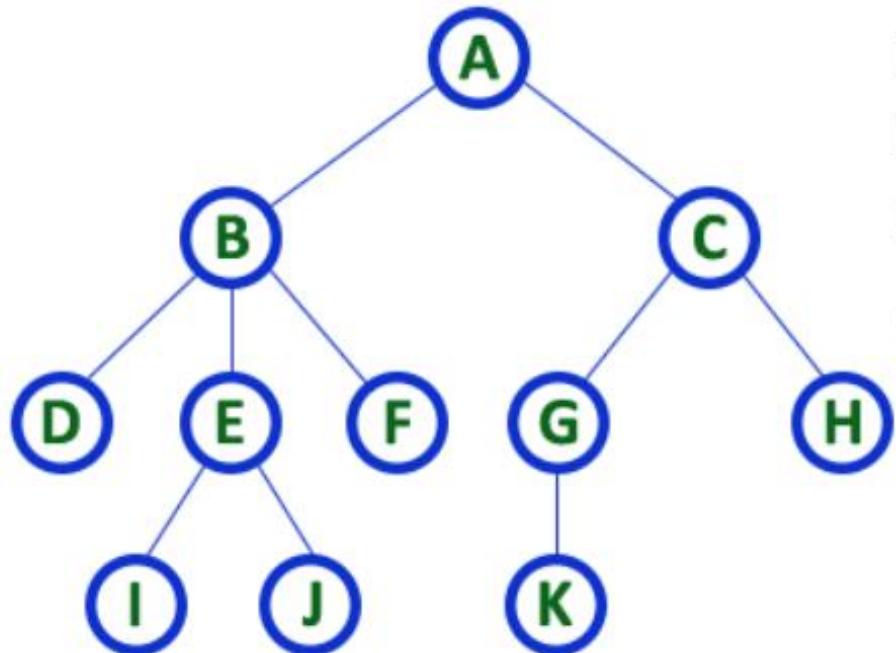


Here A, B, C, E & G are **Internal** nodes

- In any tree the node which has atleast one child is called '**Internal**' node
- Every non-leaf node is called as '**Internal**' node

# Degree

- ❖ Total number of children of a node is called as **DEGREE** of that Node.
- ❖ In simple words, the Degree of a node is total number of children it has.
- ❖ The highest degree of a node among all the nodes in a tree is called as '**Degree of Tree**'



Here Degree of B is 3

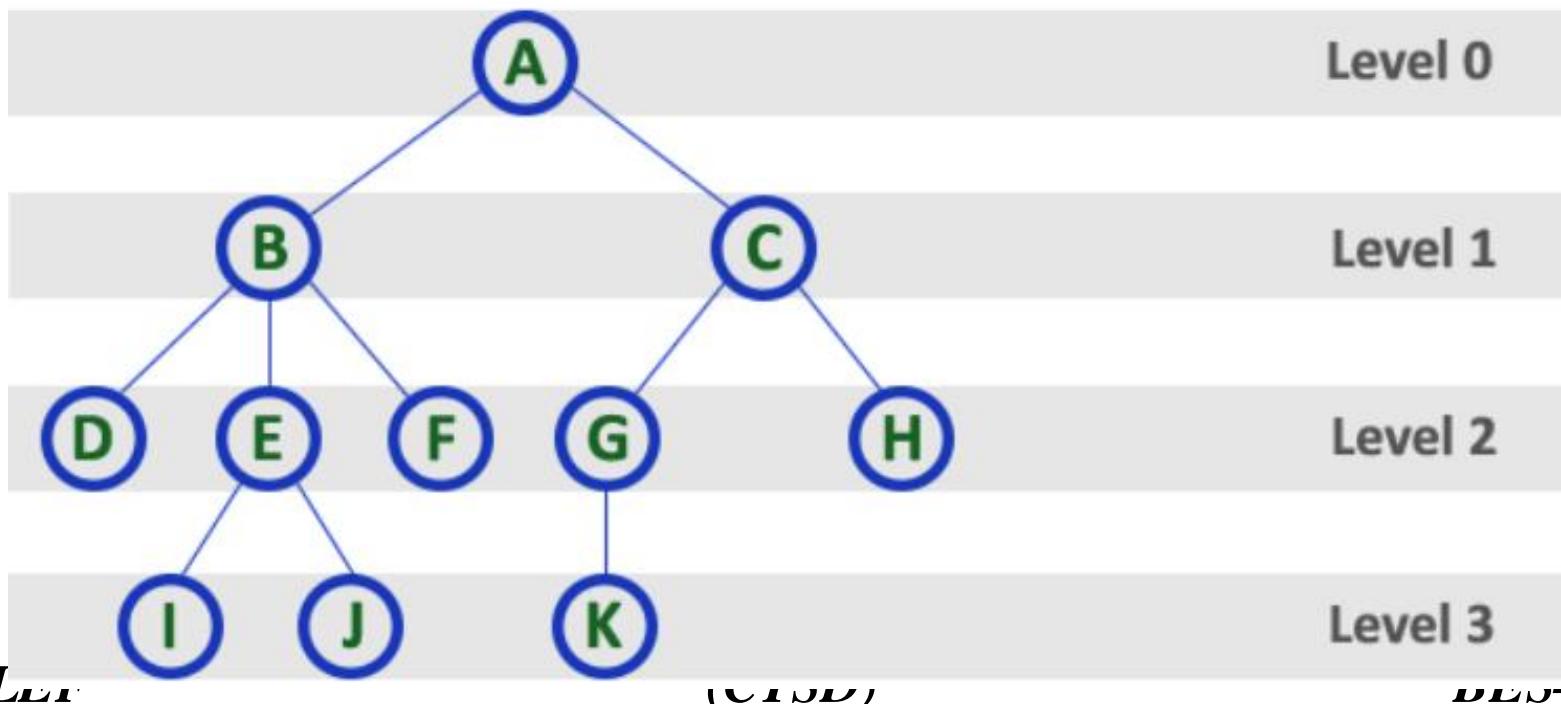
Here Degree of A is 2

Here Degree of F is 0

- In any tree, 'Degree' of a node is total number of children it has.

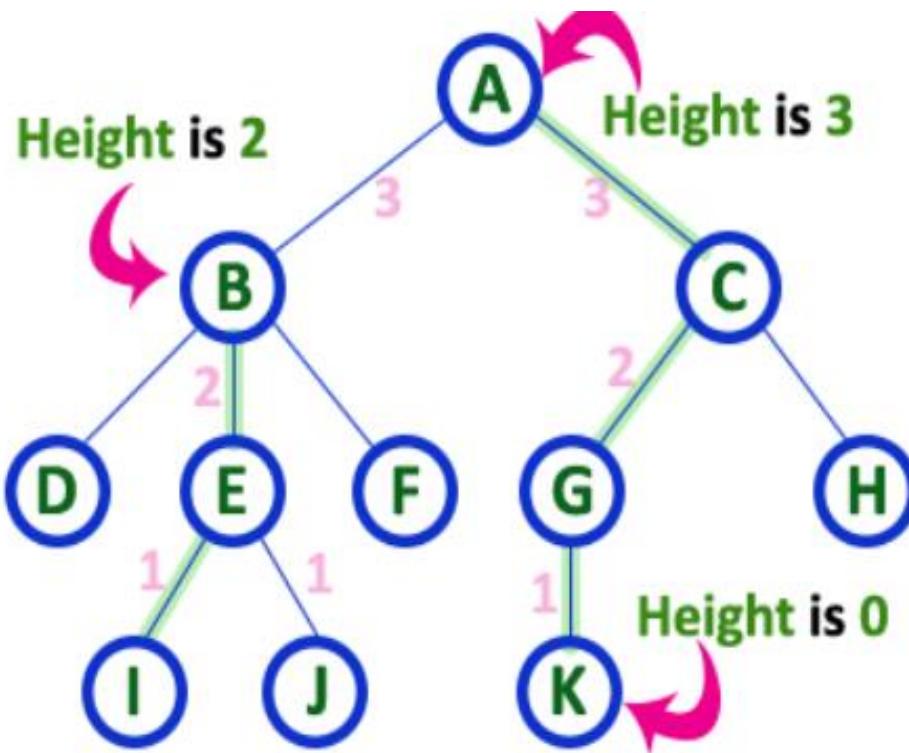
# Level

- ❖ Root node is said to be at Level 0 and the children of root node are at Level 1 and the children of the nodes which are at Level 1 will be at Level 2 and so on...
- ❖ In simple words, in a tree each step from top to bottom is called as a Level and the Level count starts with '0' and incremented by one at each level (Step).



# Height

- ❖ Total number of edges from leaf node to a particular node in the longest path is called as **HEIGHT** of that Node.
- ❖ In a tree, height of the root node is said to be **height of the tree**. In a tree, **height of all leaf nodes is '0'**.

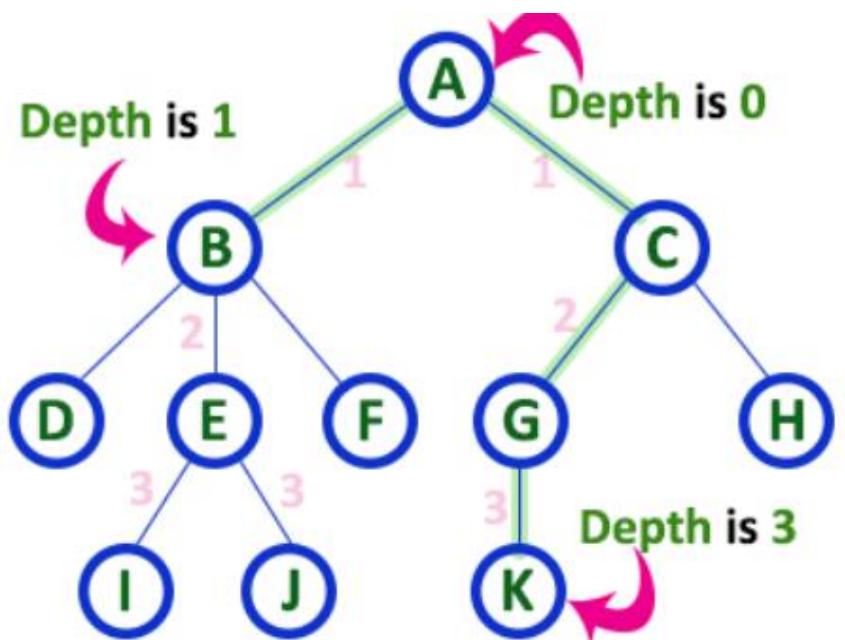


Here Height of tree is 3

- In any tree, '**Height of Node**' is total number of Edges from leaf to that node in longest path.
- In any tree, '**Height of Tree**' is the height of the root node.

# Depth

- ❖ Total number of edges from root node to a particular node is called as **DEPTH** of that Node.
- ❖ In a tree, the total number of edges from root node to a leaf node in the longest path is said to be Depth of the tree.
- ❖ In simple words, the highest depth of any leaf node in a tree is said to be depth of that tree. In a tree, **depth of the root node is '0'.**

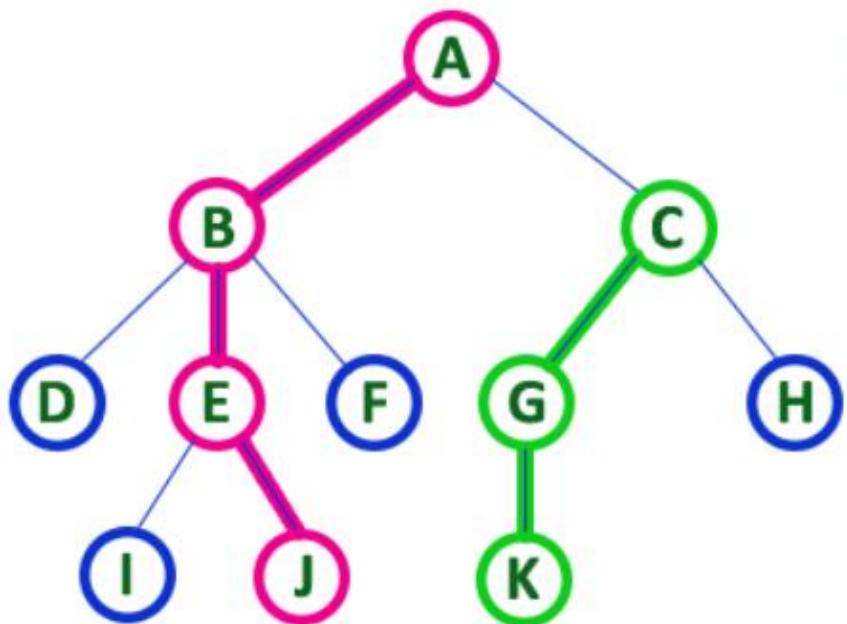


Here Depth of tree is 3

- In any tree, '**Depth of Node**' is total number of Edges from root to that node.
- In any tree, '**Depth of Tree**' is total number of edges from root to leaf in the longest path.

# Path

- ❖ In a tree data structure, the sequence of Nodes and Edges from one node to another node is called as **PATH** between that two Nodes.
- ❖ **Length of a Path** is total number of nodes in that path.
- ❖ In below example the path A - B - E - J has length 4



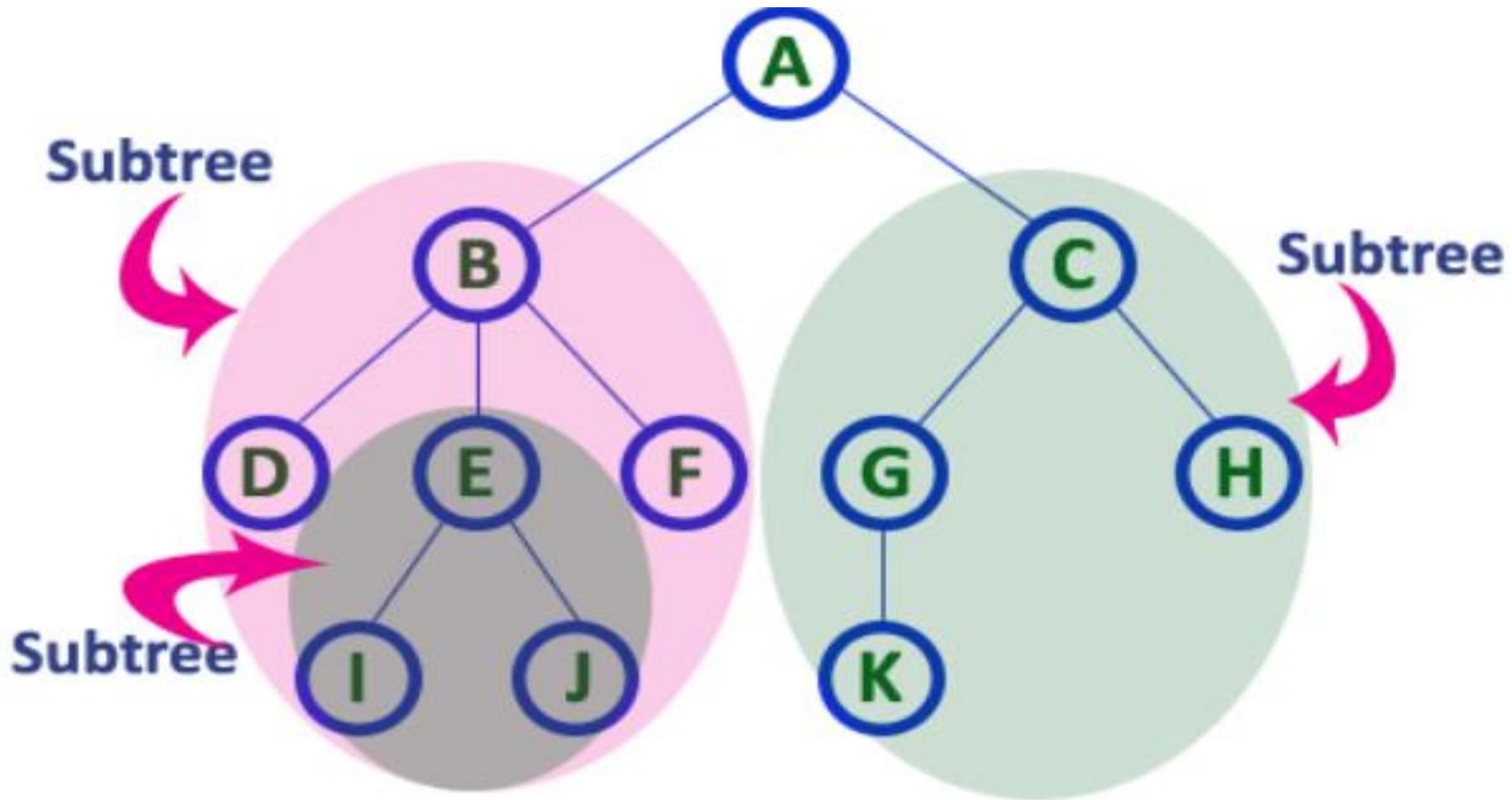
- In any tree, 'Path' is a sequence of nodes and edges between two nodes.

Here, 'Path' between A & J is  
A - B - E - J

Here, 'Path' between C & K is  
C - G - K

# Sub Tree

- ❖ Each child from a node forms a subtree recursively.
- ❖ Every child node will form a subtree on its parent node.

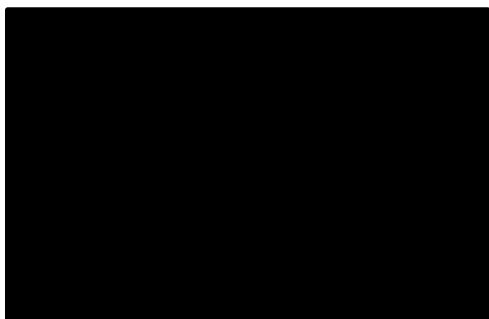




# Command Line Arguments

Command/Character  
User Interface

Command  
Prompt



Inputs

IDE  
Integrated  
Development  
Environment



## Operating System

CUI

GUI

Programmers

End Users

How to pass Inputs/Arguments to Program from  
Command Line is Command Line Arguments



cmd/> program.c      Object file

cmd/> program.exe      

arg1	arg2	arg3	arg4
------	------	------	------

**Input Values**

**Passing Inputs  
to Program**



- ❖ It is possible to pass some values from the command line to your C programs when they are executed.
- ❖ These values are called **command line arguments** and many .
- ❖ To control your program from outside instead of hard coding those values inside the code.
- ❖ The command line arguments are handled using main() function arguments
- ❖ Where **argc** refers to the number of arguments passed, and **argv[]** is a pointer array which points to each argument passed to the program



- ❖ **argv[0]** holds the name of the program itself
- ❖ **argv[1]** is a pointer to the first command line argument supplied
- ❖ **argv[n]** is the last argument.



```
#include<stdio.h>
#include<stdlib.h>
int main(int argc,char *argv[10])
{
    printf("%s\n",argv[0]);
    printf("%s\n",argv[1]);
    printf("%s\n",argv[2]);
    return 0;
}
```



```
#include<stdio.h>
int main(int argc,char *argv[10])
{
    int a,b,c;
    printf("%s\n",argv[0]);
    a = atoi(argv[1]);
    b = atoi(argv[2]);
    c=a+b;
    printf("sum=%d",c);
    return 0;
}
```



```
#include<stdio.h>
#include<stdlib.h>
int main(int argc,char *argv[10])
{
    int sum=0,i;
    printf("%s\n",argv[0]);
    for(i=1;i<=argc;i++)
    {
        sum=sum+atoi(argv[i]);
    }
    printf("sum=%d",sum);
    return 0;
}
```



```
#include<stdio.h>
int main(int argc,char *argv[10])
{
    int i;
    printf("%s\n",argv[0]);
    for(i=1;i<=argc;i++)
    {
        printf("argv[%d]=%s\n",i,argv[i]);
    }
    return 0;
}
```



Thank  
you