1. Concept of Software

Computer can neither think nor take any decision itself, we instruct by means of commands or programs.

Program:-
It is sequence of instructions, which operate on computer data to perform certain well-defined task or achieve a goal.

Software:-Collection of related program is called software.

Example:-

- MS Word
- MS Excel
- MS PowerPoint
- Notepad
- Word Pad
- Paint Brush
- Window Media Player
- VLC
- WIN AMP
- Coral Draw
- Photoshop
- PageMaker
- CAD(Computer Aided Design)
- CAM (Computer Aided Manufacturing.)
- Oracle
- MS Access
- SQL server

Etc.

Program = Algorithm + Flowchart + Language + Translator + Operating System + Data structure

Algorithm

Step by step, problem-solving technique is called algorithm.

Example:-1 **s = a + b**

Step:-1 Start/Begin
Step:-2 Read/Input a, b
Step:-3 Calculate s = a + b
Step:-4 Print s
Step:-5 Stop/End

Example:-2 **si = p*n*r/100**

Where,

si Simple Interest
p Principal Amount
r Rate of interest
n Time Period

Step:-1 Start/Begin
Step:-2 Read/Input p, n, r.
Step:-3 Calculate si= (p*n*r)/100
Step:-4 Print si
Step:-5 Stop/End

Example:-3 Write algorithm of following formula.

$a=p*(1+r/100)^n$

Where,

a	Amount
p	Principal Amount
r	Rate of interest
n	Time Period

Step:-1	Start/Begin
Step:-2	Read/Input p, n, r.
Step:-3	Calculate a= $p*(1+r/100)^n$
Step:-4	Print a
Step:-5	Stop/End

Example:-4 Write algorithms for checking year is **leap** or **Not Leap year**.

Step:-1	Start/Begin	
Step:-2	Read/Input year	
Step:-3	Calculate y = year Mod 4	**//(=Assignment operation)**
Step:-4	If y= =0	**//(== equal operation)**
Step:-5	Print "Leap Year"	
Step:-6	If y!=0	
Step:-7	Print "Not Leap Year"	
Step:-8	Stop/End	

Example:-5 Write algorithms for checking and calculating **real roots of any quadratic equation**.

$ax^2 + bx + c = 0$

Step:-1	Start/Begin
Step:-2	Read/Input coefficient of x*x, x and constant (say, a, b &c)
Step:-3	calculate d=(b*b-4*a*c)
Step:-4	if(d>=0) goto step 5 otherwise goto step-7
Step:-5	Calculate x1=(-b+sqrt(d))/(2*a) and x2=(-b-sqrt(d))/(2*a)
Step:-6	Print Roots of real roots x1 and x2
Step:-7	Print "Roots are imaginary"
Step:-8	Stop/End

Characteristics ⇨

- Finiteness
- Definiteness
- Effectiveness
- Input
- Output

Finiteness **: -** Steps of algorithm must be finite.

Definiteness **: -** Each and every step must be defined.

Effectiveness **: -** Each and every step must be effective.

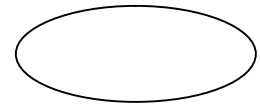
Input **: -** Algorithm must be associated with inputation.

Output **:-**Algorithm must be associated with output components.

Flowchart

The diagrammatical representation of any algorithm is called flow chart.

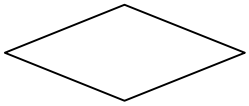
Following symbols are used in flowchart.



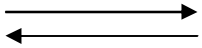
Oval (for Start/Stop)

Parallelogram (for input and output)

Rectangle (for process)

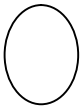


Diamond (for Decision)



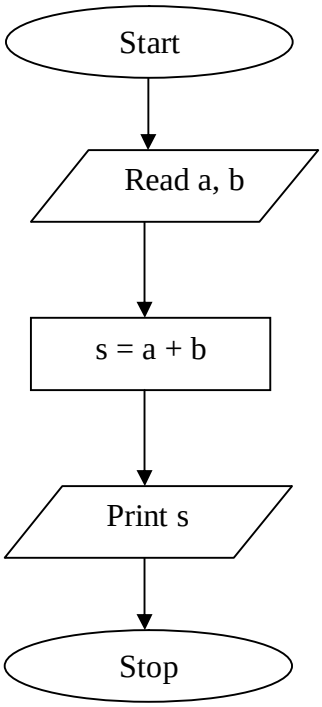
Arrow (for flow direction)

Open Ended Box (for comment)

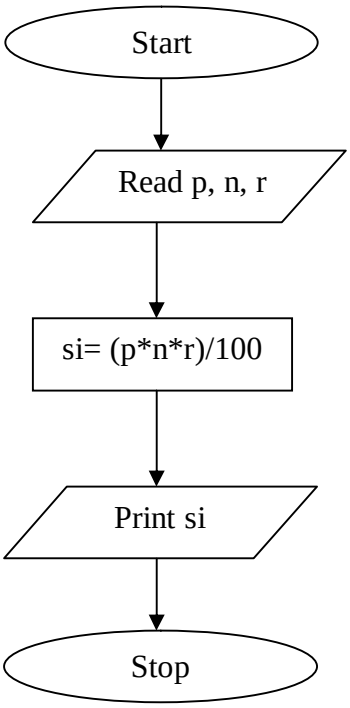


Circle (for connector)

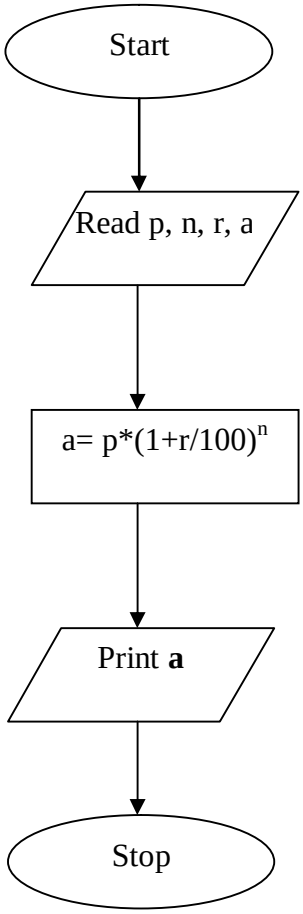
Example:-1 Draw a flow chart for **s = a + b**,



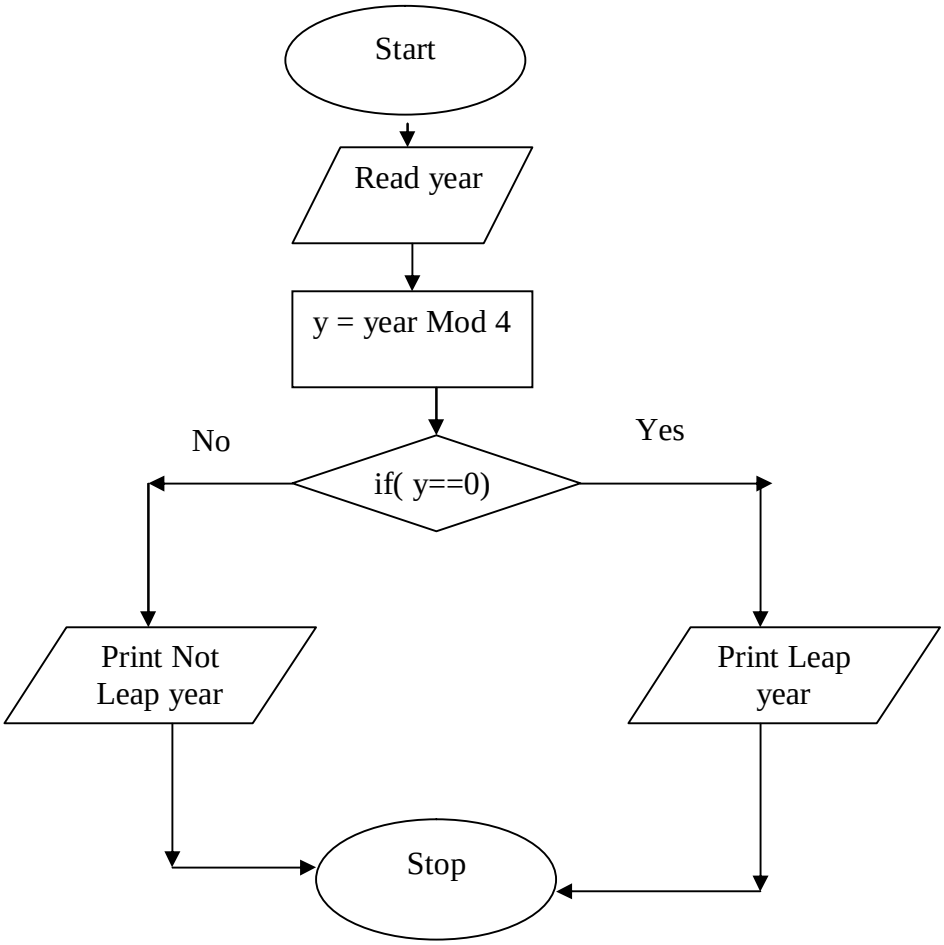
Example:-2 Draw a flow chart for **si = p*n*r/100**



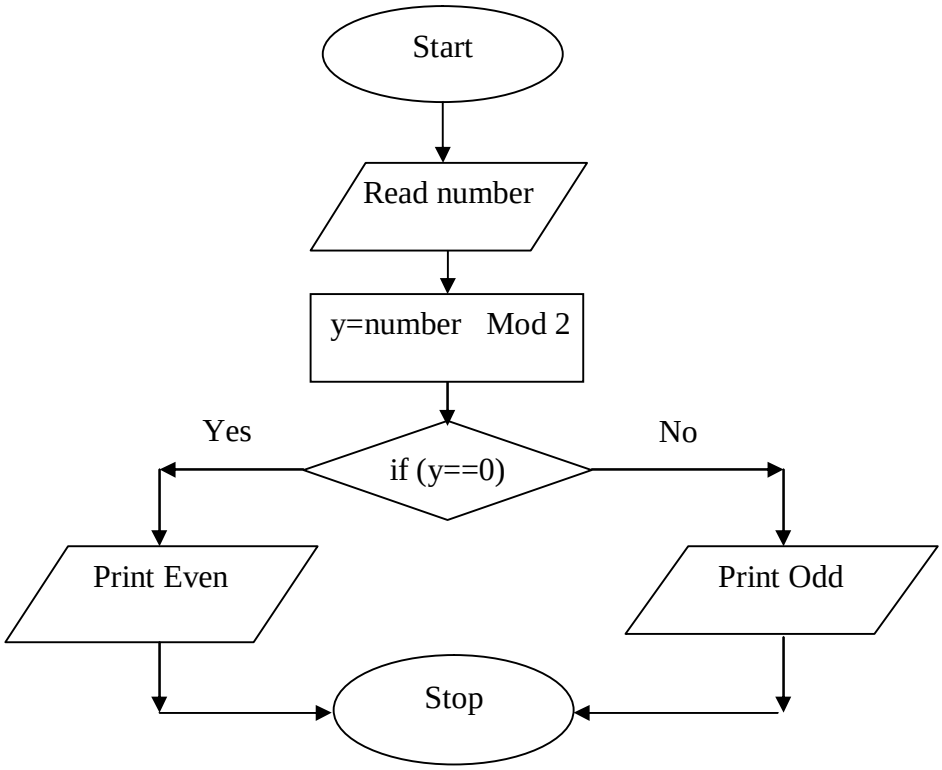
Example:-3 Draw a flow chart for **a = p*(1 + r/100)^n**



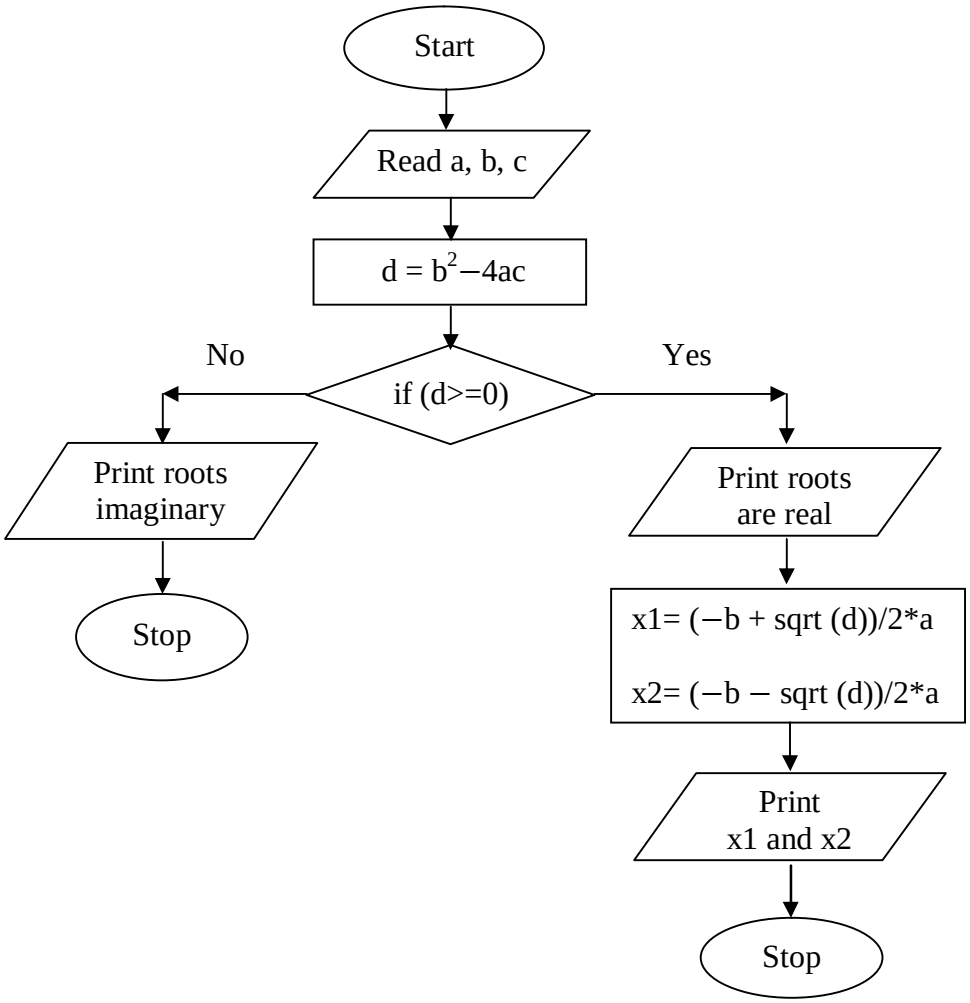
Example:-4 Draw a flow chart for checking year is **leap year** or **not leap year**



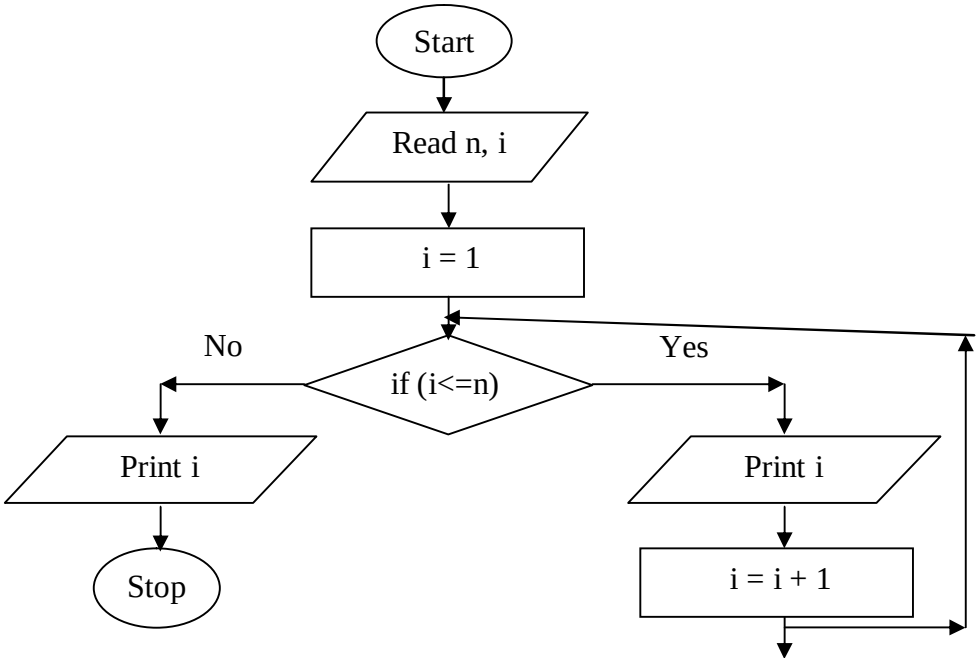
Example:-5 Draw a flow chart for checking number is **even** or **odd**.



Example:-6 Draw a flow chart for **checking and calculating real roots of any quadratic equation**.



Example:-7 Draw a flow chart for **a series of natural number.** Where n = 1, 2, 3, 4, 5, ……



Computer Language

Computer languages are categorized into **two types.**

1. **LLL**(Low Level Language)
   (a) Machine language (**0** and **1**)
   (b) Assembly language (**Symbols**, **codes** are used instead of **0** and **1**)
2. **HLL**(High Level Language)
   Natural English like language.

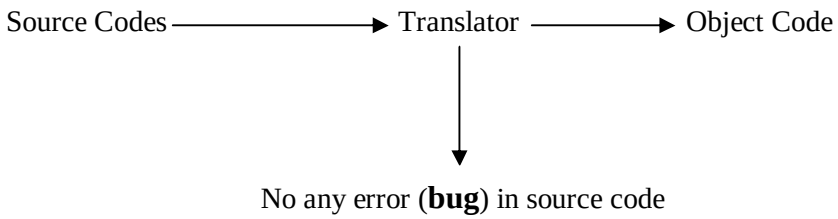
Example:-
C, C++, JAVA, C#, DOTNET, COBOL, PASCAL, FORTRAN, BASIC, LISP, PROLOG Etc.

Advantage of HLL:-
- Easy to understand
- Fast s/w development
- Fast debugging
- Natural English like language
- Better portability.

Translator

It is used for **converting** source (Program) code into object codes (Machine Codes). There are following **three types** of translator.

1. **Assembler** (Only for assembly language)
2. **Interpreter** (Only for Basic Language
3. **Compiler** (All HLL except Basic)



Debug: - To remove error from source codes.

Interpreter	Compiler
1:-Convert source code into object code line by line	1:-Convert entire source code into object code at a time
2:-Debugging is very fast	2:-Debugging is slow.
3:-More Execution time	3:-Less execution Time.
4:-Used only in BASIC	4:-Used in all HLL.

Compiler translator are used only in C Language

2. History of 'C' Language

Initially **Ken Thompson** developed a language known as BCPL (also known a s **'B' Language**).After some time **Dennis Ritchie** modified BCPL language. This modified language was called **'C' language**. Since "C' is the successor of 'B' language in BCPL.

B Basic
C Combined
P Programming
L Language

Note: - **"B" & "C"** Language developed at **Bell laboratories**.

Programming Elements/Building Blocks of C Language

A: - Data Types

1. Simple data Types
   a) Integer data types
   b) Real Data type/Floating data Types
   c) Character data types
2. Structured data types
   a) Arrays
   b) Strings
   c) Structures
   d) Unions
3. Enumerated data Types
4. Pointer Data type
5. Void data Type

Tables of Integer data types:-

Type	Size	Minimum value	Maximum Value
short int/int	2 Byte	−32768	32767
long int	4 Byte	−2147483648	2147483647

Tables of Real data types:-

Type	Size	Minimum value	Maximum Value
float	4 Byte	$3.4*10^{-38}$	$3.4*10^{+38}$
double	8 Byte	$1.7*10^{-308}$	$1.7*10^{+308}$
long double	10 Bytes	$3.4*10^{-4932}$	$1.1*10^{+4932}$

Character data types:-

Space one Byte. It is enclosed into single quote. ASCII range of characters exists between **0** to **255**.

B: - Operators

1. **Arithmetic Operators: -**

+, -, *, /, % →(Remainder or modulo operator)

2. **Relational Operators**

>, <, >=, <=, !=, == (equal)

3. **Logical Operators**

&&(And), ||(Or), !(Not)

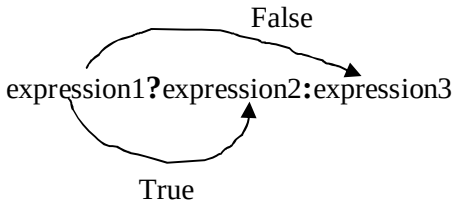
4. **Bitwise operator**

&	Bitwise and
|	Bitwise or
^	Bitwise exclusive or
~	Bitwise compliment Operator
<<	Bitwise left shift
>>	Bitwise Right shift

5. **Increment & decrement Operator**

++m	preincrement by one
m++	Post increment by one
--m	predecrement by one
m--	Post decrement by one

6. **Ternary Operator/Condititional Operator (? :) :-**



7. Address Operator (&) :-

Example:- **&a** Address of variable **a**.

8. Indirection Operator (*) :-

*p it point address of variable p.

9. Size of operator:-

sizeof (variable/expression)

10. Assignment operator:-

=

How to Write And evaluate Expression:-

Example:-

s1 = a + b It means operand
s2 = a/b
s3 = a*b
s4 = a−b
s5 = a%b

Precedence rule in Arithmetic Expression:-

%	Highest priority
/	
*	
−	
+	Lowest priority

Precedence rule in Relation Expression:-

<	Highest priority
<=	
>	
>=	
==	
!=	Lowest priority

Precedence rule in Logical Expression:-

!	Highest priority
&&	
||	Lowest priority

***Example*:-1**

Write 'c' expression of following

a) $A = p(1 + r/100)^n$

b) Area = square root of $s(s-a)(s-b)(s-c)$

Solution:-

a) m^n = pow(m,n)

A = p*pow ((1 + r/100),n)

b) Area = sqrt(s*(s-a)*(s-b)*(s-c))

Example:-2

Write 'c' expression of following

square root of m^n

y = sqrt (pow (m, n))

Example:-3

Evaluate the following 'C' expression.

m = a*b/(1 + 5*2/3 + c) + d*b/7

Where

a = 5
b = 4
c = 6
d = 8

Apply **BEDMAS** Rule

B = Bracket
E = Exponentiation
D = Division
M = Multiply
A = Addition
S = Subtraction

Rules:-

int/int = int
int/float = float
float/int = float
float/float = float

Solution:-

m = a*b/(1 + 5*2/3 + c) + d*b/7
m = 5*4/(1 + 5*2/3 + 6) + 8*4/7
= 5*4/(1 + 0 + 6) + 8*4/7
= 5*4/7 + 0
= 0 + 0
= **0**

***Example*:-4**

Evaluate the following 'C' expression.

m = a*b/(1 + 5*7/3 + c) + d*b/3

Where

a = 5
b = 4
c = 6
d = 8

Solution:-

m = a*b/(1 + 5*7/3 + c) + d*b/3
m = 5*4/(1 + 5*7/3 + 6) + 8*4/3
m = 5*4/(1 + 5*2 + 6) + 8*1
m = 5*4/(17) + 8
m = 0 + 8
m = **8**

***Example*:-5**

Evaluate the following 'C' expression.

m = a*b/(1 + 5*7/3 + c) + d*b/3

Where

a = 5
b = 4.0
c = 6
d = 8

Solution:-

m = a*b/(1 + 5*7/3 + c) + d*b/3
m = 5*4.0/(1 + 5*7/3 + 6) + 8*4.0/3
m = 5*4.0/(1 + 5*2 + 6) + 8*1.33
m = 5*4.0/(17) + 10.64
m = 5*.23 + 10.64
m = 1.15 + 10.64
m = **11.79**

Evaluation of Increment and Decrement Operator:-

***Example*:-1**

int a = 2, b = 3;
a = ++a + b++;
a = ?
b = ?

Solution:-

a = 3 + 3
a = 6
b = 3 + 1 = 4

Example:-2

int a = 2, b = 3;
a = ++a + ++b;
a = ?
b = ?

Solution:-

a = 3 + 4
a = 7
b = 4

Example:-3

int a = 2, b = 3;
a = a++ + ++b;
a = ?
b = ?

Solution:-

a = 2 + 4 = 6
a = a++ = 6 + 1 = 7
b = **4**

Example:-4

int a = 2, b = 3;
a = ++a + ++b;
b = --a + b--;
a = ?
b = ?

Solution:-

a = ++a + ++b;
a = 3 + 4
a = 7
b = 4
b = --a + b--;
b = 6 + 4
b = 10
b = b-- = 10– 1 = **9**
a = **6**

Evaluation of Bitwise operators:-

Example:-

int a = 5, b = 7;

a = 5	=	00000101
b = 7	=	00000111
a&b	=	00000101=5
a|b	=	00000111=7
a^b	=	00000010=2
~a	=	-6
a<<1	=	00001010=10
b>>2	=	00000011
	=	00000001=1

C: - Formated I/O Functions

a) **Function for Output**:- printf("Format String", list of variables);

b) **Function for Input**:- scanf("Formate specifire", list of address variables);

D: - Formated specifier

%d	integer
%f	float
%ld	Long integer
%lf	double
%s	String
%u	Unsigned decimal
%e	Exponent notation
%o	Unsigned octal integer
%x	Unsigned hexadecimal integer

E: - Scap Sequence

'\n'	new line
'\t'	tab
'\f'	form feed
'\r'	carriage return
'\'	Single quote
''\"	double quote
'\\'	back slash
'\a'	alert beep sound

***Program* 1:-**

```
#include<stdio.h>
main()
{
printf("Welcome in C Programming Language");
}
```

Where

\# → Preprocessor directive, which attach c library to header file

stdio.h → Standard input/output header file

***Program* 2:-**

```
#include<stdio.h>
 main()
{
int a = 6, b = 8, c1, c2;
c1 = a + b;
c2 = a*b;
printf("\n\n value of a = %d\t value of b = %d", a, b);
printf("\n\n Sum of two numbers = %d",c1);
printf("\n\n Product of two numbers = %d",c2);
 }
```

Out put:-

Value of a = 6 Value of b = 8

Sum of two numbers = 14

Product of two numbers = 48

***Program* 3:-** Write a program to find **sum** and **product** of any two numbers**.**

```
# include<stdio.h>
main()
{
 int a,b,c1,c2;
printf("\n Enter Value of first number = ");
scanf("%d",&a);
printf("\n Enter Value of second number = ");
scanf("%d",&b);
c1 = a + b;
c2 = a*b;
printf("\n Sum of two number = %d",c1);
printf("\n Product of two number = %d",c2);
}
```

Out put:-

Enter value of first number = 5

Enter value of second number = 6

Sum of two number = 11

Product of two number = 30

***Program*:-4** Write a program to find **division** of any two numbers.

```
# include<stdio.h>
main()
{
 float a,b,c1;
printf("\n Enter Value of first number = ");
scanf("%f",&a);
printf("\n Enter Value of second number = ");
scanf("%f",&b);
printf("\n Value of a = %6.2f\t Value of b = %6.2f",a,b);
c1 = a / b;
printf("\n Division of two number = %6.2f",c1);
}
```

Out put:-

Enter value of first number = 4.50

Enter value of second number = 2.50

Value of a = 4.50 Value of b = 2.50

Division of two number = 1.80

***Program*:-5** Write a program to find **simple interest.**

```
# include<stdio.h>
main()
{
 float si, p,r;
int n;
printf("\n Enter principal amount = ");
scanf("%f",&p);
printf("\n Enter rate of interest  = ");
scanf("%f",&r);
printf("\n Enter time period  = ");
scanf("%d",&n);
si=(p*n*r)/100;
printf("\n Simple interest = %6.2f",si);
}
```

Out put:-

Enter principal amount = 4.50

Enter rate of interest = 2.50

Enter time period = 3

Simple interest = 0.34

***Program*:-6** Write a program to calculate **amount with compound rate of interest.**

```
# include<stdio.h>
# include<math.h>
main()
{
 float a, p,r;
int n;
printf("\n Enter principal amount = ");
scanf("%f",&p);
printf("\n Enter rate of interest  = ");
scanf("%f",&r);
printf("\n Enter time period  = ");
scanf("%d",&n);
a=p*(pow((1+r/100),n));
printf("\n Amount interest = %6.2f",a);
}
```

Out put:-

Enter principal amount = 6.50

Enter rate of interest = 3.50

Enter time period = 5

Amount interest = 7.72

***Program*:-7** Write a program to convert foreignheight temperature to Celsius temperature.

(f−32)/9 = c/5 **c = 5*(f−32)/9**, f = (9*c)/5 + 32

```
#include<stdio.h>
#include<math.h>
main()
{
float f1,c1;
printf("\n\n\t Enter temperature in foreignheight=");
scanf("%f",&f1);
printf("\n\n\t Temperature in foreignheight=%8.2f",f1);
c1=5.0*(f1−32)/9;
printf("\n\n\t Temperature in Celsius=%8.2f",c1);
}
```

Out put:-

Enter temperature in Foreignheight = 37.50

Temperature in foreignheight = 37.50

Temperature in Celsius = **3.06**

***Program*:-8** Write a program to calculate area of circle, triangle & volume of sphere.

```
#include<stdio.h>
#include<math.h>
main()
{
float r,a,b,c,s,cir_area,sph_volume,tri_area;
printf("\n\n\t Enter radius for circle & sphere=");
scanf("%f",&r);
printf("\n\n\t Enter first side of triangle=");
scanf("%f",&a);
printf("\n\n\t Enter second side of triangle=");
scanf("%f",&b);
printf("\n\n\t Enter third side of triangle=");
scanf("%f",&c);
cir_area=3.14*pow(r,2);
sph_volume=4.0/3.0*pow(r,3);
s=(a+b+c)/2;
tri_area=sqrt(s*(s-a)*(s-b)*(s-c));
printf("\n\n\t Area of circle=%4.3f",cir_area);
printf("\n\n\t Volume of sphere=%6.3f",sph_volume);
printf("\n\n\t Area of triangle=%1.5f",tri_area);
}
```

Out put:-

Enter radius for circle & sphere = 3.50

Enter first side of triangle = 2.30

Enter second side of triangle = 2.60

Enter third side of triangle = 2.90

Area of circle = 38.465

Volume of sphere = 57.167

Area of triangle = 2.84816

***Program*:-9** Write a program for **swapping** any two number.

```
#include<stdio.h>
#include<math.h>
main()
{
float a,b,t;
printf("\n\n\t Enter first number=");
scanf("%f",&a);
printf("\n\n\t Enter second number=");
scanf("%f",&b);
printf("\n\n\t First number=%6.2f",a);
printf("\n\n\t Second number=%6.2f",b);
t = a;
a = b;
b = t;
printf("\n\n\t After swapping first number=%6.2f",a);
printf("\n\n\t After swapping second number=%6.2f",b);
}
```

Out put:-

Enter first number = 12.30

Enter second number = 23.50

First number = 12.30

Second number = 23.50

After swapping first number = 23.50

After swapping second number = 12.30

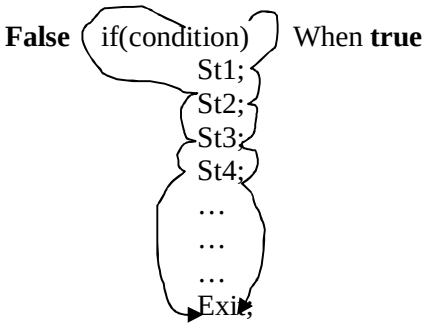
3. Control Statement

'C' language provides facilities for controlling the order of execution of the statements, which is referred to as flow control statements/control statements. There are following **three categories** of flow control statements.

1. **Decision Control Statements**
   a) if statement;
   b) if-else statement
   c) nested if-else statement
   d) else-if construct statement
   e) switch case statement
2. **Looping Control Statement**
   a) while loop
   b) do-while loop
   c) for loop
3. **Jumping Control Statement**
   a) goto
   b) break
   c) continue

1. Decision Control Statement

(a) if statement ⇨ Syntax



Example:-

```
#include<stdio.h>
#include<conio.h>
main()
{
int n;
printf("\n Enter any number=");
scanf("%d",&n);
if(n>=5)
printf("\n\t Amit");
printf("\n\t Kumar");
printf("\n\t Raj");
printf("\n\t Ye One");
printf("\n\t Thank You");
}
```

Out put:-

```
Enter any number = 7
Amit
Kumar
Raj
Ye one
Thank you
```

***Program*:-1** Write a program to check number is **even** or **odd**.

```
#include<stdio.h>
main()
{
int n;
printf("\n Enter any number=");
scanf("%d",&n);
if(n%2==0)
printf("\n Number is Even=%d",n);
if(n%2!=0)
printf("\n Number is Odd=%d",n);
}
```

Out put:-

Enter any number = 56
Number is **even** = 56

Or

Enter any number = 59
Number is **odd** = 59

***Program* :-2** Write a program to check the year is **leap year** or **not leap year**.

```
#include<stdio.h>
main()
{
int year;
printf("\nEnter any year=");
scanf("%d",&year);
if(year%4==0)
printf("\nLeap Year=%d",year);
if(year%4!=0)
printf("\nNot Leap Year=%d",year);
}
```

Out put:-

Enter any year = 2013
Not leap year = 2013

Or

Enter any year = 2008
Leap year = 2008

***Program* :-3** Write a program to checking **profit** or **loss** or **no profit** or **no loss**.

```
#include<stdio.h>
main()
{
float sale,purchase,m;
printf("\n Enter Purchase cost=");
scanf("%f",& purchase);
printf("\n Enter sale cost=");
scanf("%f",&sale);
m= sale- purchase;
if(m==0)
printf("\n Neither Profit Nor Loss");
if(m>0)
printf("\n Profit=%6.2f",m);
if(m<0)
printf("\n Loss=%6.2f",m);
}
```

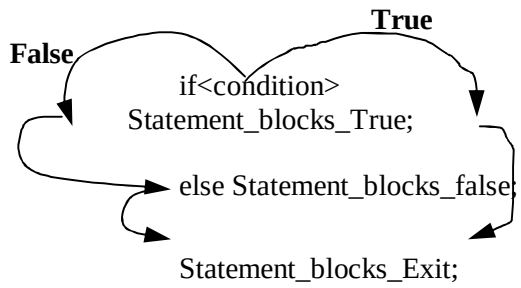
Out put:-

Enter purchase cost = 200.00
Enter sale cost = 190.00
Loss = – 10.00

Or

Enter purchase cost = 200.00
Enter sale cost = 350.00
Profit = 150.00

(b) **if-else statement** ⇨ Syntax



Example:-

```
#include<stdio.h>
main()
{
int n;
printf("\n Enter value of n=");
scanf("%d",&n);
if(n>=10)
printf("A");
else
printf("B");
printf("C");
printf("D");
}
```

Out put:-

Enter value of n = 12
ACB

Or

Enter value of n = 8
BCD

Or

```
#include<stdio.h>
main()
{
int n;
printf("\n Enter value of n=");
scanf("%d",&n);
if(n>=10)
{
printf("A");
printf("B");
printf("C");
}
else
{
printf("D");
}
}
```

Out put:-

Enter value of n = 13
ABC

Or

Enter value of n = 9
D

Or

```
#include<stdio.h>
main()
{
int n,a=5,b=4,c1,c2,c3;
printf("\n Enter value of n=");
scanf("%d",&n);
if(n>=10)
{
c1=a+b;
c2=a*b;
printf("\n\t value of c1=%d",c1);
```

Out put:-

Enter value of n = 15
Value of c1 = 9
Value of c2 = 20

Or

Enter value of n = 7
Value of c3 = 1

```
        printf("\n\t value of c2=%d",c2);
        }
else
        {
        c3=a-b;
        printf("\n\t value of c3=%d",c3);
        }
        }
```

***Program* :-4** Write a program to check year is **leap year** or **not leap year** using **if else-statement**.

```
        #include<stdio.h>
        main()
        {
        int year;
        printf("\n\t Enter any year=");
        scanf("%d",&year);
        if(year%4==0)
        printf("\n\t Leap Year=%d",year);
else
        printf("\n\t Not Leap Year=%d",year);
        printf("\n\t Thank You");
        }
```

Out put:-

Enter any year = 2015
Not leap year = 2015
Thank you

Or

Enter any year = 2020
Leap year = 2020
Thank you

(c) nested if-else statement ⇨

Syntax :-

```
        if<condition1>
        Statement_blocks1;
  else
        if<condition2>
        Statement_blocks2;
  else
        if<condition3>
        Statement_blocks3;
   else
        if<condition4>
        Statement_blocks4;
        …
        …
   else
        Statement_blocks_Exit;
```

Program 5:- Write a program to check largest of any three number.

```
        #include<stdio.h>
        main()
        {
        int a,b,c;
        printf("\n Enter First number=");
        scanf("%d",&a);
        printf("\n Enter Second number=");
        scanf("%d",&b);
        printf("\n Enter Third number=");
        scanf("%d",&c);
```

```
printf("\n\n First Number=%d\t Second Number=%d\t Third Number=%d", a,b,c);
        if(a==b && b==c )
        printf("\n All Numbers are equal");
else
        if(a>b && b>c|| a>c && c>b )
        printf("\n A is the largest numbers");
else
        if(b>c && c>a|| b>a && a>c )
        printf("\n B is the largest numbers");
else
        printf("\n C is the largest nubers");
        }
```

Out put:-

Enter first number = 88
Enter second number = 55
Enter third number = 76
First number = 88 Second number = 55 Third number = 76
A is the largest number

Program 6:- Write a program to solve any quadratic equation.

```
#include<stdio.h>
#include<math.h>
main()
{
float a,b,c,x1,x2,d;
printf("\n Enter Coefficient of x^2=");
scanf("%f",&a);
printf("\n Enter Coefficient of x=");
scanf("%f",&b);
printf("\n Enter Constant Value=");
scanf("%f",&c);
printf("\n\n%3.2fx^2 +%3.2fx+%3.2f=0",a,b,c);
d=b*b-4*a*c;
if(d==0)
{
printf("\n Roots are real and equal");
x1=-b/(2*a);
x2=-b/(2*a);
printf("\n First & Second Roots=%3.2f,x1,x2);
}
else
if(d>0)
{
printf("\n Roots are real and unequal=");
x1=(-b+sqrt(d))/(2*a);
x2=(-b-sqrt(d))/(2*a);
printf("\n First Root=%3.2f\n Second root=%3.2f",x1,x2);
}
else
printf("\n Roots are Imaginary ");
}
```

Out put:-

Enter coefficient of x^2 = 1.0
Enter coefficient of x = −8.0
Enter constant value = 16.0

1.00x^2 − 8.00x + 16.00 = 0
Roots are real & equal
First & Second Roots = 4.00

Or

Enter coefficient of x^2 = 1.0
Enter coefficient of x = −10.0
Enter constant value = 9.0

1.00x^2 − 10.00x + 9.00 = 0
Roots are real & unequal
First Roots = 9.00
Second Roots = 1.00

Or

Enter coefficient of x^2 = 9.0
Enter coefficient of x = 6.0
Enter constant value = 5.0

9.00x^2 + 6.00x + 5.00 = 0
Roots are Imaginary

Program 7:- Write a program to check and display grading of any student.

```
#include<stdio.h>
#include<math.h>
main()
{
int h,e,m,p,c,tot;
float per;
printf("\n Enter Marks Obtained In Hindi=");
scanf("%d",&h);
printf("\n Enter Marks Obtained In English=");
scanf("%d",&e);
printf("\n Enter Marks Obtained In Maths=");
scanf("%d",&m);
printf("\n Enter Marks Obtained In Physics=");
scanf("%d",&p);
printf("\n Enter Marks Obtained In Chemistry=");
scanf("%d",&c);
tot=(h+e+m+p+c);
per=tot/5;
printf("\n----------\n");
printf("\n Hindi Marks = %d\n English Marks  = %d\n Maths Marks = %d",h,e,m);
printf("\n Physic Marks = %d\n Chemistry Marks = %d",p,c);
printf("\n\n Total Marks = %d\n Percentage = %6.2f",tot,per);
printf("\n\n----------\n");
if(per>=75 &&per<=100 && h>=33 && e>=33 && m>=33 && p>=33 &&c>=33)
printf("\n\n A Grade & Passed");
else
if(per>=65 &&per<=75 && h>=33 && e>=33 && m>=33 && p>=33 &&c>=33)
printf("\n\n B Grade & Passed");
else
if(per>=45 &&per<=65 && h>=33 && e>=33 && m>=33 && p>=33 &&c>=33)
printf("\n\n C Grade & Passed");
else
if(per>=33 &&per<=45 && h>=33 && e>=33 && m>=33 && p>=33 &&c>=33)
printf("\n\n D Grade & Passed");
else
printf("\n\n E Grade & Failed");
}
```

Out put:-

```
Enter marks obtained in Hindi = 85
Enter marks obtained in English = 75
Enter marks obtained in Maths = 65
Enter marks obtained in Physics = 64
Enter marks obtained in Chemistry = 97
……
Hindi marks = 85
English marks = 75
Maths marks = 65
Physics marks = 64
Chemistry marks = 97

Total marks = 386
Percentage = 77.00
……..
A Grade & Passed
```

(d) **else-if-Construct Statement/Ladder statement** ⇨

Syntax :-

```
…
…
…
else
if<condition>
        Statement_Blocks
        …
        …
        …
```

It is used for solving choice based problem.

Program:-

Example 1. Write a program for solving arithmetical operators, year and number checking problems.

```
#include<stdio.h>
#include<math.h>
main()
{
int ch,n,year;
float a,b,s1,p1,d1,sub;
printf("\n\n\n\t\t-----------------------------------------");
printf("\n\t\tChoice Based Arithmetic,Year & Number Checking=");
printf("\n\t\t-----------------------------------------");
printf("\n\t\t\t1:-Addition");
printf("\n\t\t\t2:-Subtraction");
printf("\n\t\t\t3:-Product");
printf("\n\t\t\t4:-Division");
printf("\n\t\t\t5:-Year Checking");
printf("\n\t\t\t6:-Number Checking");
printf("\n\t\t-----------------------------------------");
printf("\n\t\t-----------------------------------------");
printf("\n\t\t Enter Your Choice=");
scanf("%d",&ch);
printf("\n\t\t-----------------------------------------");
printf("\n\t\t Enter Any Two Numbers For Arithmetic Operators=");
scanf("%f%f",&a,&b);
printf("\n\t\t-----------------------------------------");
printf("\n\t\t-----------------------------------------");
if(ch==1)
{
s1=a+b;
printf("\n\t\t Sum=%6.2f",s1);
}
else
if(ch==2)
{
sub=a-b;
printf("\n\t\t Difference=%6.2f",sub);
}
else
if(ch==3)
{
p1=a*b;
printf("\n\t\t Product=%6.2f",p1);
}
else
if(ch==4)
{
d1=a/b;
printf("\n\t\t Division=%6.2f",d1);
}
else
if(ch==5)
```

Out put:-

```
……..
Choice Based Arithmetic, year & Number checking=
……..
        1:- Addition
        2:- Subtraction
        3:- Product
        4:- Division
        5:- Year checking
        6:- Number checking
……..
……..
Enter your choice =1
……..
Enter any two numbers for arithmetic operators = 2.0
                                                 3.0
……...
……
Sum = 5.00
```

```
        {
        printf("\n\t\t Enter Year=");
        scanf("%d",&year);
        if(year%4==0)
        printf("\n\t\t Leap Year=%d",year);
else
        printf("\n\t\t Not Leap Year=%d",year);
        }
else
        if(ch==6)
        {
        printf("\n\t\t Enter Number=");
        scanf("%d",&n);
        if(n%2==0)
        printf("\n\t\t Even Number=%d",n);
else
        printf("\n\t\t Odd Number =%d",n);
        }
else
        printf("\n\t\t Wrong Choice Again Enter...!");
        }
```

(e) **Switch-case Statement** ⇨

It is also used for solving choice based problems.

Syntax:-

```
        switch(expression)
        {
 case <value1>:
        Statement_Blocks;
        break;
case <value2>:
        Statement_Blocks;
        break;
case <value3>:
        Statement_Blocks;
        break;
        …
        …
        …
default:
        Statement_Blocks_False;
        }
        Exit_Statement;
```

Program:-

*Example*1. Write a program for solving arithmetical operators, year and number checking problems **by using switch-case statement**.

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

```
        {
        int n, year, ch;
        float a,b,s1,p1,d1,sub;
        printf("\n\n\n\t\t-----------------------------------------------");
        printf("\n\t\t Choice Based Arithmetic, Year & Number Checking=");
        printf("\n\t\t-----------------------------------------------");
        printf("\n\t\t\t1:-Addition");
        printf("\n\t\t\t2:-Subtraction");
        printf("\n\t\t\t3:-Product");
        printf("\n\t\t\t4:-Division");
        printf("\n\t\t\t5:-Year Checking");
        printf("\n\t\t\t6:-Number Checking");
        printf("\n\t\t-----------------------------------------------");
        printf("\n\t\t-----------------------------------------------");
        printf("\n\t\t Enter Your Choice=");
        scanf("%d",&ch);
        printf("\n\t\t-----------------------------------------------");
        printf("\n\t\t Enter Any Two Numbers For Arithmetic Operators=");
        scanf("%f\t %f",&a,&b);
        printf("\n\t\t-----------------------------------------------");
        printf("\n\t\t-----------------------------------------------");
switch (ch)
        {
case 1:
        {
        s1=a+b;
        printf("\n\t\t Sum=%6.2f",s1);
        }
        break;
case 2:
        {
        sub=a-b;
        printf("\n\t\t Difference=%6.2f",sub);
        }
        break;
case 3:
        {
        p1=a*b;
        printf("\n\t\t Product=%6.2f",p1);
        }
        break;
 case 4:
        {
        d1= a/b;
        printf("\n\t\t Division=%6.2f",a/b);
        }
        break;
case 5:
        {
        printf("\n\t\t Enter Year=");
        scanf("%d",&year);
        if(year%4==0)
        printf("\n\t\t Leap Year=%d",year);
        if(year%4!=0)
        printf("\n Not Leap Year=%d",year);
        }
```

Out put:-

```
……..
Choice Based Arithmetic, year & Number checking=
……..
        1:- Addition
        2:- Subtraction
        3:- Product
        4:- Division
        5:- Year checking
        6:- Number checking
……..
……..
Enter your choice =5
……..
Enter any two numbers for arithmetic operators = 2.0
                                                 8.0
……...
……
Enter Year = 2020

Leap Year = 2020
```

```
        break;
case 6:
        {
        printf("\n\t\t Enter any number=");
        scanf("%d",&n);
        if(n%2==0)
        printf("\n\t\t Even Number=%d",n);
        if(n%2!=0)
        printf("\n Odd Number=%d ",n);
        }
        break;
        default:
        printf("\n\t\t Thanks");
        }
        }
```

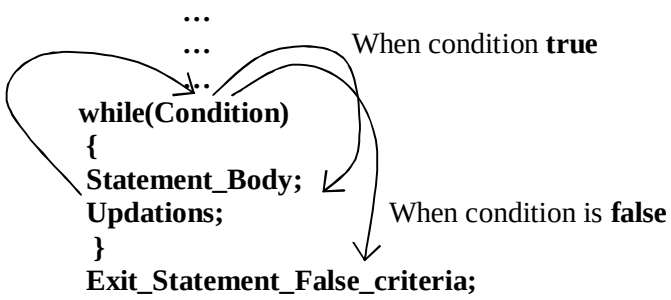
2. **Looping Control** Statement/**Iteration** Statements/**Repetition** Statement

It allows the execution of some set of statements repeatedly till either for a known number of times or till certain conditions is met. There are following three types of looping statements.

a) while loop
b) do-while loop
c) for loop

(a) **while loop** ⇨ **It executes looping body when condition is true.**

Syntax



Program:-

Example 1. Write a program to generate series of **natural number** and **also find there sum**.

```
#include<stdio.h>
main()
{
int n,i=1,s=0;
printf("\n\ Enter value of n=");
scanf("%d",&n);
printf("\n\ Series of Natural Number=");
while(i<=n)
{
printf("%d\t",i);
i++;
s=s+i;
```

Out put:-

Series of natural number =

Enter value of n = 10

1 2 3 4 5 6 7 8 9 10

False value = 11 **Sum** of natural number = 65

```
}
printf("False value=%d\t Sum of Natural Number=%d",i,s);
}
```

Example 2. Write a program to Reserve generate series of **natural number** and **also find there sum**.

```
#include<stdio.h>
main()
{
int n, i=1, s=0;
printf("\n\ Enter value of n=");
scanf("%d",&n);
printf("\n\ Reserve Series of Natural Number=\n");
while(n>=i)
{
printf("%d\t",n);
n--;
s = s + n;
}
printf("False value=%d\n",i);
printf("Sum of natural number=%d",s);
}
```

Out put:-

Enter value of n = 10

Reverse series of natural number =
10 9 8 7 6 5
4 3 2 1
False value = 1
Sum of natural number = 45

Example 3. Write a program to generate series of **odd** and **even number.**

```
#include<stdio.h>
main()
{
int n,i=1,j=2;
printf("\n\n Enter Value of n=");
scanf("%d",&n);
printf("\n Series of Odd Numbers=\t");
while(i<=n)
{
printf("%d\t",i);
i=i+2;
}
printf("\n Series of Even Numbers=\t");
while(j<=n)
{
printf("%d\t",j);
j=j+2;
}
printf("\n Thank You ");
}
```

Out put:-

Enter value of n = 19

Series of odd number = 1 3 5 7
9 11 13 15 17 19

Series of even number = 2 4 6 8
10 12 14 16 18

Example 4. Write a program to **generate series** of **Fibonacci number.**

```
#include<stdio.h>
main()
{
int n,i=1,a=0,b=1,s=0;
printf("\n\t Enter Value of n=");
scanf("%d",&n);
printf("\n\t Series of Fibonacci Number=\n");
while(i<=n)
{
printf("%d\t",s);
a=b;
b=s;
s=a+b;
i++;
}
printf("\n False value of i=%d",i);
printf("\n\t Thank You Amit");
}
```

Out put:-

Enter value of n = 10
Series of Fibonacci number =
0 1 1 2 3 5 8 13 21 34

False value of **i** = 11
Thank you Amit

Example 5. Write a program to **generate series** of **Factorial number.**

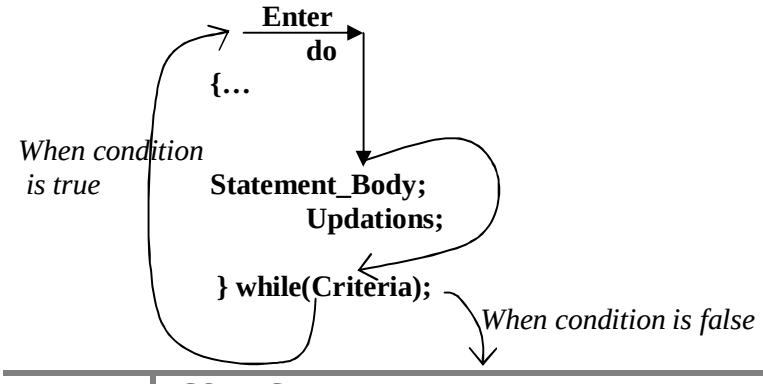
```
#include<stdio.h>
main()
{
long int n,fact=1;
int i=1;
printf("\n Enter Value of n=");
scanf("%ld",&n);
printf("\n Series of Factorial Number=\n");
while(i<=n)
{
fact=fact*i;
printf("%d = %ld\t\n",i,fact);
i=i+1;
}
printf("\n Thank You Amit");
}
```

Out put:-

Enter value of **n** = **9**
Series of Factorial Number =
1 = 1
2 = 2
3 = 6
4 = 24
5 = 120
6 = 720
7 = 5040
8 = 40320
9 = 362880
Thank You Amit

(b) do-**while loop** ⇨ It executes looping body one time when condition is false. And further execute when condition is true.

Syntax



Exit_Statement_False_criteria;

Example:-

```
#include<stdio.h>
main()
{
int i=1,n;
printf("\n\t Enter Value of n=");
scanf("%d",&n);
do
{
printf("%d\t",i);
i =i+1;
}while(i<=n);
printf("n Thank");
}
```

Example 1. Write a program to generate series of **ASCII Code**.

```
#include<stdio.h>
main()
{
int i=0,n=255;
char ch;
printf("\n\t ASCII Characters & Codes=\n");
do
{
printf("%d=%c\t",i,i);
i=i+1;
}while(i<=n);
printf("\n\t Thanks");
}
```

Example 2. Write a program to generate series of **Fibonacci** using **do-while loop**.

```
#include<stdio.h>
main()
{
int i=0,a=0,b=1 ,s=0, n;
char ch;
printf("\n\t Enter Value of n=");
scanf("%d",&n);
printf("\n Series of Fibonacci");
do
{
printf("%d\t",s);
a=b;
b=s;
s=a+b;
i=i+1;
}while(i<=n);
printf("\n\t Thanks");
}
```

Out put:-

Enter value of n = 10

Series of Fibonacci

0 1 1 2 3

5 8 13 21 34

55

Thanks

Example 1. Write a program to generate series of **Armstrong Number**.

1 153 370 371 407 . . .

Logic:-

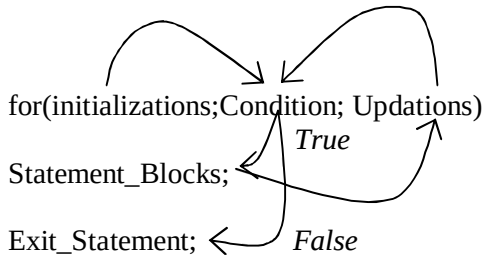
$1^3=1$
$153=1^3+5^3+3^3=153$
$370=3^3+7^3+0^3=370$
$371=3^3+7^3+1^3=371$
$407=4^3+0^3+7^3=407$
…
…
…

```
#include<stdio.h>
main()
{
int n=0,m=0,d=0,s=0,a=1,k;
printf("\n\n Enter Value of k=");
scanf("%d",&k);
while(a<=k)
{
n=a;
s=0;
while(n>0)
{
m=n%10;
n=n/10;
s=s+m*m*m;
}
if(s==a)
printf("Series of Armstrong Numbers=%d\n",a);
a++;
}
}
```

Out put:-

Enter value of **k** = **500**
Series of Armstrong numbers = 1
Series of Armstrong numbers = 153
Series of Armstrong numbers = 370
Series of Armstrong numbers = 371
Series of **Armstrong** numbers = 407

(c) for **loop** ⇨ **Syntax**



Initializations :-There are many initializations by using commas.
Updations :-There are many Updations by using commas.
Condition :-Only one Condition will be defined.
Semicolon :-There are two semicolons must be inside for loop.

Program:-

Example 1. Write a program to generate series of **odd number** using **for loop**.

```
#include<stdio.h>
main()
{
int i,n;
printf("\n Enter Value of n=");
scanf("%d",&n);
printf("\n Series of odd number\n");
for(i=1;i<=n;i=i+2)
printf("%d\t",i);
}
```

Out put:-

Enter value of n = 20

Series of odd number=

1	3	5	7	9
11	13	15	17	19

Or

```
#include<stdio.h>
main()
{
int i=1,n;
printf("\n Enter Value of n=");
scanf("%d",&n);
for(;i<=n;)
{
printf("%d\t",i);
i=i+2;
}
}
```

Or

```
#include<stdio.h>
main()
{
for(;;)
}
```

⇨Program is correct and no any syntactical errors but execution of program becomes infinite loop.

Example 2.

```
#include<stdio.h>
main()
{
int i,j,n;
printf("\n Enter Value of n=");
scanf("%d",&n);
for(i=1; i<=n; i++)
{
for(j=i; j<=n; j++)
printf("%d",j);
printf("\n ");
}
for(i=1; i<=n; i++)
{
for(j=i; j<=n; j++)
printf("*",j);
printf("\n ");
}
}
```

Program Based **On Pattern**

1. Rectangle shape

```
#include<stdio.h>
main()
{
int i,j,n;
printf("\n\t Enter Value of n=");
scanf("%d",&n);
for(i=1; i<=n; i++)
{
for(j=1; j<=n-i; j++)
printf(" ");
for(j=1; j<=i; j++)
printf("*");
printf("\n");
}
}
```

Out put:-

```
Enter Value of n = 10
                                        *
                                    *   *
                                *   *   *
                            *   *   *   *
                        *   *   *   *   *
                    *   *   *   *   *   *
                *   *   *   *   *   *   *
            *   *   *   *   *   *   *   *
        *   *   *   *   *   *   *   *   *
```

2. Triangle shape

```
#include<stdio.h>
main()
{
int i,j,n;
printf("\n\t Enter Value of n=");
scanf("%d",&n);
for(i=1; i<=n; i++)
{
for(j=1; j<=n-i; j++)
printf(" ");
for(j=1; j<=i; j++)
printf("*");
for(j=1; j<i; j++)
printf("*");
printf("\n");
}
}
```

Out put:-

```
Enter Value of n = 10
          *
        *   *
      *   *   *
    *   *   *   *
  *   *   *   *   *
*   *   *   *   *   *
```

3. Inverted Pyramid shape

```
#include<stdio.h>
main()
{
int i,j,n;
printf("\n\t Enter Value of n=");
scanf("%d",&n);
for(i=1; i<=n; i++)
{
for(j=1; j<=i; j++)
printf(" ");
for(j=1; j<=n-i; j++)
printf("*");
for(j=1; j<(n-i); j++)
printf("*");
```

Out put:-

```
Enter Value of n = 10
* * * * * * * * * *
 * * * * * * * * *
  * * * * * * * *
   * * * * * * *
    * * * * * *
     * * * * *
      * * * *
       * * *
        * *
         *
```

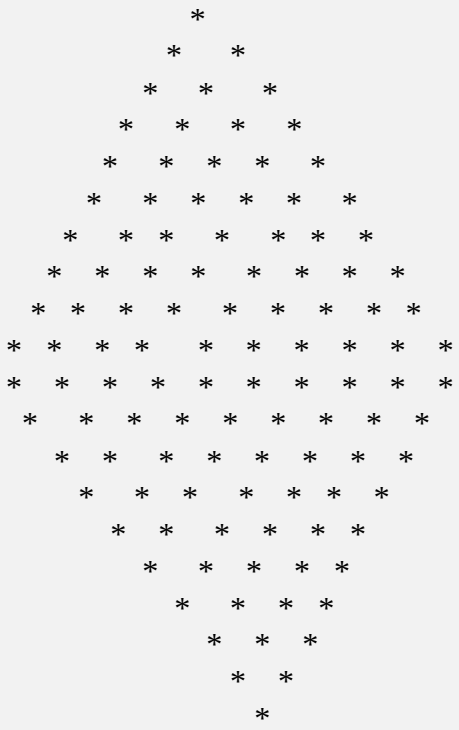
```
printf("\n");
}
}
```

4. Diamond shape

```
#include<stdio.h>
main()
{
int i,j,n;
printf("\n\t Enter Value of n=");
scanf("%d",&n);
for(i=1; i<=n; i++)
{
for(j=1; j<=n-i; j++)
printf(" ");
for(j=1; j<=i; j++)
printf("*");
for(j=1; j<i; j++)
printf("*");
printf("\n");
}
for(i=1; i<=n-1; i++)
{
for(j=1; j<=i; j++)
printf(" ");
for(j=1; j<=n-i; j++)
printf("*");
for(j=1; j<n-i; j++)
printf("*");
printf("\n");
}
}
```

Out put:-

Enter Value of n = 10



Example:- Write a program to generate a series of **prime number.**

```
#include<stdio.h>
main()
{
int i,j,n,f;
printf("\n\t Enter Value of n=");
scanf("%d",&n);
printf("\n Series of Prime Number\n\n");
for(i=2; i<n; i++)
{
for(f=0,  j=2; j<i; j++)
{
if(i%j==0)
{
f=1;
break;
}
}
if(f==0)
{
printf("\t%d",i);
}
}
}
```

Out put:-

Enter Value of n = 10

Series of Prime Number

2 3 5 7

3. Jumping Control Statement

Jump Statement *makes the control jump to another section of the program unconditionally when encountered. It is usually used to terminate the* ***loop*** *or* ***switch-case*** *instantly. It is also used to escape the execution of a section of the program.*

(a) goto statement ⇨

Syntax: - goto<lable>;

Syntax for define label:-

<labe l_name>:

Example→

```
#include<stdio.h>
main()
{
int
c1;
printf("\n I am Amit1");
goto c4;
c2:
printf("\n I am Amit2");
c3:
printf("\n I am Amit3");
c4:
printf("\n I am Amit4");
goto c8;
c5:
printf("\n I am Amit5");
c6:
printf("\n I am Amit6");
c7:
printf("\n I am Amit7");
c8:
printf("\nExist_Statement");
}
```

Out put:-

I am Amit 1
I am Amit 4
Exist_statement

Example:-1. Write a program to generate **series of natural number** without using looping.

```
#include<stdio.h>
main()
{
int i=1, n, s=0;
printf("\n Enter value of n=");
scanf("%d",&n);
printf("Series of Natural Number=\n");
c1:
printf("%d\t",i);
c2:
if(i<n)
{
```

Out put:-

Enter value of n = 11
Series of Natural Number =
1 2 3 4
5 6 7 8
9 10 11
Sum = 65
Thank You….!

```
i=i+1;
s=s+i;
goto c1;
}
printf("\n sum=%d\n Thank You----!",s);
}
```

Example:-2. Write a program to generate series of **Fibonacci number** without using loop.

```
#include<stdio.h>
main()
{
int i=1, n, s=0,a=0,b=1;
printf("\n Enter value of n=");
scanf("%d",&n);
printf("Series of Fibonacci Number=\n");
c1:
printf("%d\t",s);
c2:
if(i<n)
{
a=b;
b=s;
s=a+b;
i=i+1;
goto c1;
}
printf("\n Thank You----!",s);
}
```

Out put:-

Enter value of n = 13
Series of Fibonacci number =
0 1 1 2 3 5
8 13 21 34 55 89
144
Thank You….!

Example:-3. Write a program for **arithmetic** operation and execution of operation depend upon goto statement.

```
#include<stdio.h>
main()
{
float a, b, c1, c2, c3;
printf("\n Enter value of a & b=");
scanf("%f%f",&a, &b);
m1:
{
c1 = a + b;
printf("\n sum=%3.2f",c1);
goto m4;
}
m2:
{
c2=a*b;
printf("\n product=%f",c2);
}
m3:
{
c3 = a/b;
printf("\n division=%f",c3);
}
m4:
```

Out put:-

Enter value of a & b = 5
6
Sum = 11.00
Thank You……!

```
printf("\n Thank You----!");
}
```

(b) break statement ⇨ break statement **used only** in **switch case control** and **looping control statement**. It exist execution of looping body statement. When we want to no further execution.

Syntax: - break;

Example:-1. Write a C program to **check input number** is **prime** or **not**.

```
#include<stdio.h>
main()
{
int n,i;
printf("\n Enter any number=");
scanf("%d",&n);
for(i=2; i<n; i++)
{
if(n%i==0)
{
printf("\n not prime");
break;
}
}
if(i==n)
printf("\n Prime Number");
}
```

Out put:-

Enter any number = 59

Prime Number

Example:-2. Write a program to **change day_code** into **day**.

```
#include<stdio.h>
main()
{
int day_code;
printf("\n\t\t----------------------------------------=\n");
printf("\n\t\t conversion of day code into day");
printf("\n\t\t----------------------------------------=\n");
printf("\n\t\t Enter day_code=");
scanf("%d",&day_code);
switch(day_code)
{
case 1:
printf("\n\t Sunday");
break;
case 2:
printf("\n\t Monday");
break;
case 3:
printf("\n\t Tuesday");
break;
case 4:
printf("\n\t Wednesday");
break;
case 5:
printf("\n\t Thursday");
break;
case 6:
```

Out put:-

…….

Conversion of day code into day

…….

Enter day_code = 6

Friday

```
printf("\n\t Friday");
break;
case 7:
printf("\n\t Saturday");
break;
default:
printf("\n\t Wrong day code");
}
}
```

(c) continue statement ⇨ This control statement skips the remainder of the current iteration and initiator the execution of the next iteration/repetion. When this statement is encounter in a loop, the rest of the statements loop are skipped and control passes to the condition, which is evaluated and if true the loop is entered again.

Syntax: - continue;

Example:-1. Write a program to generate **series** of **odd number** with the help of **continue**.

```
#include<stdio.h>
main()
{
int i,n;
printf("\n Enter value of n=");
scanf("%d",&n);
printf("\n Series of odd number=");
for(i=1; i<=n; i++)
{
if(i%2==0)
continue;
printf("\n\n\t exist statement=%d",i);
}
}
```

Out put:-

Enter value of n = 12

Series of odd number =
exist statement = 1
exist statement = 3
exist statement = 5
exist statement = 7
exist statement = 9
exist statement = 11

4. Function

It is **subprogram** which is used for performing some well defined specific task. **Function may or may not consist of arguments**. Arguments enclosed within **parenthesis**.

Or

A function is a **set of program** statements that can be processed independently. A function can be invoked which behaves as though its code is inserted at the point of the function call. The **communication** between **caller** (Calling function) and **callee** (called function) takes place through **parameter**.

Example:-

f(x)	Function with single argument/Parameter x.
f()	Function with no argument/Parameter x.
f(x1,x2,x3,…)	Function with multiple arguments/Parameters.

Advantage of Function ⇨

1. Modular programming.
2. Reduction in the amount of work and development time.
3. Program and function debugging is easier.
4. Reduction in the size of program due to code reusability
5. Library of functions can be implemented by combining well designed, tested and proven functions.

Types of functions ⇨

1. Built In function/System defined functions
2. User Defined Functions

- **a)** A function without argument and no return value.
- **b)** A function without argument and return value.
- **c)** A function with argument and no return value.
- **d)** A function with argument and return value.
- **e)** A function call by Value and call by reference.
- **f)** Recursive function/Calling itself function.

1. Built In function/System defined functions:-

Example:-

pow(m,n)	m^n	
log(m)	m>0	
sqrt(x)	x>=0	
ln(x)	x>0	Natural log to the base 2<e<3
strlen(string)	For measuring length of string	
strrev(string)	For reversing of string	
gets(string)	Accept a string from standard input device	
puts(string)	This function outputs a string **constant** or **variable** to the standard output device.	
getchar()	It return a character that has been recently typed.	

getche() It also return a character that has been recently typed. The typed character is echoed to the computer screen.

getch() This function too returns a character that has been recently typed. But neither the user is required to type enter key after entering character nor the typed character echoed to the computer screen.

putchar() This function output a character **constant** or **variable** to the standard output device.

printf("format of string",list of variables) **Console output**

scanf("format specifier",list of address variables) **Console input**

main()

clrscr()

etc.

Components of function:-

1. Function declaration or **prototype**.
2. Function parameter(Formal Parameter)
3. Combination of function declaration and its definition
4. Function definition(function declarator and function body)
5. Return statement.
6. Function call.

2. User Defined Functions:-

a) A function **without argument** and **no return value**.

***Example* 1.**

```
#include<stdio.h>
main()
{
sum();
}
sum()                                    //
{
int a, b, s;
printf("\n Enter value of a & b=");
scanf("%d%d",&a, &b);
s = a + b;
printf("\n Sum of two number=%d",s);
}
```

Out put:-

Enter value of a & b = 5
9
Sum of two number = 14

***Example* 2. ×**

```
#include<stdio.h>
main()
{
div();
check_year();
check_num();
}
div()
```

```
{
float a, b, c;
printf("\n Enter any two digits=");
scanf("%f%f",&a, &b);
c = a/b;
printf("\n Division of any two digits=%7.3f",c);
}
check_year()
{
int year;
printf("\n Enter any year=");
scanf("%d",&year);
if(year%4==0)
printf("\n Leap year");
else
printf("\n Not leap year");
}
ckeck_num()
{
int n;
printf("\n Enter any number=");
scanf("%d",&n);
if(n%2==0)
printf("\n Even number");
else
printf("\n Odd number");
}
```

b) A function **without argument** and **return value**.

Example 1.

```
#include<stdio.h>
main()
{
int m, n;
m = check_year();
if(m==1)
printf("\n Leap year");
else
printf("\n Not leap year");
n=cal();
printf("\n Product of two digits=%d",n);
}
check_year()
{
int year;
printf("\n Enter any year=");
scanf("%d",&year);
if(year%4==0)
return 1;
else
return 0;
}
cal()
{
```

Out put:-

Enter any year = 2015
Not leap year
Enter any two digits = 2
3
Product of two digits = 6

//

```
int a, b, c;
printf("\n Enter any two digits=");
scanf("%d%d",&a, &b);
c=a*b;
return c;
}
```

c) A function **with argument** and **no return value**.

Example 1.

```
#include<stdio.h>
main()
{
float s, p;
printf("\n Enter purchase amount");
scanf("%f",&p);
printf("\n Enter sale amount");
scanf("%f", &s);
pro_loss(p,s);                    //Actual Argument
}
pro_loss(float p1, float s1)      //Formal Argument        //
{
float m;
m=s1-p1;
if(m==0)
printf("\n\n Neither profit nor loss");
if(m>0)
printf("\n\n Profit=%7.2f",m);
if(m<0)
printf("\n\n Loss=%7.2f",m);
}
```

Out put:-

Enter purchase amount = 20.30
Enter sale amount = 32.00
Profit = 11.70

d) A function **with argument** and **return value**.

Example- 1.

```
#include<stdio.h>
main()
{
float a, b, c;
printf("\n Enter value of a and b=");
scanf("%f%f",&a,&b);
c=f1(a,b);
printf("\n Value of c=%3.2f",c);
}
float f1(float a1, float b1)      //
{
float t;
t = a1/b1;
return t;
}
```

Out put:-

Enter value of a and b = 12.30
15.30
Value of c = 0.80

Example 2.

```
#include<stdio.h>
#include<math.h>
main()
{
float a, b, c, d;
printf("\n Enter value of a and b=");
scanf("%f%f",&a,&b);
c = f1(a, b);
printf("\n Value of c=%6.3f",c);
d = f2(a, b);
printf("\n Value of d=%6.3f",d);
}
float f1(float a1, float b1)                    //
{
float t;
t = pow(a1,b1);
return t;
}
float f2(float a2, float b2)
{
float t1;
t1 = sqrt(a2*b2);
return t1;
}
```

e) A function **call by Value** and **call by reference**:-

The **call by value** method of passing arguments to a function copies the actual value of an argument into the formal parameter of the function. In this case, changes made to the parameter inside the function have no effect on the argument. By default, C programming uses call by value to pass arguments.

The **call by reference** method of passing arguments to a function copies the address of an argument into the formal parameter. Inside the function, the address is used to access the actual argument used in the call. It means the changes made to the parameter affect the passed argument.

Example 1.

```
#include<stdio.h>
main()
{
float a, b, c;
printf("\n Enter value of a and b=");
scanf("%f%f",&a,&b);
c = f1(&a, &b);
printf("\n Value of c = %6.2f",c);
}
float f1(float *a1, float *b1)                  //
{
float t;
t = (*a1)*(*b1);
return t;
}
```

Out put:-

Enter value of a and b = 20.30
30.20

Value of c = 613.06

***f)** **Recursive** function/**Calling itself** function: - A function that contains a function call to itself or a function call to a second function which eventually calls the first function is **known as recursive function**.

Condition for recursion:-

1) Each time a function calls itself it must be nearer, in some sense to a solution.
2) There must be a decision criterion for stopping the process or computation.

Example: - 1.

Recursive function for factorial:-

fact (n) = 1 if n=0
fact (n) = n*fact (n-1) if n>0

if n = 5 ,then

f (5) = 5 × fact(4)
= 5 × 4 × fact (3)
= 5 × 4 × 3 × fact (2)
= 5 × 4 × 3 × 2 × fact (1)
= 5 × 4 × 3 × 2 × 1 × fact (0)
= 5 × 4 × 3 × 2 × 1 × 1

Program: 1 - factorial using recursion method.

```
#include<stdio.h>
main()
{
int n;
long int t;
printf("\n\t Enter value of n=");
scanf("%d",&n);
t =fact (n);
printf("\n\n Factorial of number = %d = %ld", n, t);
}
long int fact(long int n1)                                    //
{
if(n1==0)
return 1;
else
return n1*fact(n1-1);
}
```

Out put:-

Enter value of n = 5
Factorial of number = 5 = 120

Program: 2- Series of factorial using recursion method

```
#include<stdio.h>
main()
{
int n, i;
long int t;
printf("\n Enter value of n=");
scanf("%d",&n);
printf("\n Series of Factorial of number=");
for(i=0; i<=n; i++)
{
t = fact(i);
printf("\n\n Factorial of number=%d=%ld\n", i, t);
}
```

Out put:-

Enter value of **n** = 6
Series of Factorial of number =
Factorial of number = 0 = 1
Factorial of number = 1 = 1
Factorial of number = 2 = 2
Factorial of number = 3 = 6
Factorial of number = 4 = 24
Factorial of number = 5 = 120
Factorial of number = 6 = 720

```
}
long fact(long int n1)                    //
{
if(n1==0)
return 1;
else
return n1*fact(n1-1);
}
```

Example: - 2.

Recursive function for Fibonacci:-

fib (n) = 0 if n = 0
fib (n) = 1 if n = 1
fib (n) = fib (n-1) + fib (n-2) if n > 1.

Program: -1. **Series of Fibonacci** using **recursion method**

```
#include<stdio.h>
main()
{
int n, i, t;
printf("\n Enter value of n=");
scanf("%d",&n);
printf("\n series of fibonacci =\n\n");
for(i=0; i<=n; i++)
{
t = fib(i);
printf("%d\t", t);
}
}
int fib(int n1)                           //
{
if(n1==0 || n1==1)
return 1;
else
return fib(n1-1) + fib(n1-2);
}
```

Out put:-

Enter value of **n** = 12
Series of Fibonacci =
1 1 2 3 5 8 13
21 34 55 89 144 233

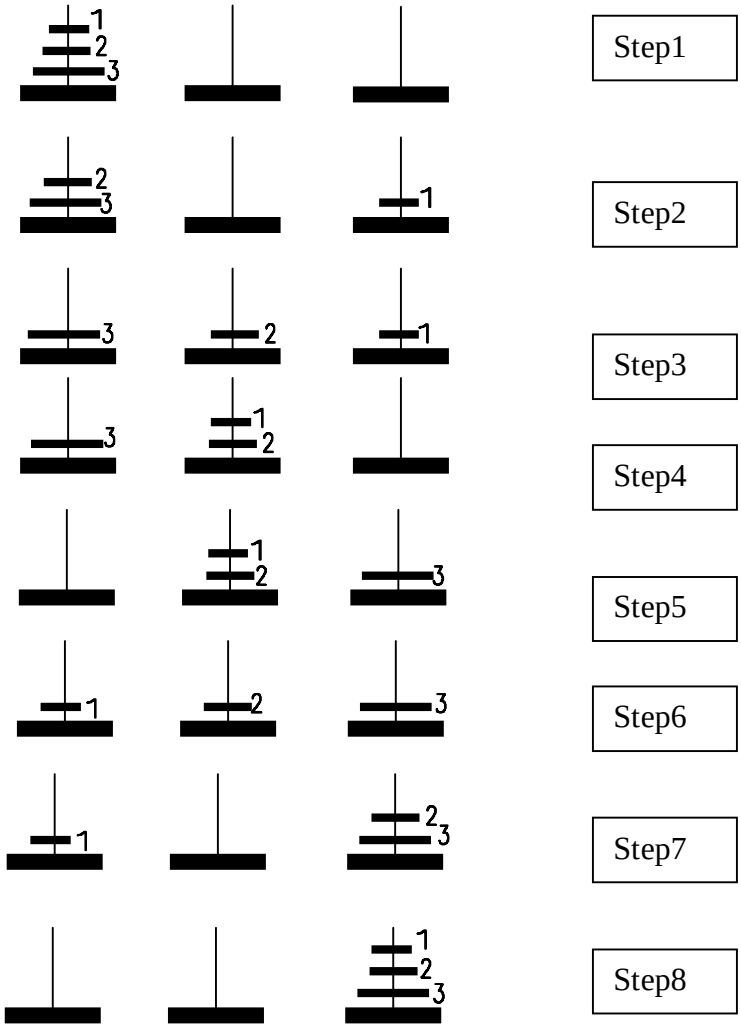
Tower of Hanoi Using Recursion Method

Tower of Hanoi is a **historical problem**, which can be easily expressed using recursion. There are **n disks** of **decreasing size** stacked on **one needle**, and **two other empty needles**. It is required to stack all disks onto a second needle in the decreasing order of size. **Third needle can be used, as temporary storage**.

The movement of disks must confirm to the following rules.

1. Only one disk may be moved at a time
2. A disk can be moved from any needle to any other.
3. At no time, a larger disk rests upon a smaller one.

Example 1:- Let number of disks **n** = **3**



Program: - Solving of **tower of Hanoi** using **recursion method**

```
#include<stdio.h>
main()
{
int n;
char Source = 'A', Middle = 'B', Target = 'C';
void hanoi(int,char,char,char);
printf("\n Enter Number of Disks=");
scanf("%d",&n);
printf("\n Tower of Hanoi with Disk=%d",n);
hanoi(n,Source,Middle,Target);
}
void hanoi(int n1,char left,char mid,char right)
{
if(n1!=0)
{
hanoi(n1-1, left, right, mid);
printf("\n Move Disk=%d\tFrom\t%c\tTo\t%c",n1,left,right);
hanoi(n1-1,mid,left,right);
}
}
```

Out put:-

```
Enter Number of Disks = 3
Tower of Hanoi with Disk = 3
Move Disk = 1    From  A  to  C
Move Disk = 2    From  A  to  B
Move Disk = 1    From  C  to  B
Move Disk = 3    From  A  to  C
Move Disk = 1    From  B  to  A
Move Disk = 2    From  B  to  C
Move Disk = 1    From  A  to  C
```

5. Array

Collection of similar data type's element is called **array**.

Types of array:-

1:- Single dimensional array.

2:- Double Dimensional Array

1:-Single dimensional array:-

Syntax:-

<Data_types> <Array_Name>[Size];

Example:-

int m[10] = {4,7,1,9,2,2,2,2,2,3};

m[0]	m[1]	m[2]	m[3]	m[4]	m[5]	m[6]	m[7]	m[8]	m[9]
4	7	1	9	2	2	2	2	2	3

2:-Double Dimensional Array

<Data_types><Array_Name>[Size1][Size2];

Or

<Data_types><Array_Name>[Row][Column];

Example 1:-

int P[2][3]

P[0][0]	P[0][1]	P[0][2]
P[1][0]	P[1][1]	P[1][2]

Example 2:-

int P[2][3] = {{2, 3, 4},{4, 7, 9}}

P[0][0]	P[0][1]	P[0][2]
P[1][0]	P[1][1]	P[1][2]
2	3	4
4	7	9

Program:- How to **input** elements in **single dimensional array.**

```
#include<stdio.h>
main()
{
int m[10],i;
printf("\n Enter elements of Single Dimensional arrays=\n");
for(i=0; i<=9; i++)
scanf("%d",&m[i]);
printf("\n Elements of Single dimensional arrays=\n");
for(i=0; i<=9; i++)
printf("%d\t",m[i]);
}
```

Out put:-

```
Enter elements of Single Dimensional array=
5
4
8
9
7
6
3
2
5
4
Elements of Single dimensional arrays=
5       4       8       9       7
6       3       2       5       4
```

Program: - Write a program to **sum** elements of **single dimensional array**.

```
#include<stdio.h>
main()
{
int m[10], i, s = 0;
printf("\n Enter elements of Single Dimensional arrays=\n");
for(i=0;i<=9; i++)
scanf("%d",&m[i]);
printf("\n Elements of Single dimensional arrays=\n");
for(i=0;i<=9; i++)
{
printf("%d\t",m[i]);
s = s + m[i];
}
printf("\nSum Of Arrays Elements=%d",s);
}
```

Out put:-

```
Enter elements of Single Dimensional array=
8
9
7
5
6
8
4
2
5
6
Elements of Single dimensional arrays=
8       9       7       5       6       8
4       2       5       6
Sum of Arrays Elements = 60
```

Program: - How to input and print elements in **double dimensional arrays and** in the form of **matrix** and print its **transpose matrix**.

```
#include<stdio.h>
main()
{
int m[3][4], row, col;
printf("\n Enter elements of Double Dimensional arrays=\n");
for(row=0; row<=2; row++)
for(col=0; col<=3; col++)
scanf("%d",&m[row][col]);
printf("\n Elements of Double dimensional arrays in the form
 of matrix=\n\n");
for(row=0; row<=2; row++)
{
for(col=0; col<=3; col++)
printf("[%d]\t",m[row][col]);
printf("\n");
}
printf("\n Elements of Double dimensional arrays in the form
of transpose matrix =\n\n");
for(col=0; col<=3; col++)
{
```

Out put:-

```
Enter elements of Double Dimensional arrays=
3
6
8
9
5
6
2
3
7
5
6
9
Elements of Double Dimensional arrays in the form
of matrix=
[3]     [6]     [8]     [9]
[5]     [6]     [2]     [3]
[7]     [5]     [6]     [9]
Elements of Double Dimensional arrays in the form
of transpose matrix=
[3]     [5]     [7]
[6]     [6]     [5]
[8]     [2]     [6]
[9]     [3]     [9]
```

```
for(row=0; row<=2; row++)
printf("[%d]\t",m[row][col]);
printf("\n");
}
}
```

Program: - Write a program to **add any two matrix**.

```
#include<stdio.h>
main()
{
int m1[3][3], m2[3][3], m3[3][3], rows, cols;
printf("\n Enter elements of First Matrix=\n");
for(rows=0; rows<=2; rows++)
for(cols=0; cols<=2; cols++)
scanf("%d",&m1[rows][cols]);
printf("\n Enter elements of Second Matrix=\n");
for(rows=0; rows<=2; rows++)
for(cols=0; cols<=2; cols++)
scanf("%d",&m2[rows][cols]);
printf("\n Elements of First Matrix=\n\n");
for(rows=0; rows<=2; rows++)
{
 for(cols=0; cols<=2; cols++)
printf("[%d]\t",m1[rows][cols]);
printf("\n");
}
printf("\n Elements of Second Matrix=\n\n");
for(rows=0; rows<=2; rows++)
{
for(cols=0; cols<=2; cols++)
printf("[%d]\t",m2[rows][cols]);
printf("\n");
}
for(rows=0; rows<=2; rows++)
{
for(cols=0; cols<=2; cols++)
m3[rows][cols] = m1[rows][cols] + m2[rows][cols];
}
printf("\n Sum of Above Two of Matrix=\n\n");
for(rows=0; rows<=2; rows++)
{
for(cols=0; cols<=2; cols++)
printf("[%d]\t",m3[rows][cols]);
printf("\n");
}
}
```

6. Sorting

It is a technique for **ordering elements** either ascending order **or** descending order.

or

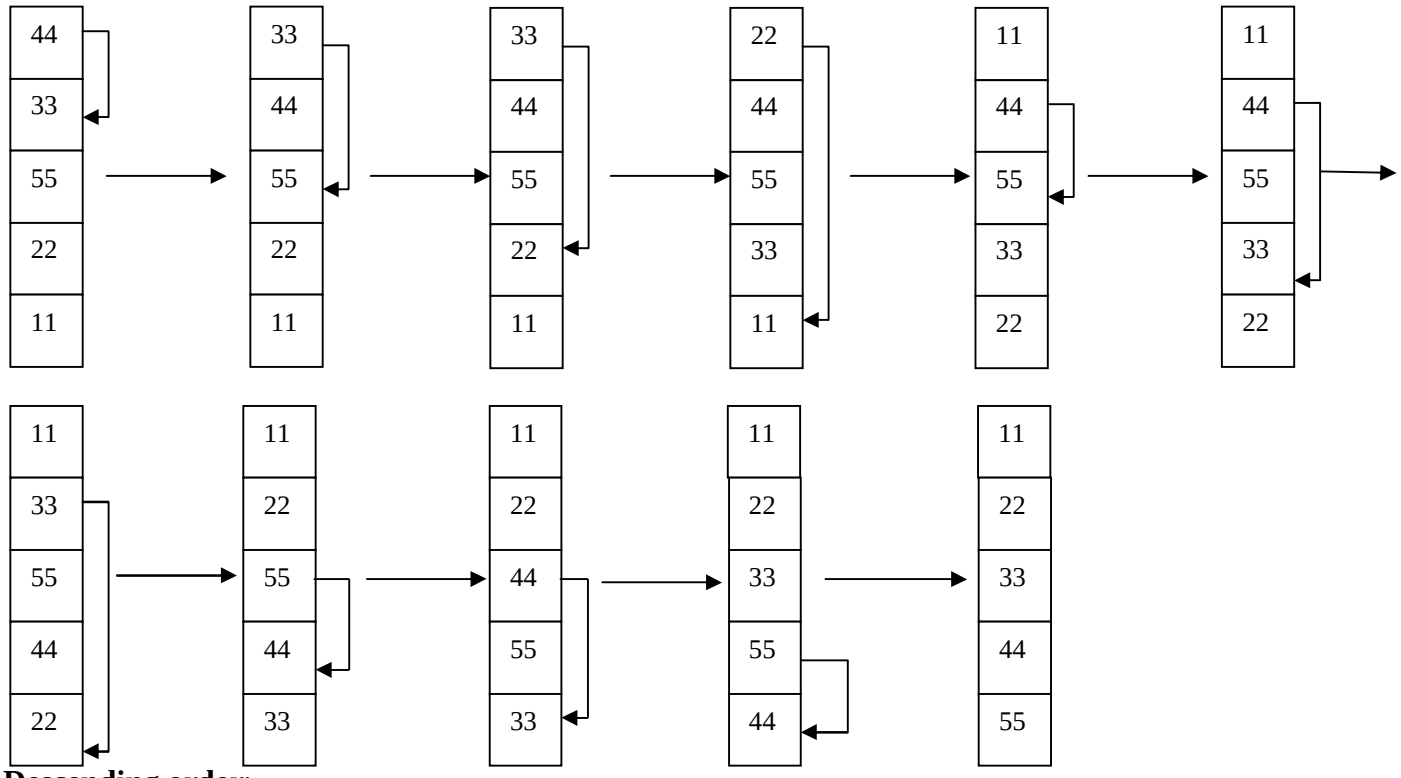
Sorting is any process of arranging items systematically, and has two common, yet distinct meanings: ordering: arranging items in a sequence ordered by some criterion; categorizing: grouping items with similar properties.

1. Selection Sort
2. Bubble Sort
3. Insertion Sort
4. Heap Sort
5. Radix Sort/Bucket Sort
6. Quick Sort
7. Merge Sort ×

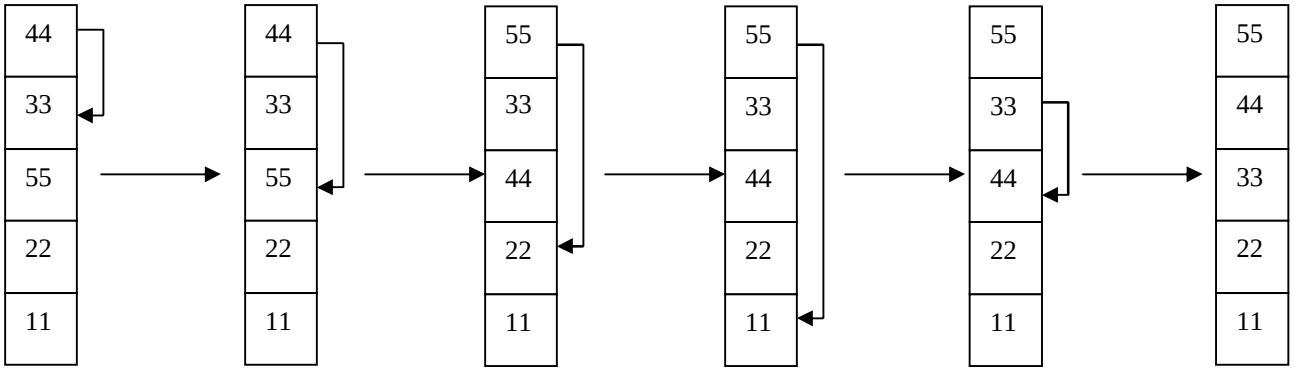
1. <u>Selection Sort</u> ⇨ Sort the following unordered elements in either ascending order or descending order.

44, 33,55,22,11

Ascending order:-



Descending order:-



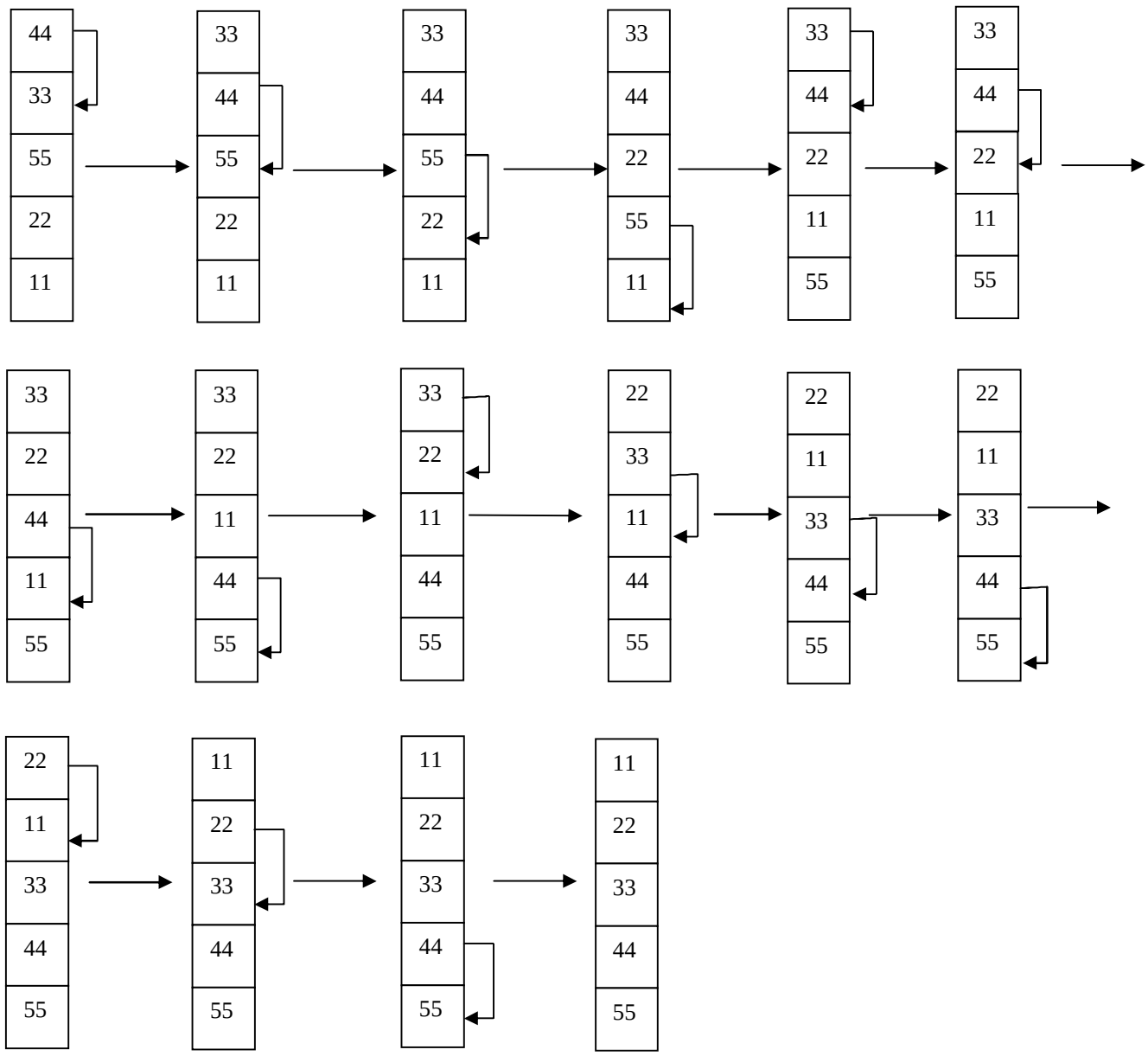
Program of selection sort:-

```
#include<stdio.h>
main()
{
int m[10], i, j, t;
printf("\n Enter Unordered Elements =\n ");
for(i=0; i<=9; i++)
scanf("%d",&m[i]);
printf("\n Unordered Elements =\n ");
for(i=0; i<=9; i++)
printf("%d\t",m[i]);
printf("\n Ascending order element=\n");
for(i=0; i<=9; i++)
{
for(j=i; j<=9; j++)
{
if(m[i]>m[j])
{
t = m[i];
m[i] = m[j];
m[j] = t;
}
}
}
for(i=0; i<=9; i++)
printf("%d\t",m[i]);
printf("\n Descending order element=\n");
for(i=0; i<=9; i++)
{
for(j=i+1; j<=9; j++)
{
if(m[i]<m[j])
{
t = m[i];
m[i] = m[j];
m[j] = t;
}
}
}
for(i=0; i<=9; i++)
printf("%d\t",m[i]);
}
```

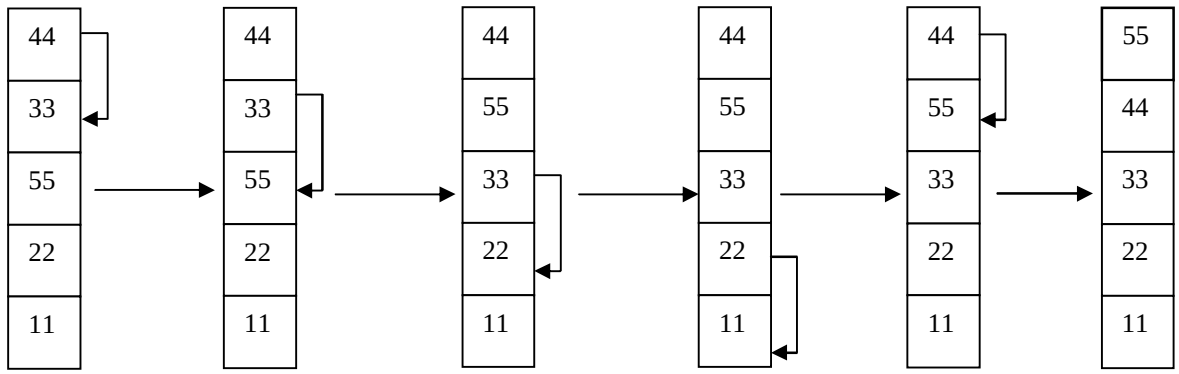
2. <u>Bubble Sort</u> ⇨ Sort the following unordered elements in either ascending order or descending order.

44, 33,55,22,11

Ascending order:-



Descending order:-



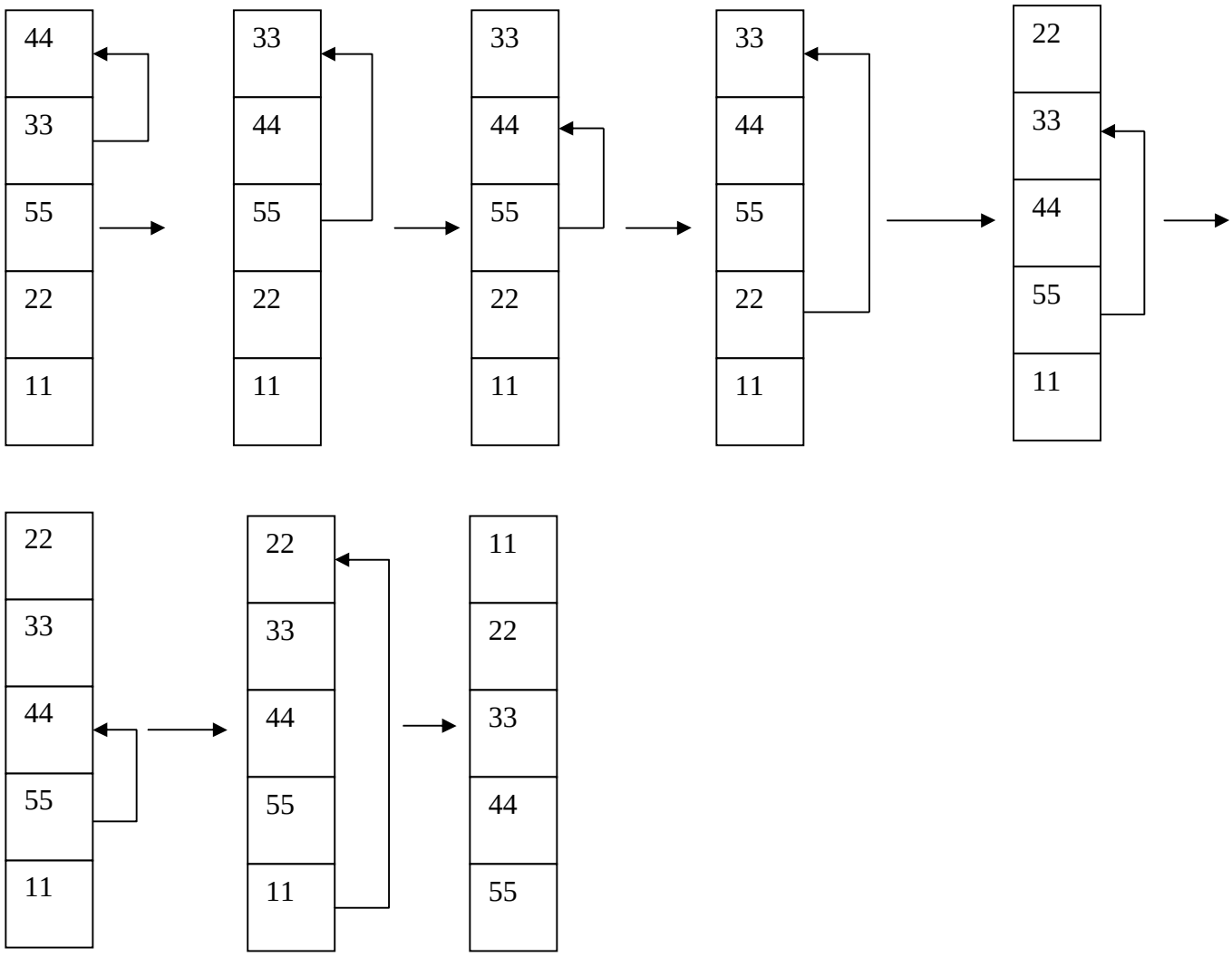
Program of bubble sort:-

```
#include<stdio.h>
main()
{
int m[10], i, j, t;
printf("\n Enter Unordered Elements =\n ");
for(i=0; i<=9; i++)
scanf("%d",&m[i]);
printf("\n Unordered Elements =\n ");
for(i=0; i<=9; i++)
printf("%d\t",m[i]);
printf("\n Ascending order element=\n");
for(i=0; i<=9; i++)
{
for(j=0; j<=9-i; j++)
{
if(m[j]>m[j+1])
{
t = m[j];
m[j] = m[j+1];
m[j+1] = t;
}
}
}
for(i=0; i<=9; i++)
printf("%d\t",m[i]);
printf("\n Descending order element=\n");
for(i=0; i<=9; i++)
{
for(j=0; j<=9-i; j++)
{
if(m[j]<m[j+1])
{
t = m[j];
m[j] = m[j+1];
m[j+1] = t;
}
}
}
for(i=0; i<=9; i++)
printf("%d\t",m[i]);
}
```

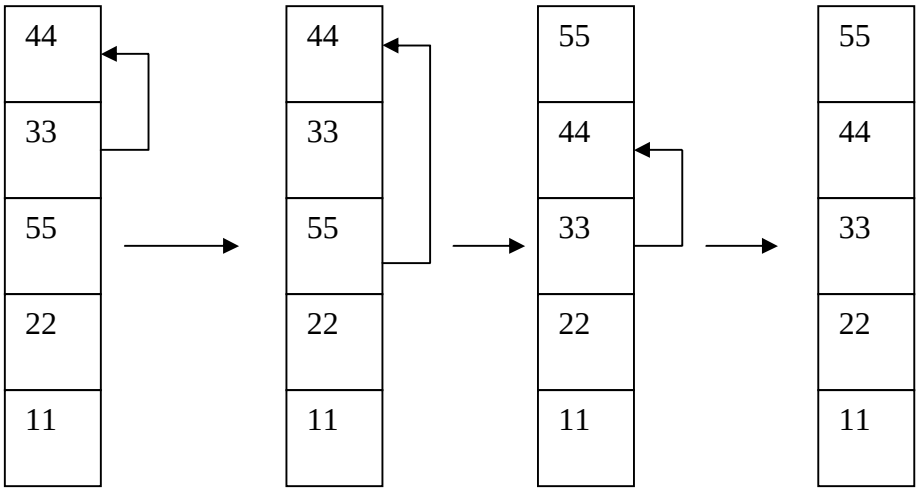
3. <u>Insertion Sort</u> ⇨ Sort the following unordered elements in either ascending order or descending order.

44, 33,55,22,11

Ascending order:-



<u>Descending order</u>:-



Program of insertion sort:-

```
#include<stdio.h>
main()
{
int m[10], i, j, k, t;
printf("\n Enter Unordered Elements =\n ");
for(i=0; i<=9; i++)
scanf("%d",&m[i]);
printf("\n Unordered Elements =\n ");
for(i=0; i<=9; i++)
printf("%d\t",m[i]);
printf("\n Ascending Ordered Elements =\n ");
for(i=0; i<=9; i++)
{
t=m[i];
for(j=0; j<i; j++)
{
 if(t<m[j])
{
for(k=i; k>=j; k--)
m[k] = m[k-1];
m[j] = t;
break;
}
}
}
for(i=0; i<=9; i++)
printf("%d\t",m[i]);
printf("\n Descending Ordered Elements =\n ");
for(i=0; i<=9; i++)
{
t=m[i];
for(j=0; j<i; j++)
{
if(t>m[j])
{
for(k=i; k>=j; k--)
m[k] = m[k-1];
m[j] = t;
break;
}
}
}
for(i=0; i<=9; i++)
printf("%d\t",m[i]);
}
```

4. <u>Heap Sort</u> ⇨ The elements of the heap tree are **represented by an array**. The **root** will be the **largest elements** of the heap tree. Since it is maintained in array, so the largest value should be the **last element of array**.

Steps for Heap Sort:-

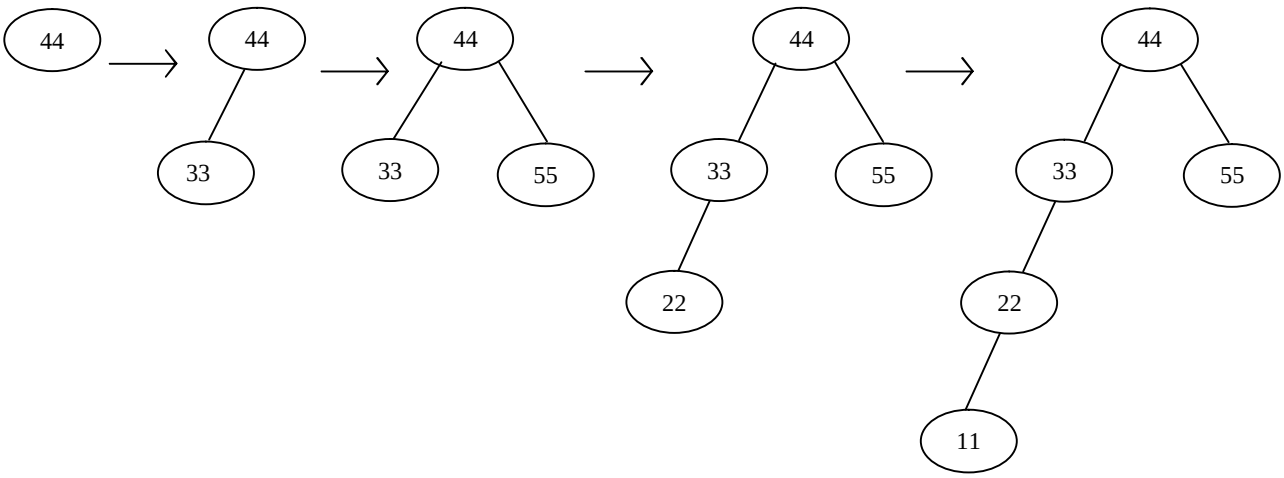
Step 1:- Replace the root with last node of the heap.

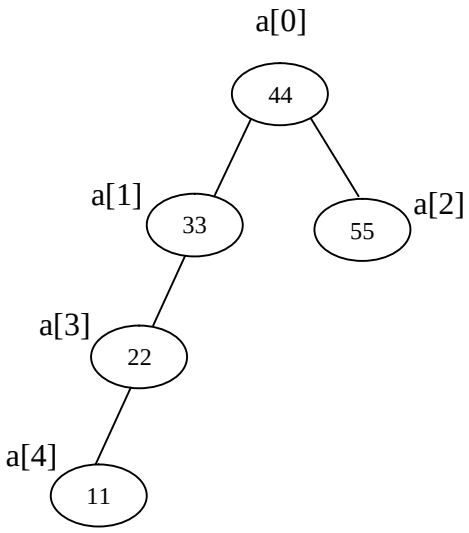
Step 2:- Keep the last node at the proper position; means do the delete operation in heap tree **but here deleted node is root**

Example: - Sort the following unordered elements in either ascending order or descending order.

44, 33, 55, 22, 11

Step1:-





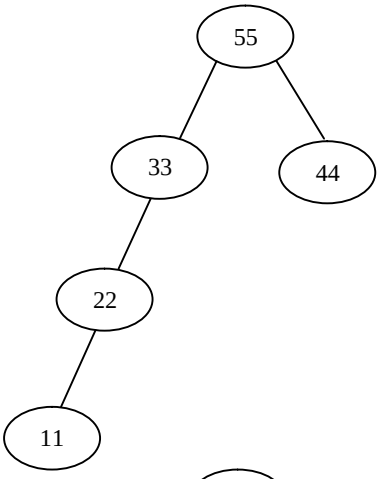
44	33	55	22	11

Keep the last node at proper position

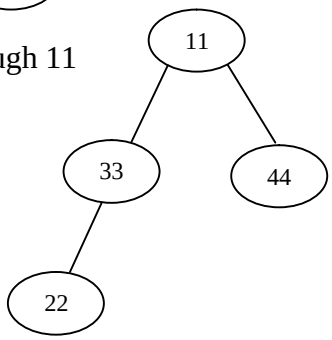
55	33	44	22	11

Step2:- Make **Heap Tree** of above binary tree

55	33	44	22	11



55 replace through 11



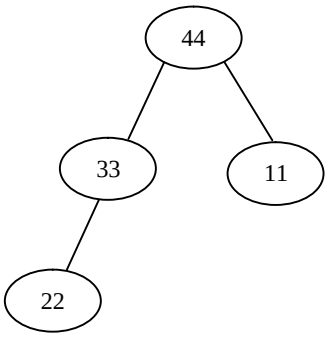
11	33	44	22	55

Keep the last node at proper position

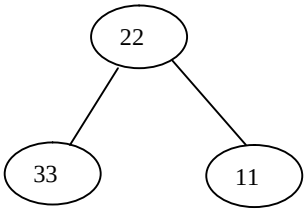
44	33	11	22	55

Step3:- Make **Heap Tree** of above tree

44	33	11	22	55



44 replace through 22



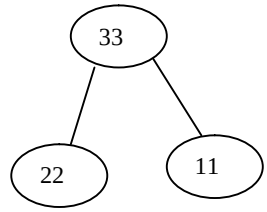
22	33	11	44	55

Keep the last node at proper position

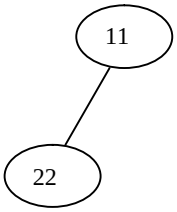
33	22	11	44	55

Step4:- Make **Heap Tree** of above tree

33	22	11	44	55



33 replace through 11



11	22	33	44	55

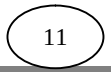
Keep the last node at proper position

22	11	33	44	55

Step5:- Make **Heap Tree** of above tree

22	11	33	44	55

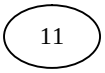
22 replace th 1



11	22	33	44	55

Step6:- Make **Heap Tree** of above tree

11	22	33	44	55

11 replace 

11	22	33	44	55

5. <u>Radix/Bucket sorting technique</u> ⇨ This sorting technique based on the logic of alphabetical order process.

Example 1:-Sort the following elements either ascending order or descending order.

44	33	55	22	11

Pass 1:-

	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]
44					44	
33				33		
55						55
22			22			
11		11				

11, 22, 33, 44, 55

Example 2:- Sort the following elements either ascending order or descending.

944	633	355	722	811

Pass1:-

	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
944					944					
633				633						
355						355				
722			722							
811		811								

811, 722, 633, 944, 355

Pass 2:-

	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
811		811								
722			722							
633				633						
944					944					
355						355				

811, 722, 633, 944, 355

Pass 3:-

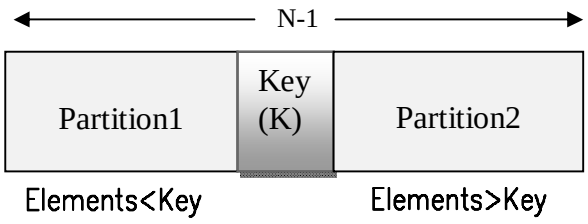
	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
811									811	
722								722		
633							633			
944										944
355				355						

355, 633, 722, 811, 944

6. Quick Sort/Partition Exchange technique

⇨ It is one of the most popular sorting techniques. It was **developed** by **C.A.R. Hoare**. The quick sort algorithm works by partitioning the array to be sorted and each partition is in turn sorted recursively. In partition, one of the array elements is chosen as **key** (pivot) **elements**. This **key** value can be the **first element** of an array. That is, if a is an array then **key** = **a [0]** and rest of the array elements are grouped into two partitions such that–

1. One partition contains elements **smaller than the key value**.
2. Another partition contains elements **larger than the key value**.



Rule:-

a) All the elements on the **left side** of pivot should be **smaller** or **equal** to (**key**) pivot.
b) All the elements on the **right side** of pivot should be **greater** to (**key**) pivot.

Similarly, we choose the pivot for dividing the sub lists until there are 2 or more elements in the list.

Example:- Sort the following elements **by using quick sort technique**.

44	33	55	22	11

Key (Pivot) Value

44	33	55	22	11

11 is smaller than 44 we interchange 44 with 11

11	33	55	22	**44**

Now we start from 11 will be from left to right.
The first element greater than 44 is 55.So interchange it with key.

11	33	**44**	22	55

Now the comparison will start from 55 and will be from right to left
The first element Smaller than 44 is 22.So interchange it with key.

The first element less than 44 is 22.So interchange it with key.

11	33	22	**44**	55

Now the comparison will start from 22 and will be from left to right.
The first element Smaller than 44 is 33.So interchange it with key.

11	**44**	22	33	55

Now the comparison will start from 33 and will be from Left to Right
The first element smaller than 44 is 22.So interchange it with key.

11	22	**44**	33	55

Now the comparison will start from 22 and will be from Right to Left
The first element smaller than 44 is 33.So interchange it with key.
The first element smaller than 44 is 33.So interchange it with key.

11	22	33	**44**	55

7. String

String ⇨ It is sequence of characters enclosed within double quote.

Example:-1

"VARANASI" Consist of **9 characters**

'V','A','R','A','N','A','S','I','\0'→**Null Character**

Example:-2

"V A R A N A S I" Consist of **16 characters**

V		A		R		A		N		A		S		I	**'\0'**

Note: - String terminated by null character ('\0').

Syntax:-

Data_Types <String_name> [**Size**];

Example:-3

char name[12];

Example:-4

"9" = **Two Bytes** '9' = **One Byte**

"a" = Two Bytes 'a' = One Byte
" " = Two Bytes ' ' = One Byte

Inputation and Displaying of strings:-

Method 1.

```
#include<stdio.h>
#include<string.h>
main()
{
char name[20], father[30];
printf("\n Enter Name of Student=");
scanf("%s",&name);
printf("\n Enter Name of Father=");
scanf("%s",&father);
printf("\n Name of student =%s\n Name of father=%s", name, father);
}
```

Method 2.

```
#include<stdio.h>
#include<string.h>
main()
{
char name[20], father[30];
printf("\n Enter Name of Student=");
gets(name);
```

```
printf("\n Enter Name of Father=");
gets(father);
printf("\n Name of Student =”);
puts(name);
printf("\n Name of Father =”);
puts(father);
}
```

String Functions:- a) strlen(string), b) strrev(string), c) strcat(string1, string2), d) strcpy(string1, string2)
e) strcmp(string1, string2)

a) strlen (string**)**:- It returns **length** of string.

Example**:**

```
#include<stdio.h>
#include<string.h>
main()
{
int m;
char name[20];
printf("\n Enter Name of Student=");
gets(name);
printf("\n Name of Student =”);
puts(name);
m = strlen(name);
printf("\n Length of string name=%d", m);
}
```

b) strrev (string**)**:- It **reverses** string.

Example**:**

```
#include<stdio.h>
#include<string.h>
main()
{
char name[20];
printf("\n Enter Name of Student=");
gets(name);
printf("\n Name of Student =”);
puts(name);
printf("\n Reverse Name of Student=%s", strrev(name));
}
```

c) strcat (String1**,** String2**):-** It is **used** for **joining** string2 into string1.

Example:

```
#include<stdio.h>
#include<string.h>
main()
{
char str1[10],str2[20];
printf("\n Enter Any String1=");
gets(str1);
printf("\n Enter Any String2=");
```

```
gets(str2);
printf("\n Input String1=");
puts(str1);
printf("\n Input String2=");
puts(str2);
strcat(str1,str2);
printf("\n Concatenated String=%s",str1);
}
```

d:-strcpy (String1,String2):- It copy string2 into string1.

Example:

```
#include<stdio.h>
#include<string.h>
main()
{
char str1[10],str2[20];
printf("\n Enter Any String1=");
gets(str1);
printf("\n Enter Any String2=");
gets(str2);
printf("\n Input String1=");
puts(str1);
printf("\n Input String2=");
puts(str2);
strcpy(str1,str2);
printf("\n String1 after copy String2=%s",str1);
}
```

e:-strcmp (String1,String2):- This function **compares** string1, string2, and **return** following values.

a) Less than zero (It means string1 is less than string2)
b) Equal zero (It means both strin1 and string2 are identical)
c) Greater than zero(It means string1 is greater than string2)

Example:

```
#include<stdio.h>
#include<string.h>
main()
{
char str1[10], str2[20];
int m;
printf("\n Enter Any String1=");
gets(str1);
printf("\n Enter Any String2=");
gets(str2);
printf("\n Input String1=");
puts(str1);
printf("\n Input String2=");
puts(str2);
m=strcmp(str1,str2);
if(m==0)
printf("\n Both strings are identical");
else
if(m>0)
printf("\nstring1 is greater than string2");
```

```
else
        printf("\nstring1 is Less than string2");
        }
```

Q. Write a program to check input string is **palindrome** or **not palindrome**.

```
        #include<stdio.h>
        #include<string.h>
        main()
        {
        char str1[10],str2[15];
        int m;
        printf("\n Enter Any String1=");
        gets(str1);
        printf("\n Input String1=");
        puts(str1);
        strcpy(str2,str1);
        strrev(str2);
        m=strcmp(str1,str2);
        if(m==0)
        printf("\n String is palindrome");
else
        printf("\n String1 is not palindrome");
        }
```

8. Structures and Unions

Structure

Structure **combine logically** related data items into a **single unit**. The data items enclosed within a structure are known as **members** and they can be same as different data types. It is **defined by users** as per their requirements.

Syntax:-

```
struct <structure_Name>
{
Data_Type1 member1;
Data_Type2 member2;
Data_Type3 member3;
…
…
…

}<object1>,<object2>…;
```

Example:-

```
#include<stdio.h>
struct st1
{
int eno;
char name[20];
float sal;
};
main()
{
struct st1 x;
printf("\n Enter Employee Number=");
scanf("%d",&x.eno);
printf("\n Enter Employee Name=");
scanf("%s",&x.name);
printf("\n Enter Employee salary=");
scanf("%f",&x.sal);
printf("\n Employee Number=%d”,x.eno);
printf("\n Employee Name=%s”,x.name);
printf("\n Employee salary=%f”,x.sal);
printf("\n\n Size of structure 1=%d",sizeof (x));
}
```

or

```
#include<stdio.h>
struct st1
{
int eno;
char name[20];
float sal;
};
struct st2
{
char job[10], fname[20];
 int age;
```

```
};
main()
{
struct st1 x;
struct st2 y;
printf("\n Enter Employee Number=");
scanf("%d",&x.eno);
printf("\n Enter Employee Name=");
scanf("%s",&x.name);
printf("\n Enter Employee Father Name=");
scanf("%s",&y.fname);
printf("\n Enter Employee Job=");
scanf("%s",&y.job);
printf("\n Enter Employee Age=");
scanf("%d",&y.age);
printf("\n Enter Employee Salary=");
scanf("%f",&x.sal);
printf("\n Employee Number=%d",x.eno);
printf("\n Employee Name=%s",x.name);
printf("\n Employee Father Name=%s",y.fname);
printf("\n Employee Age=%d",y.age);
printf("\n Employee salary=%f",x.sal);
printf("\n\n Size of structure 1=%d",sizeof (x));
printf("\n\n Size of structure 2=%d",sizeof (y));
}
```

Union

It is the **collection of different types of data item**. It is different **from** structure i.e. **in the structure** all the members uses **individual memory locations** where **in the case of union** all the members are pointing the **same memory locations**. The **size of memory** will be according to the size of those data members **whose memory size is the biggest** among all.

Syntax:-

```
union  <Union_Name>
{
Data_Type1 member1;
Data_Type2 member2;
Data_Type3 member3;
…
…
…
}<object1>,<object2>…;
```

Example:-

```
#include<stdio.h>
struct st1
{
int eno;
char name[20];
float sal;
};
struct st2
{
 char job[10],fname[20];
```

```
int age;
};

union un1
{
struct st1 x;
struct st2 y;
};
main()
{
union un1 z;
printf("\nEnter Employee Number=");
scanf("%d",&z.x.eno);
printf("\nEnter Employee Name=");
scanf("%s",&z.x.name);
printf("\nEnter Employee Father Name=");
scanf("%s",&z.y.fname);
printf("\nEnter Employee Job=");
scanf("%s",&z.y.job);
printf("\nEnter Employee Age=");
scanf("%d",&z.y.job);
printf("\nEnter Employee Salary=");
scanf("%f",&z.x.sal);
printf("\n Employee Number=%d",z.x.eno);
printf("\n Employee Name=%s",z.x.name);
printf("\n Employee Father Name=%s",z.y.fname);
printf("\n Employee Job=%s",z.y.job);
printf("\n Employee Age=%d",z.y.age);
printf("\n Employee salary=%f",z.x.sal);
printf("\n\nSize of Union z =%d",sizeof(z));
}
```

9. Pointer

Pointer: -

It is variable which point the address of variable. Pointer variable represented by using **indirection operator** (*).

Syntax:-

<Data_type> <variable1>,<variable2>,<variable3>…,*ptr1, *ptr2,*ptr3…;

Where :-

ptr1=address of variable1 (&variable1);
ptr2=address of variable2 (&variable2);
ptr3=address of variable3 (&variable3);
…
…
…
*ptr1=Value at address variable1;
*ptr2=Value at address variable2;
*ptr3=Value at address variable3;
…
…
…

Example:-1

```
int p=12,q=20,*p1,*p2;
p1=&p;
p2=&q;
*p1=value at address p;
*p2=value at address q;
```

Example:-2

```
#include<stdio.h>
main()
{
int p=12,q=20,*p1,*p2;
p1=&p;
p2=&q;
printf("\nValue at address p=%d",p);
printf("\nValue at address q=%d",q);
printf("\nValue at address p=%d",*p1);
printf("\nValue at address q=%d",*p2);

printf("\nAddress of p=%u",p1);
printf("\nAddress of q=%u",p2);

printf("\nAddress of p=%u",&p);
printf("\nAddress of q=%u",&q);
}
```

Q. W.A.P. to generate a series using pointer in single dimensional array**.**

```
#include<stdio.h>
main()
{
int m[10],i;
printf("\n Enter Elements of Array=");
for(i=0;i<=9;i++)
scanf("%d",&m[i]);
printf("\n Elements of Array=\n");
for(i=0;i<=9;i++)
printf("%d\t",*(m+i));
printf("\n Address of Array=\n");
for(i=0;i<=9;i++)
printf("%u\t",&m[i]);
}
```

Q. W.A.P. to **display matrix** using **pointer.**

```
#include<stdio.h>
main()
{
 int m[3][3],i,j;
printf("\n Enter elements in matrix=\n");
for(i=0;i<=2;i++)
for(j=0;j<=2;j++)
scanf("%d",&m[i][j]);
printf("\n Elements in matrix=\n");
for(i=0;i<=2;i++)
{
for(j=0;j<=2;j++)
printf("%d\t",*(*(m+i)+j));
printf("\n");
}
}
```

Q. W.A.P. to **display double matrix** using **pointer.**

```
#include<stdio.h>
main()
{
int m[4][4], rows, cols;
printf("\n Enter elements of Double dimensional srrays=\n");
for(rows=0; rows<=3; rows++)
for(cols=0; cols<=3; cols++)
scanf("%d",&m[rows][cols]);
printf("\n Elements of double dimensional arrays=\n");
for(rows=0; rows<=3; rows++)
{
for(cols=0; cols<=3; cols++)
printf("%d\t",*(*(m+rows)+cols));
printf("\n");
}
```

```
printf("\n Address of elements of matrix=\n");
for(rows=0; rows<=3; rows++)
{
for(cols=0; cols<=3; cols++)
printf("%u\t",& m[rows][cols]);
printf("\n");
}
printf("\n Transpose of matrix=\n");
 for(cols=0; cols<=3; cols++)
{
for(rows=0; rows<=3; rows++)
printf("%d\t",*(*(m+rows)+cols));
printf("\n");
}
}
```

Use of Pointer:-

1. A pointer allows a function or a program to access a variable outside the preview of the function or program.
2. **Use of pointer** increases makes the **program execution faster**.
3. Using pointers, arrays and structures can be handled in **more efficient** way.
4. **Without pointers**, it will be impossible to create complex data structure such as **linked lists**, **trees** and **graphs**

Q. W.A.P. to solve arithmetic problems by **using pointer**.

```
#include<stdio.h>
main()
{
 float a, b, sum, pro ,div, minus, *p1,*p2;
 p1=&a;
 p2=&b;
 printf("\n Enter values of a and b=");
 scanf("%f%f", &a,&b);
 printf("\n Value of a=%6.2f\t Value of b=%6.2f", a,b);
 sum=(*p1)+(*p2);
 minus=(*p1)-(*p2);
 pro=(*p1)*(*p2);
 div=(*p1)/(*p2);
 printf("\n Sum=%6.2f", sum);
 printf("\n Difference=%6.2f", minus);
 printf("\n Product=%6.2f", pro);
 printf("\n Division=%6.2f", div);
 printf("\n Address of a=%u",p1);
 printf("\n Address of b=%u",p2);
}
```

Pointer To Pointer: -

A variable that hold an address of a variable that in turn holds an address of another variable. This type of variable will be known as pointer to pointer.

Example:-

```
int a = 2,*p1,**p2;
p1 = &a;
p2 = &p1;
```

Example:-

```
#include<stdio.h>
main()
{
int a=2,*p1,**p2;
p1=&a;
p2=&p1;
printf("\n Value at address p1=%d",*p1);
 printf("\n Value at address p2=%d",*p2);
printf("\n Address of First pointer=%u",p1);
printf("\n Address of Second pointer=%u",p2);
}
```

Types of pointer function in 'c' Language (Dynamic memory management):-

1. malloc() function
2. free() function

1. malloc() function: - It **allocates** a block of memory of requisite size and returns a generic pointer to the newly allocated block.

```
void *malloc(size_t,size)
```

Where:-

size_t is a type used for memory object size.
size represents the size of requested blocks in byte.

2. free() function:- It **de-allocates** a block of memory that was previously allocated with malloc().It takes a pointer to the block of memory as its arguments.

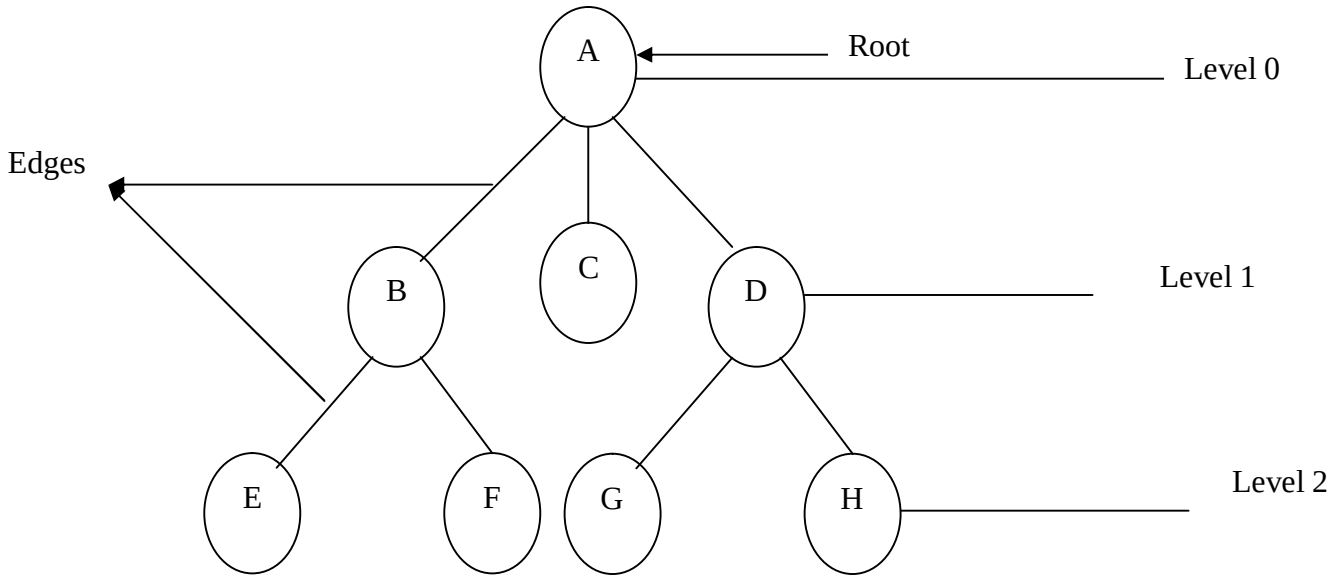
```
void *free(void *p);
```

Where:-

p is a pointer to the block to be **de-allocated**.

10. Trees

Tree ⇨ It is **a non-linear data structure** in which items are arranged in a **sorted sequence**. It is used to represent **hierarchical relationship** existing amongst several data items.



(**Figure of tree**)

Terminology of tree:-

Root: - In figure **A** is called root of tree.

Node: - Each data item in a tree is called node. There are 8 nodes in figure.

Degree of Node: - It is the **number of sub trees of a node** in a given tree

Example:-

Degree of Root (**A**) = 3
Degree of Node (**B**) = 2
Degree of Node (**G**) = 0

Degree of Tree: - It is the **maximum degree of nodes** in a given tree.

Example:-

Degree of Tree = 3

Terminal Node/Leaf: - A **node witch degree zero** is called terminal node or leaf.

Example:-

In figure there are five terminal nodes (C, E, F, G, H).

Non-Terminal Node: - Any **node whose degree is not zero** is called non-terminal node.

Example:-

In figure there are two non –terminal nodes (B, D).

Sibling: - The **children nodes of a given parent node** are called sibling. They are also called brothers.

Example:-

E and F are sibling whose parent is B
G and H are sibling whose parent is D

Level: - The entire tree structure is leveled in such a way that the **root node is always level 0**.Then its intermediate children at level-1 and their intermediate children at level-2 and so on.

Example:-

There are three levels in tree)

Edge: - It is a **connecting line of two nodes**.

Path: - It is a sequence of consecutive edges **from the source node to the destination node**.

Example:-

Path from A to G is (A, D) and (D, G)

Depth/Path: - It is the **maximum level of any node** in a given tree.

Example:-

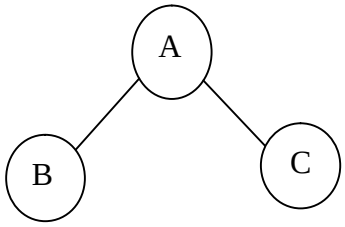
Node A has the highest level.

Depth = (A→D→H)

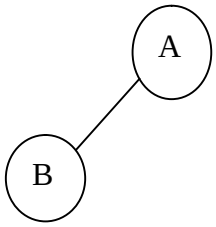
Forest: - In a set of disjoint trees. If we **remove its root node** then it becomes a forest.

Binary Tree :- It is a finite set of elements that is either empty or is partitioned into three disjoint subsets.

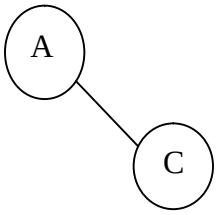
Example:-1



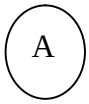
Example:-2



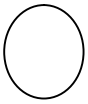
Example:-3



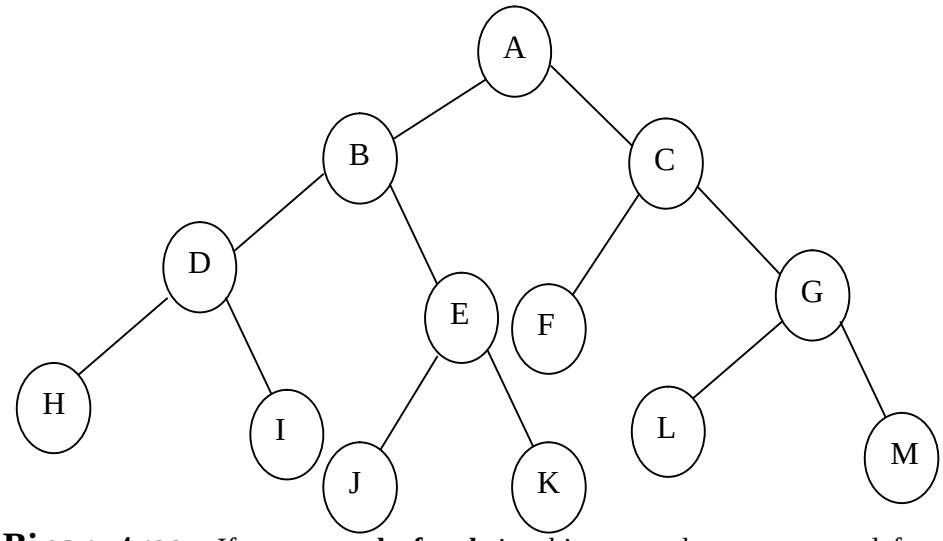
Example:-4 Binary tree with **one item.**



Example:-5 Binary tree with **no item.**



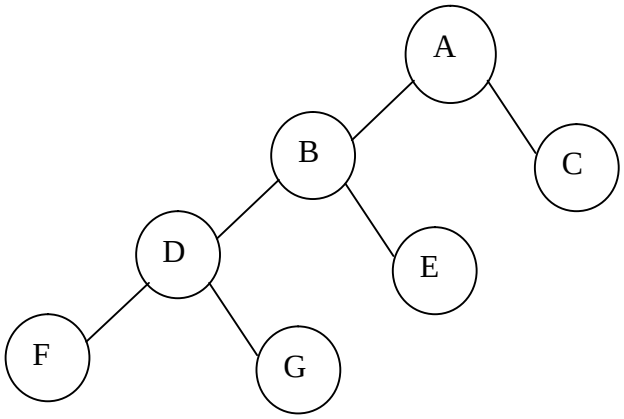
Complete Binary Tree :-



Strictly Binary tree: - If **every non-leaf node** in a binary tree has **non-empty** left and right sub tree then the tree is called strictly binary tree. Strictly binary tree with **n nodes** always contains **2*n-1 nodes**.

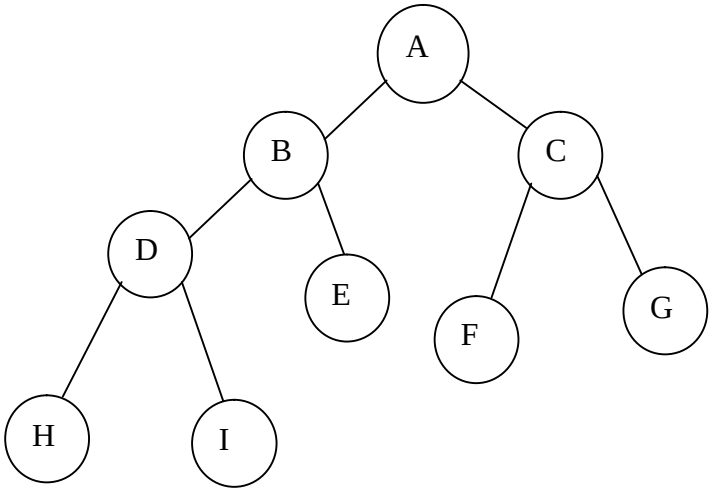
Example:-1 Strictly Binary Tree

Number of nodes = 7
Number of leaf = 4
Nodes = 2*n−1 = 2*4−1 = 7



Example:-2 Strictly Binary Tree

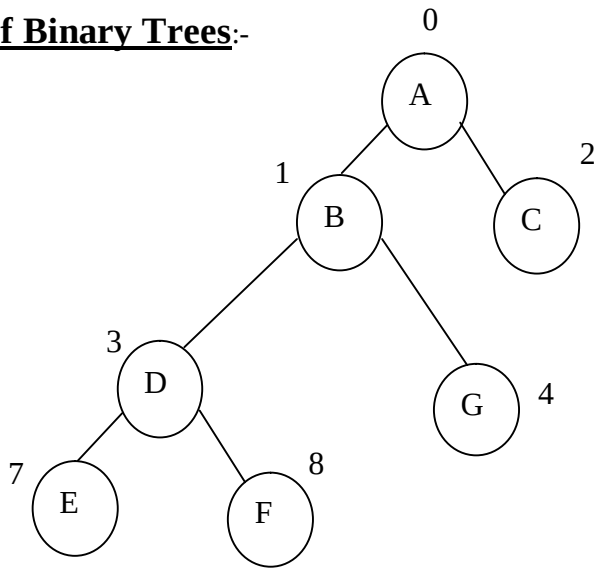
Number of nodes = 9
Number of leaf = 5
Nodes = 2*n−1 = 2*5−1 = 9



Note: - Complete binary tree is always Strictly Binary Tree **while** Strictly Binary Tree may or may not be Complete binary tree

Array Representation of Binary Trees:-

Example:-1



Tree [0] = A Tree [1] = B Tree [2] = C Tree [3] = D Tree [4] = G Tree [5] = 0 Tree [6] = 0

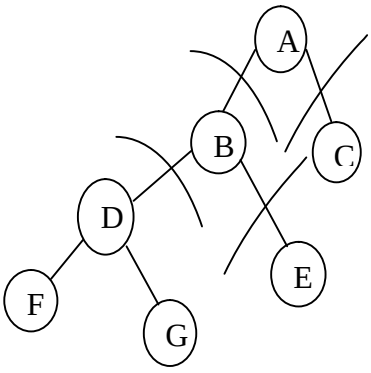
Tree [7] = E Tree [8] = F Tree [9] = 0 Tree [10] = 0

Traversal of Binary tree:-

1. Preorder Traversal. Rule:- **Node** – Left – Right
2. Inorder Traversal. Rule:- Left – **Node** – Right
3. Postorder Traversal. Rule:- Left – Right – **Node**

Example: - **Preorder Traversal**.

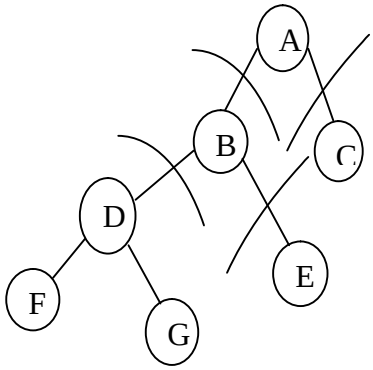
Rule: - **Node** – Left – Right



ABDFGEC

Example: - **Inorder Traversal.**

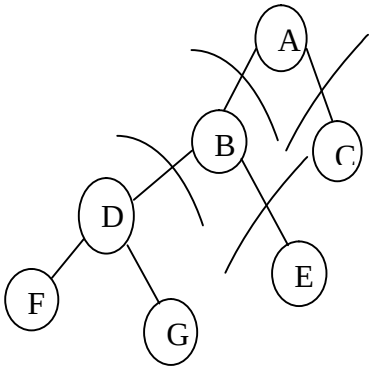
Rule: - Left – **Node** – Right



FDGBEAC

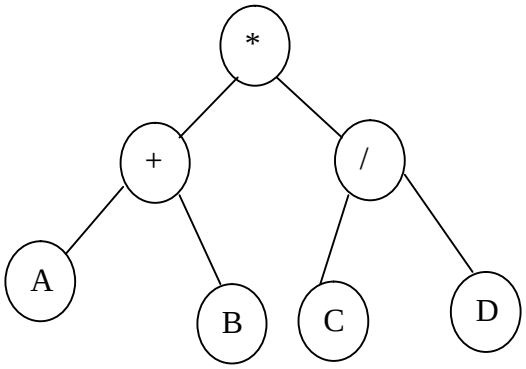
Example: - **Postorder Traversal.**

Rule: - Left – Right – **Node**



FGDEBCA

Example:- 2



Preorder Traversal: - Rule: - **Node** – Left – Right
* + A B / C D

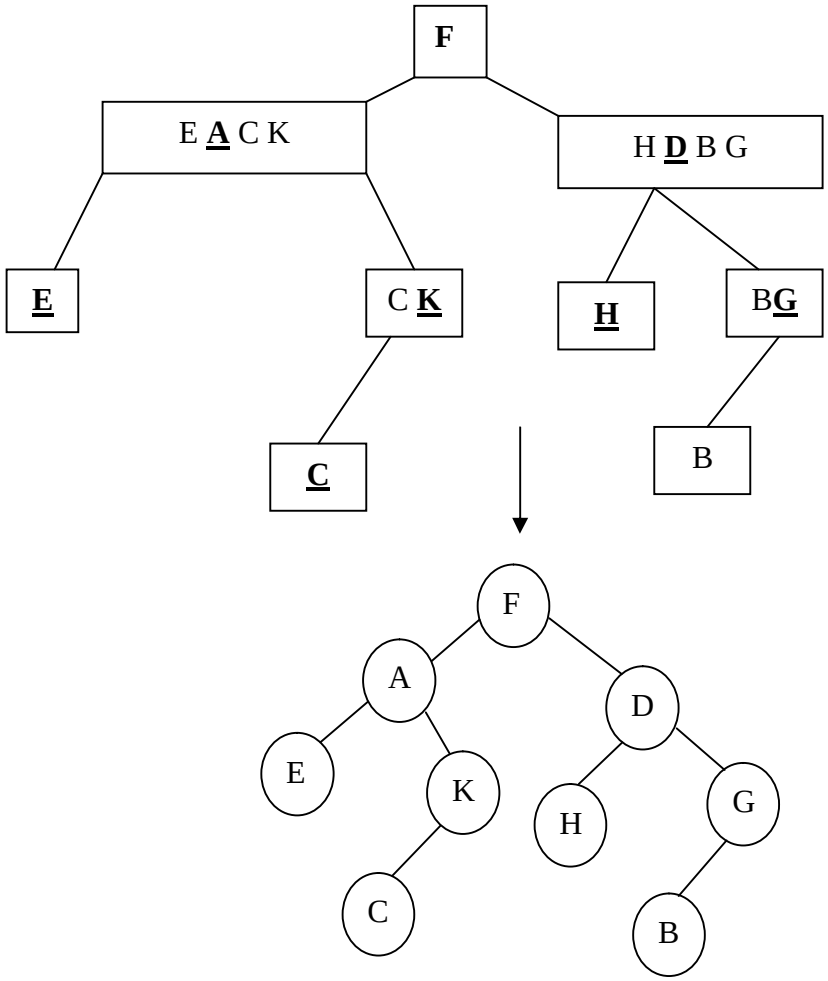
Inorder Traversal: - Rule: - Left – **Node** – Right
A + B* C / D

Postorder Traversal: - Rule: - Left – Right – **Node**
A B + C D / *

***Example**: - A binary tree **T** has **9 nodes**. The inorder and preorder traversals of yield the following sequence of nodes.

Inorder: - E A C K **F** H D B G

Preorder: - F A E K C D H G B



11. AVL-Tree

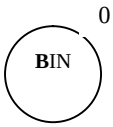
*AVL Tree (Adelson-Velskii and E.M. Lendis):-/Height Balanced Tree

Balancing Factor = Height of **Left** sub tree – Height of **Right** sub

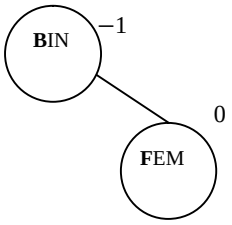
Example: - Make binary tree of following given order.

BIN	**F**EM	**I**ND	**N**EE	**L**AL	**P**RI	**J**IM	**A**MI	**H**EM	**D**IN

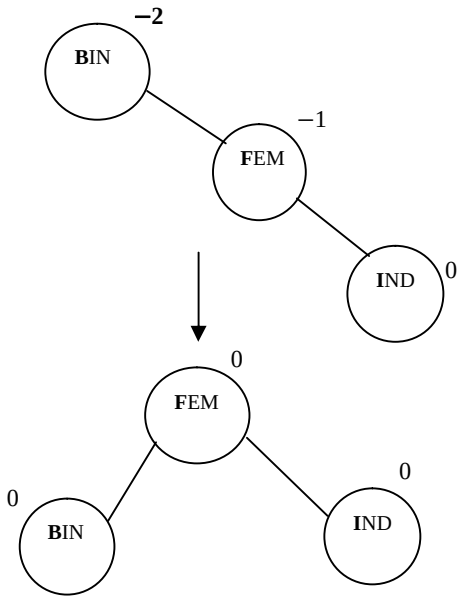
Step 1.



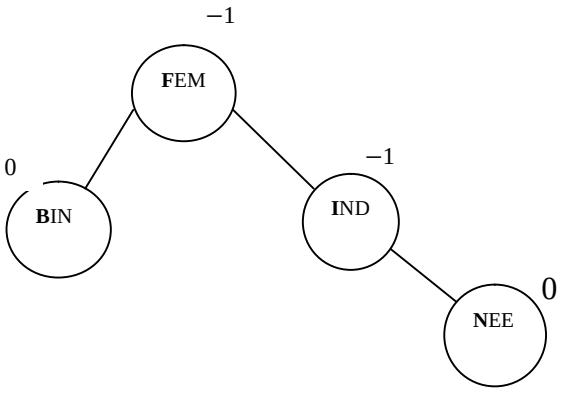
Step 2. Add FEM



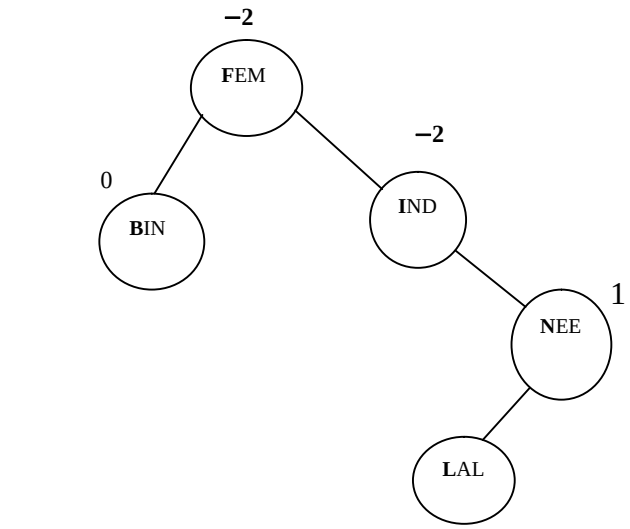
Step 3. Add IND

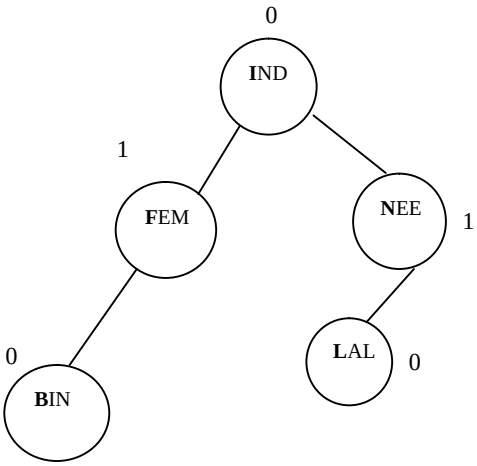


Step 4. Add NEE

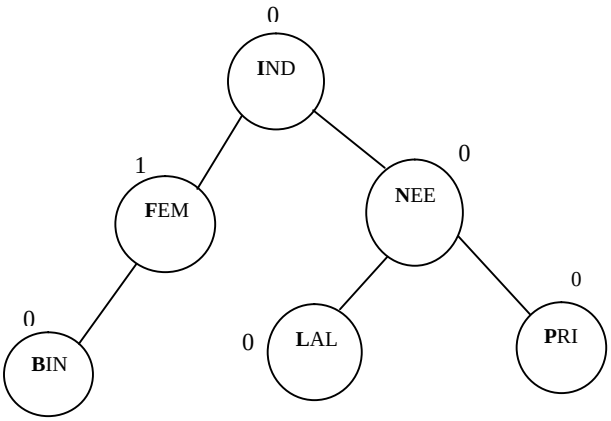


Step 5. Add LAL

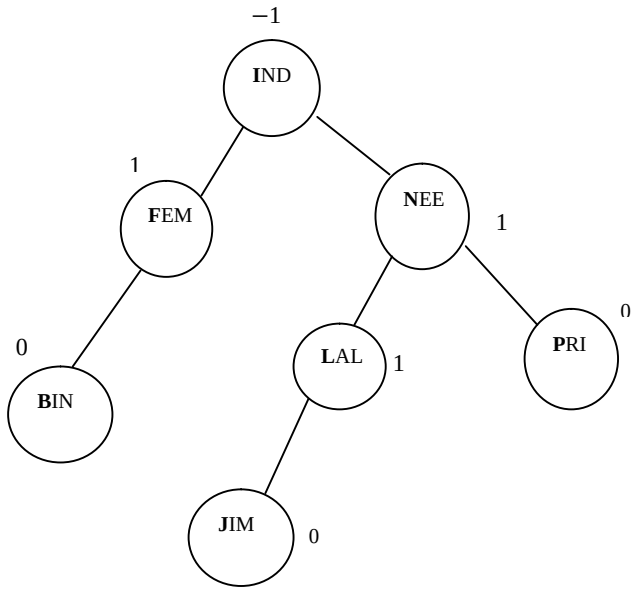




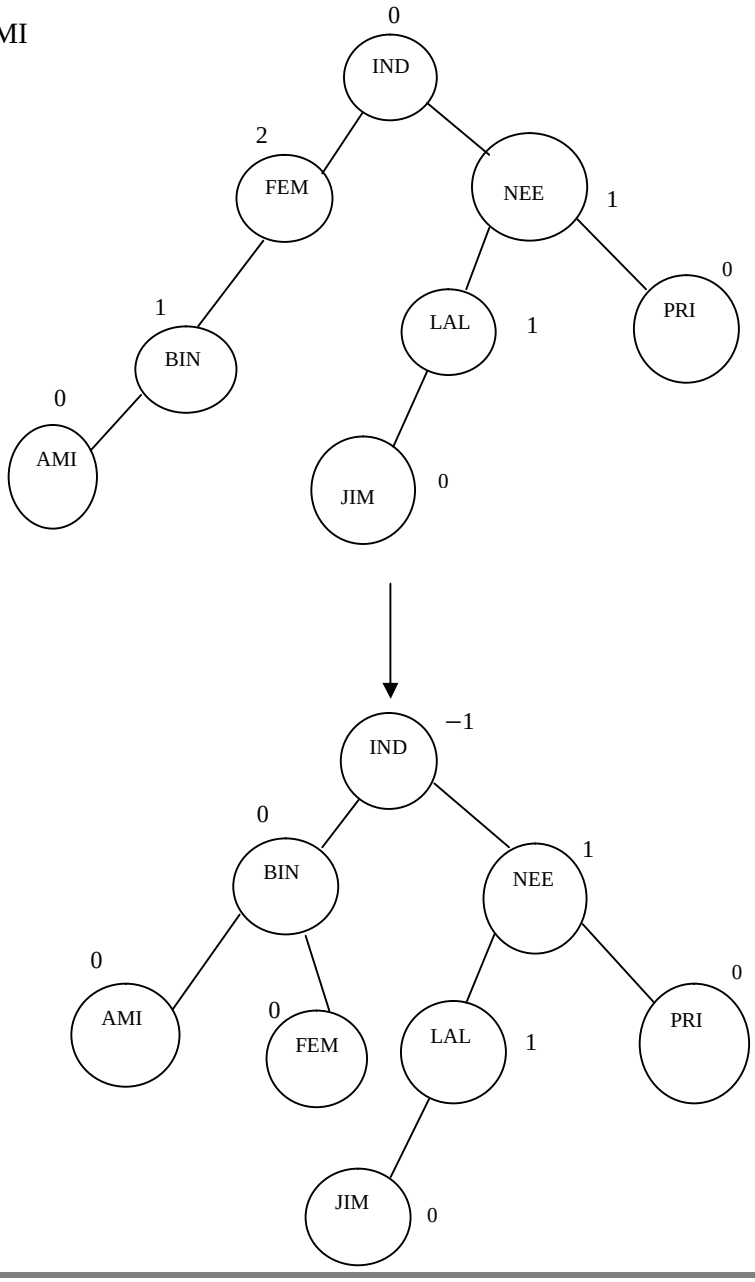
Step 6. Add PRI



Step 7. Add JIM

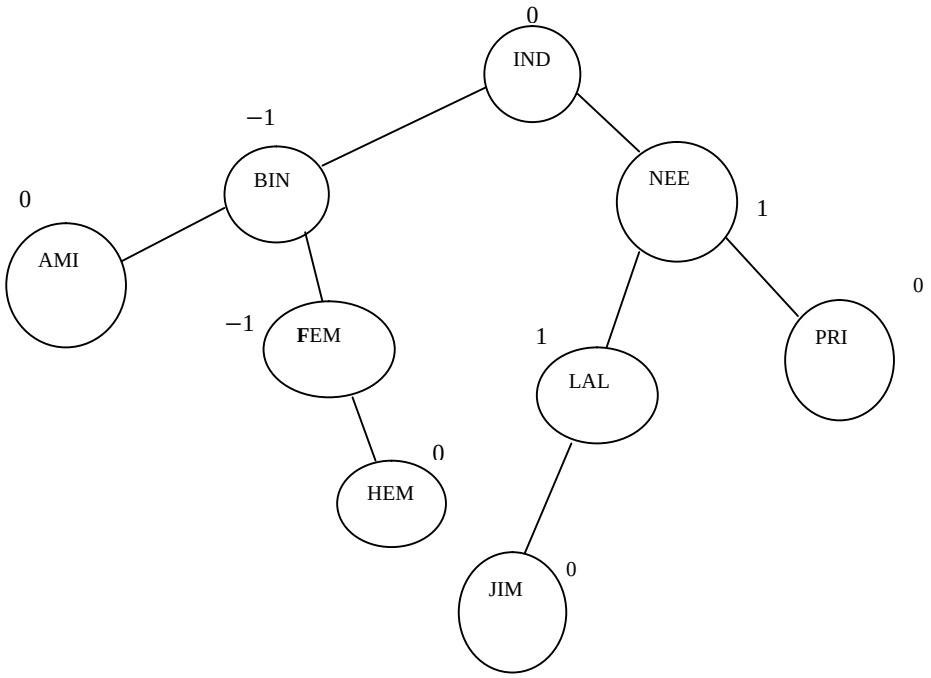


Step 8. Add AMI

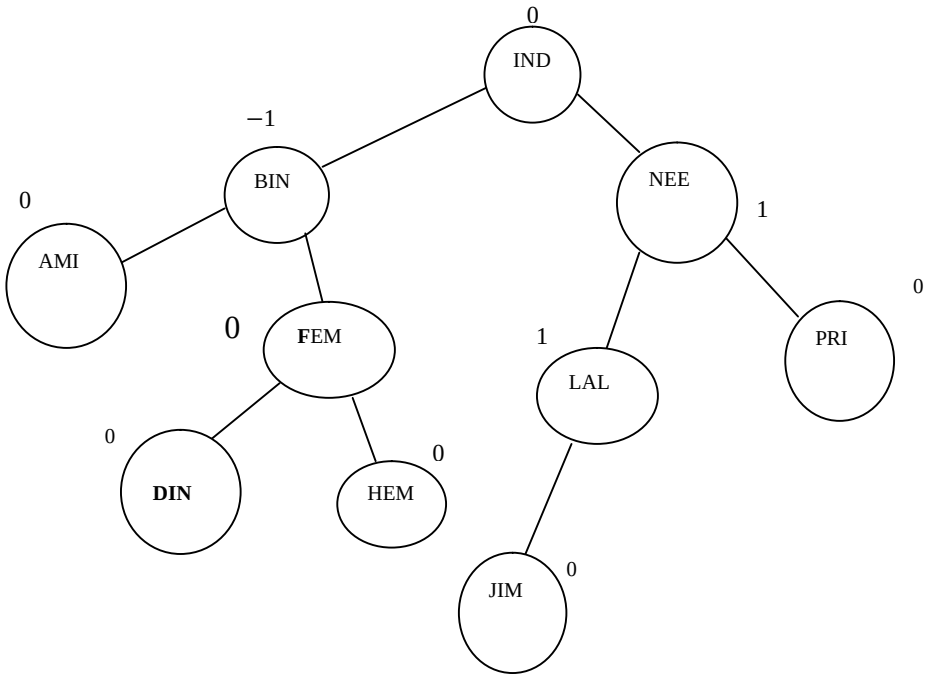


0

Step 9. Add HEM



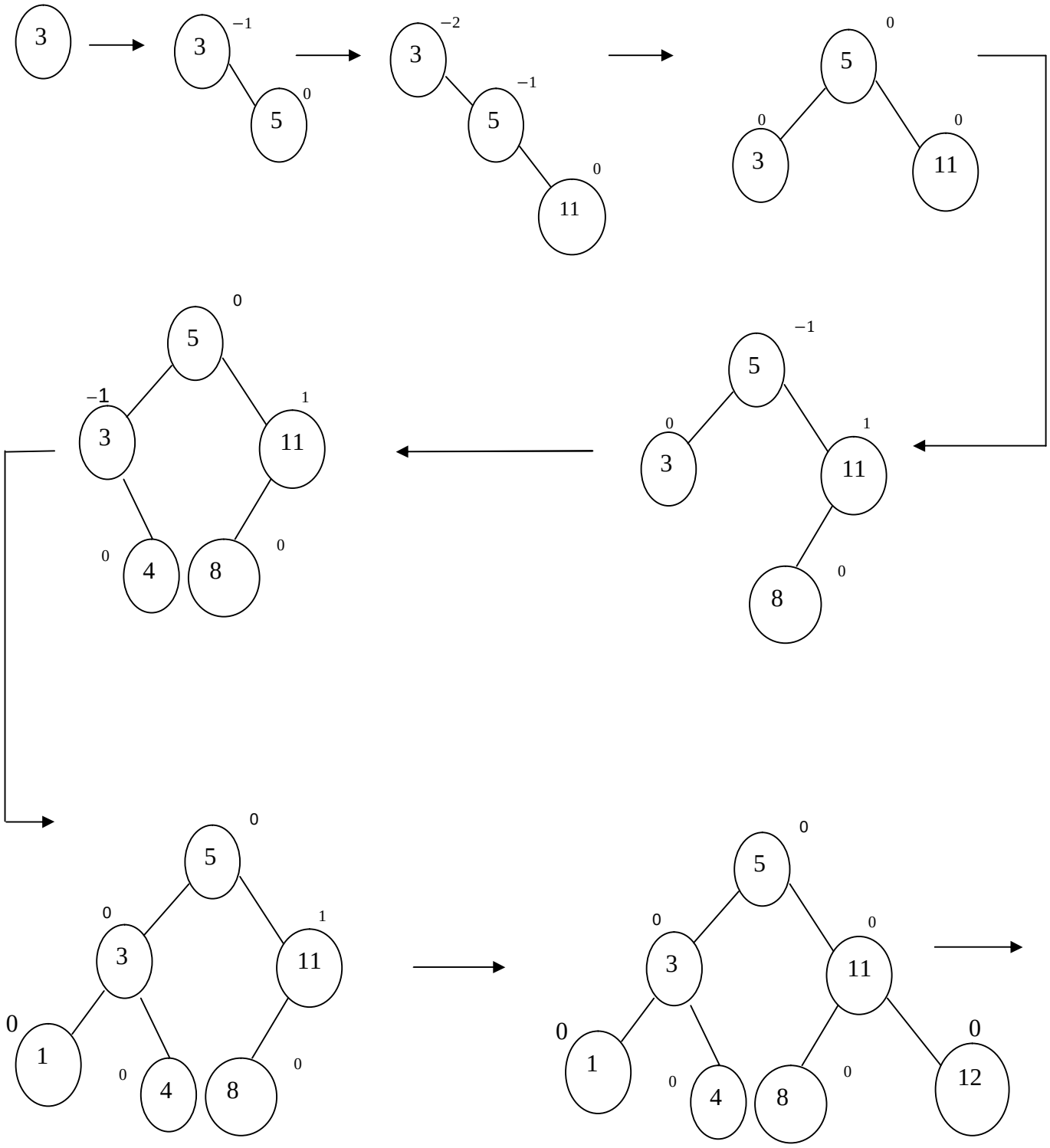
Step10. Add DIN

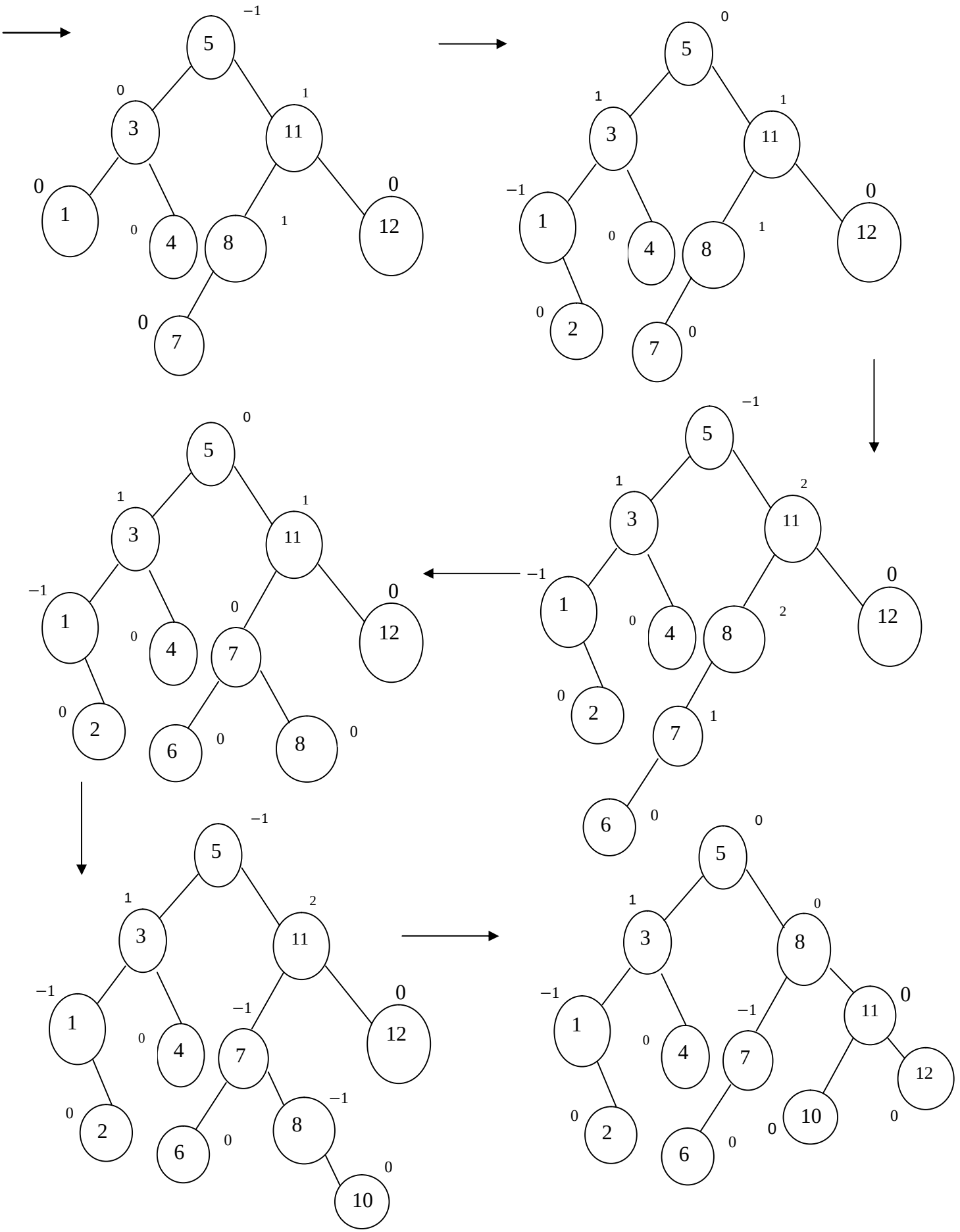


Example 2:- Consider the following list of Elements

Balancing Facto = Height of **Left** sub tree – Height of **Right** sub tree

3	5	11	8	4	1	12	7	2	6	10




5 −1
3 0
11 1
1 0
4 0
8 1
12 0
7 0
5 0
3 1
11 1
1 −1
4 0
8 1
12 0
2 0
7 0
5 −1
3 1
11 2
1 −1
4 0
8 2
12 0
2 0
7 1
6 0
5 0
3 1
11 1
1 −1
4 0
7 0
12 0
2 0
6 0
8 0
5 −1
3 1
11 2
1 −1
4 0
7 −1
12 0
2 0
6 0
8 −1
10 0
5 0
3 1
8 0
1 −1
4 0
7 −1
11 0
12 0
2 0
6 0
10 0

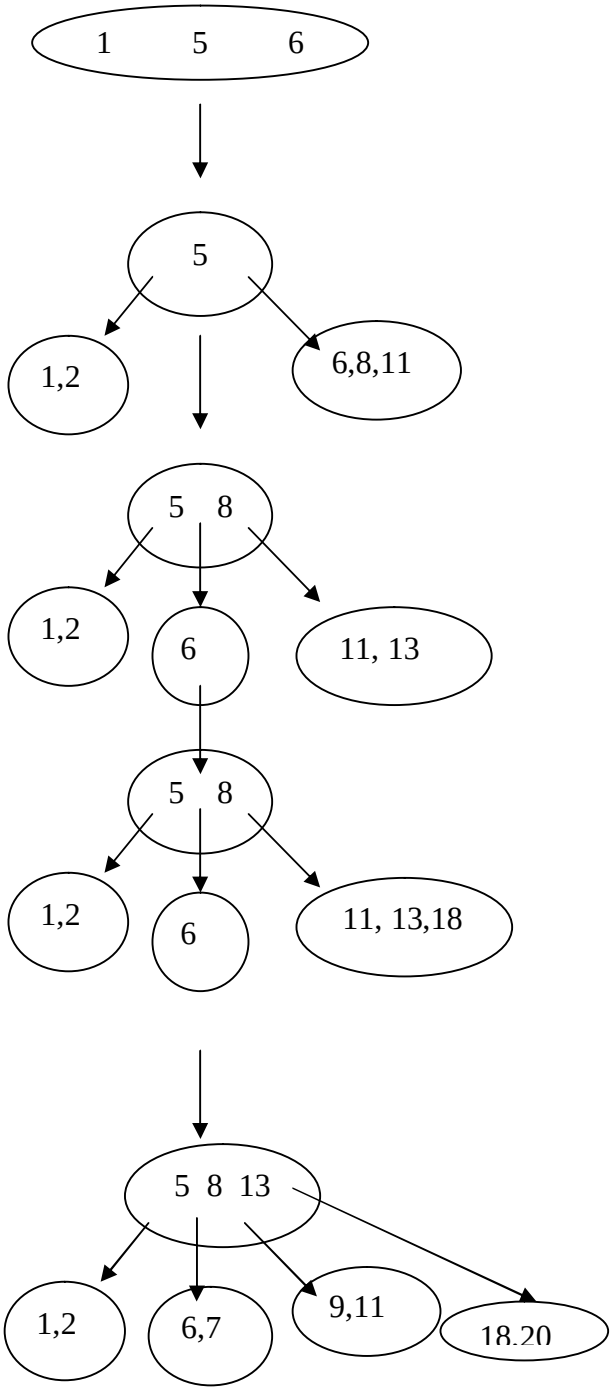
12. B-Tree

A **B-tree** is also known as **balanced sort tree**. It finds its use in external sorting. To **reduce disk accesses**, several **conditions** of the tree must be true.

- The height of the tree must be kept to a minimum.
- There must be no empty sub-trees above the leaves of the tree.
- The leaves of the tree must all be on the same level
- All nodes except the leaves must have at least some minimum number of children.

Example: - Build a **B-Tree** of **degree 4**

1	5	6	2	8	11	13	18	20	7	9



13. Graphs

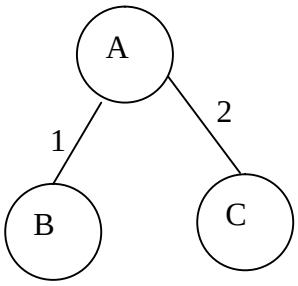
A **graph G consist** of **a set** V of vertices (nodes) and **a set** E of edges (arcs).We write

G = (V, E)

Where

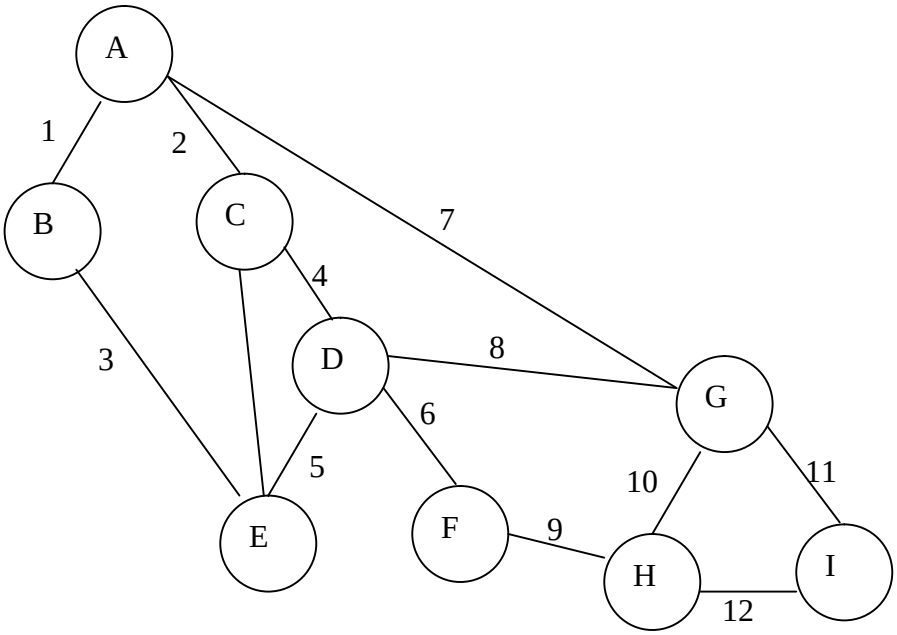
V is **finite** and **non empty** set of vertices.
E is **a set of pairs** of vertices, their pairs are called edges.

Example: -1



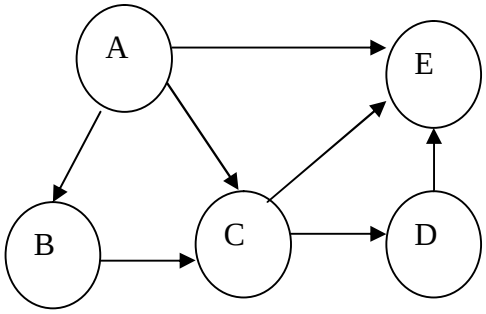
V= {A, B, C}
E= {1, 2}

Example: -2



V= {A, B, C, D, E, F, G, H, I}
E= {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}

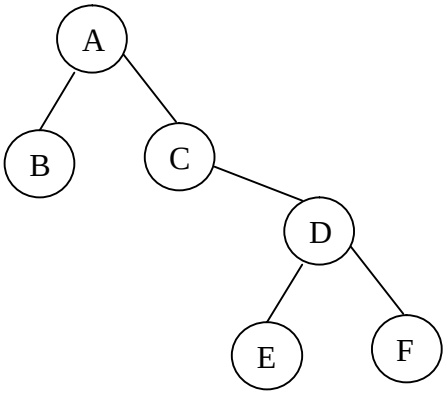
Directed Graph:-



Basic Terminology

1:- **Adjacent Vertices**: -

Example:-



- Adjacent vertex A are B and C
- Adjacent vertex C is D
- Adjacent vertex E is 0
- Adjacent vertex F is 0

2:- **Path**: - A path from vertex w is a sequence of vertices, each adjacent to the next.

Example 1:- A path from node '**A**' to '**F**' is

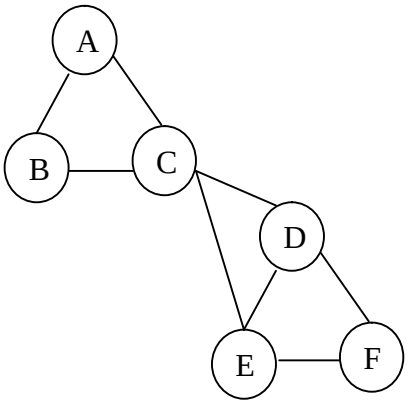
A→C→D→F

Example 2:- A path from node '**A**' to '**E**' is

A→C→D→E

3:- **Cycle**: -

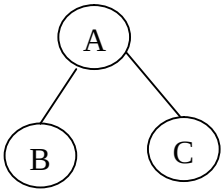
Example:-



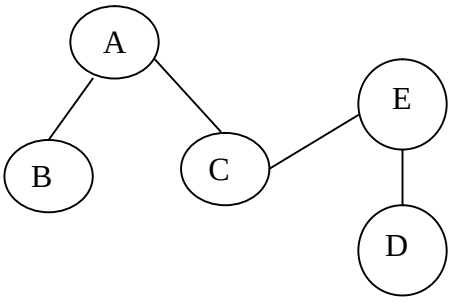
1: - A→B→C→A
2: - D→E→F→D
3: - C→E→F→D→C

4:- **Connected Graph**: -

Example 1:-



Example 2:-



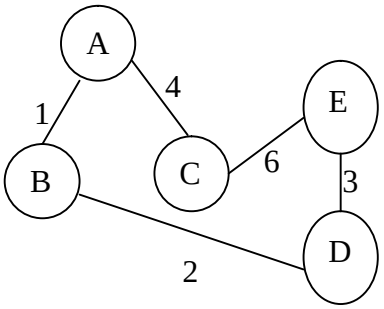
5:- **Degree**: - Number of edges **incident on a vertex** determine its degree. Degree of any vertex v is denoted by d (v).

Example: -

d(A)=2
d(E)=2
d(D)=1

6:- **Weighted Graph**: -

Example: -

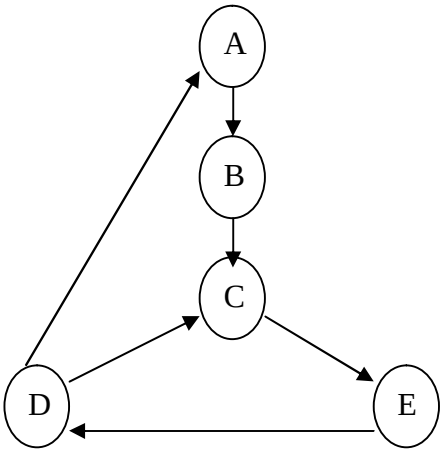


7: -**Tree**: - A **graph is a tree** *if* it has **two properties**.

- It is connected
- There is no cycle in graph.

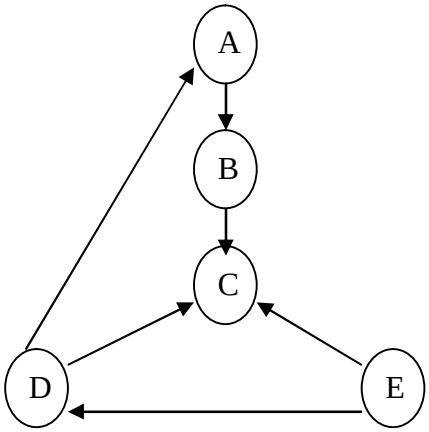
8:- **Strongly Connected Graph**: -

Example: -



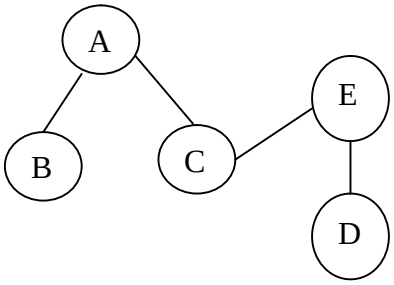
9:-**Weakly Connected Graph**: -

Example: -



Graph Representation in Matrix Form

Example:-



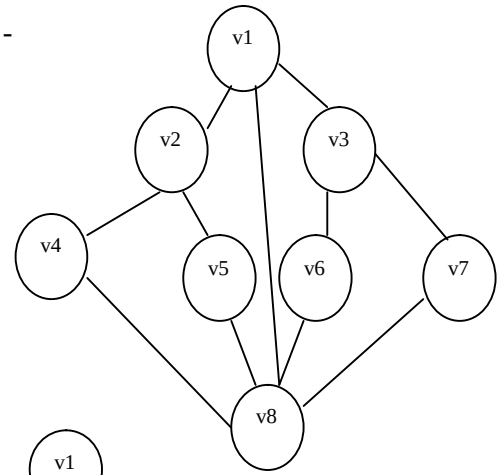
	A	B	C	D	E
A	0	1	1	0	0
B	1	0	0	0	0
C	1	0	0	0	1
D	0	0	0	0	1
E	0	0	1	1	0

Graph Traversal

1. **BFS** (Breadth first Search)
2. **DFS** (Depth First Search)

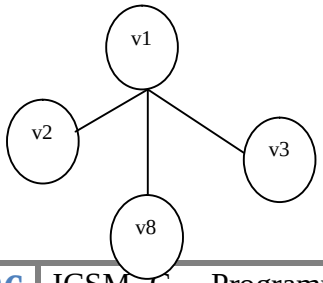
1. BFS (Breadth first Search):-

Example: -

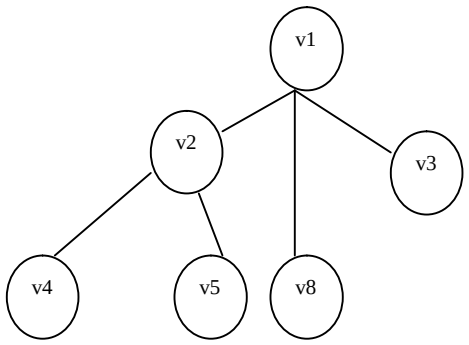


Step 1:-

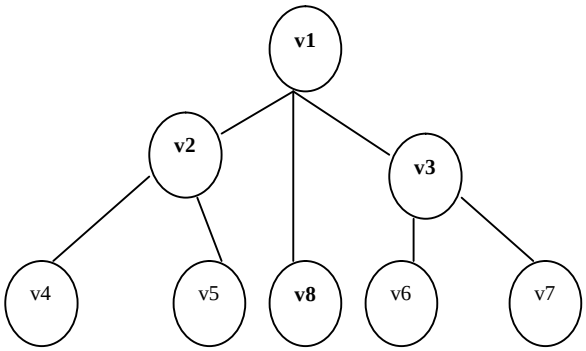
Step 2:-



Step 3:-



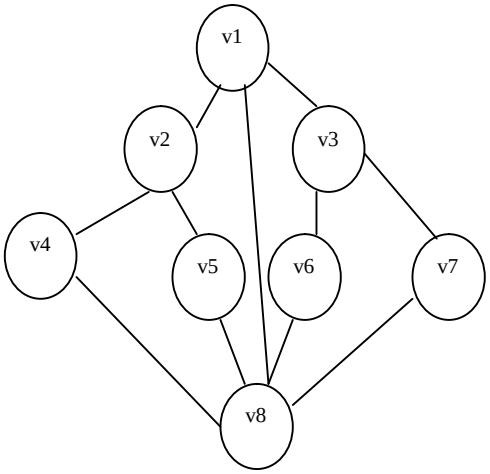
Step 4:-



***Traversal**: - V1→ V2→ V8→ V3→ V4→ V5→ V6→ V7

2. **DFS** (Depth First Search):-

Example: -



Step 1:- Start from **V1**
Its adjacent vertices are V2, V3, and V8

Step 2:- Start from **V2**
Its adjacent vertices are V1, V4, and V5

Step 3:- Start from **V4**
Its adjacent vertices are V2, V8

Step 4:- Start from **V8**
Its adjacent vertices are V1, V4, V5, V6, and V7

Step 5:- Start from **V5**
Its adjacent vertices are V2, V8

Step 6:- Start from **V6**
Its adjacent vertices are V3, V7, and V8

Step 7:- Start from **V3**
Its adjacent vertices are V1, V6, and V7

Step 8:- Start from **V7**
Its adjacent vertices are V3, V8

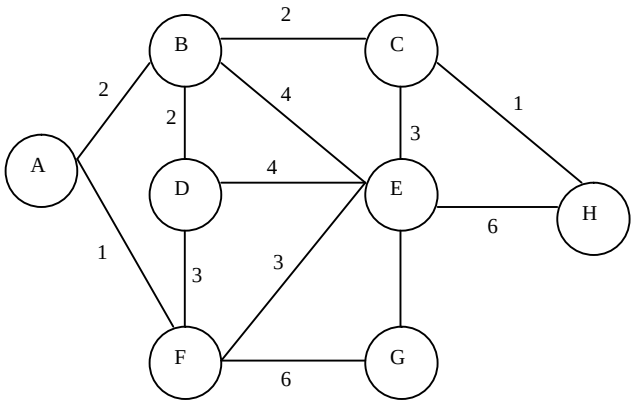
***Traversal**: - V1→V2→V4→V8→V5→V6→V3→V7

13. Minimal Spanning Tree :- (Shortest Path)

Minimal Spanning Tree⇨ There is two types

- ❖ Prims Algorithm
- ❖ Kruskals Algorithm

Example: - **Traverse** the following graphs **using Prims** and **Kruskals Algorithm**.

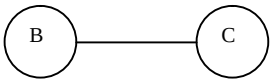


Method Prims Algorithm:-

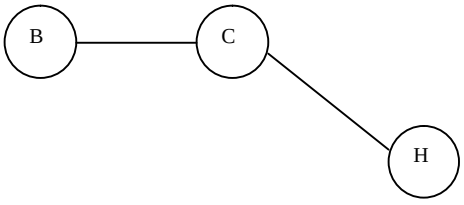
Step1:- Choose **any Vertex** in given graph (Consider **Vertex B**).

Adjacent of vertices B are A, D, E, C.
Weight (B→A) =2
Weight (B→D) =2
Weight (B→E) =4
Weight (B→C) =**2**

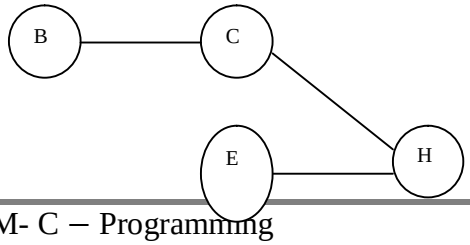
Choose the smallest weight edge. B→C



Step2:- Weight (C→E) = 3
Weight (C→H) = **1**



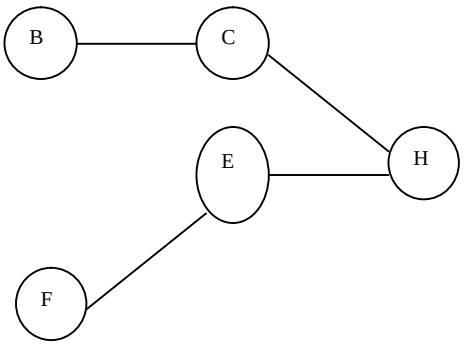
Step3:- Weight (H→E) = 6



Step4:- Weight attached through **E is**

Weight (E→D) = 4

Weight (E→F) = **3**

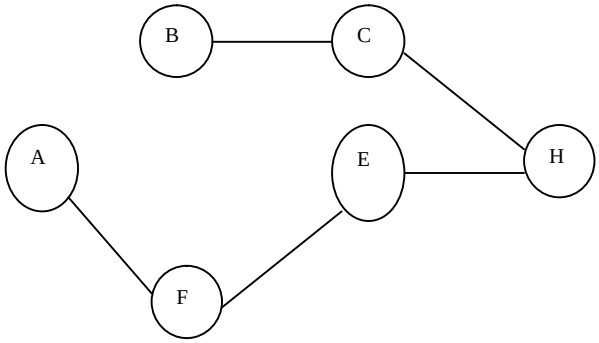


Step5:- Weight attached through **F is**

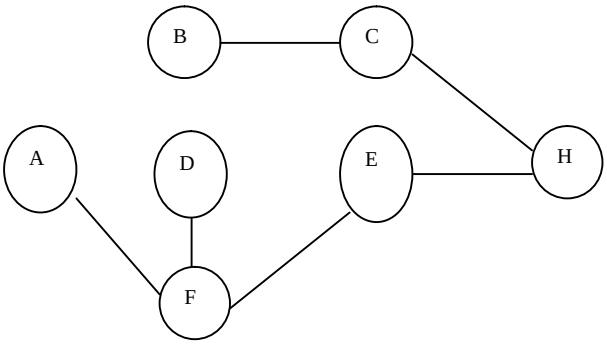
Weight (F→A) = **1**

Weight (F→D) = 3

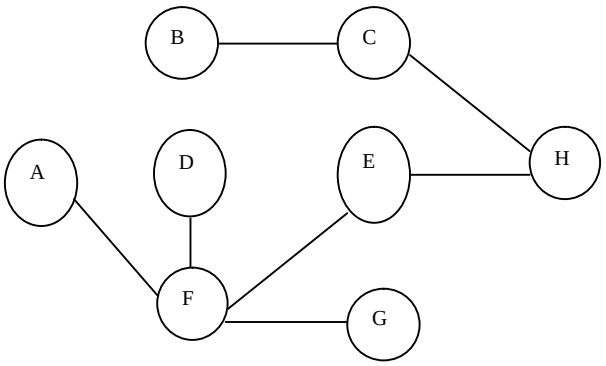
Weight (F→G) = 6



Step6:- Weight (F→D) = 3



Step7:- Weight (F→G) = 6



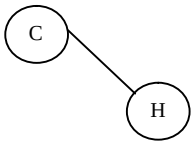
Minimal Spanning Tree (**Shortest Path**)

Method Kruskals Algorithm:-

Step1:- **Choose** the **smallest Vertex** in given graph

W (A→F) = 1

W (C→H) = **1**



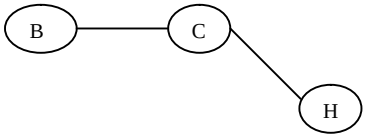
Step2:- Weight attached through **H is**

W (H→E) = 6

Weight attached through **C is**

W (C→B) = **2**

W (C→E) = 3

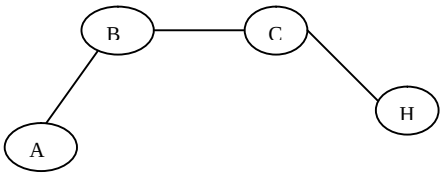


Step3:- Weight attached through **B is**

W (B→A) = **2**

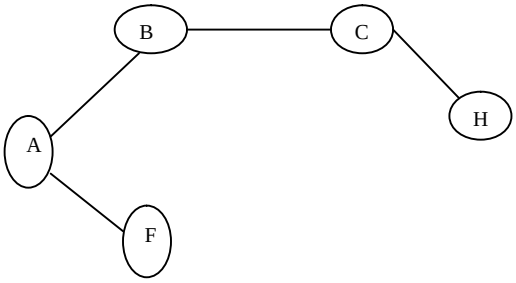
W (B→D) = 2

W (B→E) = 4



Step4:- Weight attached through A is

W (A→F) = **1**

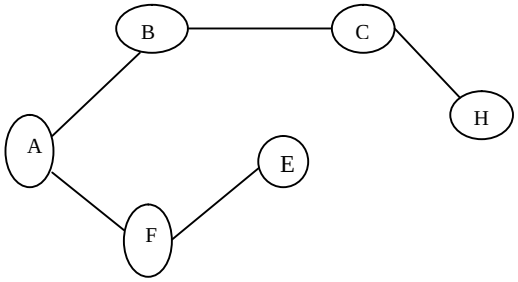


Step5:- Weight attached through **F is**

W (F→D) = 3

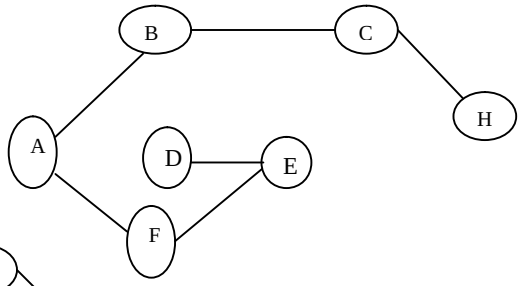
W (F→G) = 6

W (F→E) = **3**

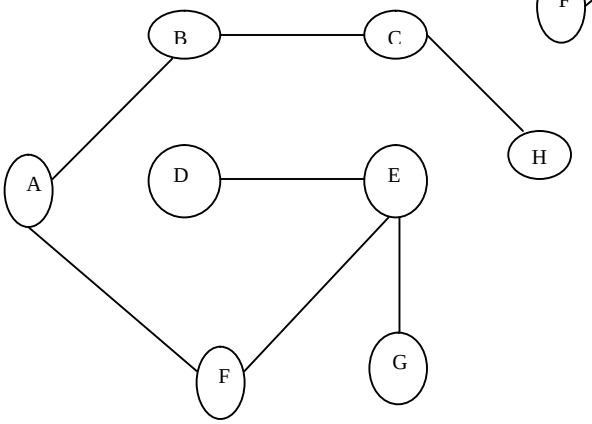


Step6:- Weight attached through **E is**

W (E→D) = **4**



Step7:- Traverse whose **weight is not** given



Minimal Spanning Tree (**Shortest Path**)

14. Stack & Queues

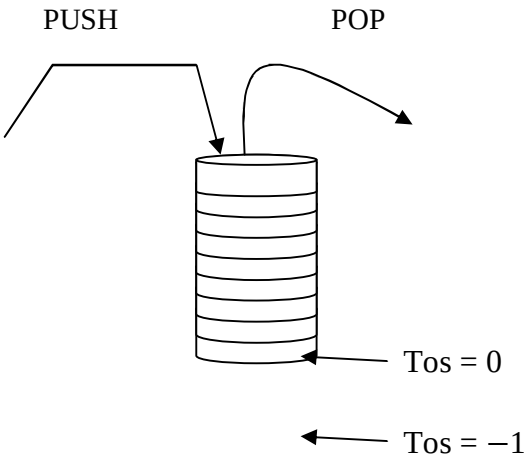
Stack

It is a **non-primitive linear data structure**. It is an ordered list in which **addition of new data item** and **deletion of already existing data item** is done from only one end known as **stack** (**TOS**). Stack is **also called LIFO** (Last-**In**-First-**Out**).

There are **two types** of operations are performed in Stack.

1. **PUSH** → (Insertion)
2. **POP** → (Remove)

Example:-



PUSH function In C:-

```
void push( )
{
 int item;
if(tos= = maxsize-1)
{
 printf ("\n The Stack is Full");
exit(0);
}
else
{
 printf("\n Enter the element to be inserted");
scanf("%d",&item);
tos=tos+1;
stack[tos]=item;
}
}
```

POP function In C:-

```
int pop()
{
 int item;
if(tos= =−1)
{
 printf("The Stack is empty");
```

```
        exit(0);
        }
else
        {
         item=stack[tos];
        tos=tos−1;
        }
        return(item);
        }
```

Application Of Stack:-

- ❖ Infix notation
- ❖ Prefix Notation
- ❖ Postfix notation

Method: - We use **BODMAS**

B	**Bracket**
O	Of //
D	Division
M	Multiplication
A	Addition
S	Subtraction

Priority:-

^ Highest Priority

*, / ↓

+, − Least Priority

Example:-1 Express following **infix notation** into **prefix** and **postfix notation**.

Infix notation

A+B*C

Prefix Notation:-

A + (B*C)

⇒ A + (*BC)

⇒ +A (*BC)

⇒ +A *BC

Postfix Notation:-

A + (B*C)

⇒ A + (BC*)

⇒ A (BC*) +

⇒ A BC*+

Example:-2 Express following **infix notation** into **prefix** and **postfix notation**.

(A*B)+C/D Infix notation

Prefix Notation:-

(A*B)+C/D

⇒ (*AB)+(/CD)

⇒ + (*AB) (/CD)

⇒ + *AB /CD

Postfix Notation:-

(A*B)+C/D

⇒ (AB*)+(CD/)

⇒ (AB*) (CD/)+

⇒ AB* CD/+

Example:-3 Express following **infix notation** into **prefix** and **postfix notation**.

A+B/C−D Infix notation

Postfix Notation:-

A+B/C−D

⇒ A+(BC/)−D

⇒ (A (BC/)+)−D

⇒ (A (BC/)+)D−

⇒ A BC/+D−

Prefix Notation:-

A+B/C−D

⇒ A+(/BC) −D

⇒ (+A (/BC)) −D

⇒ − (+A (/BC)) D

⇒ −+A /BCD

Example:-4 Express following **infix notation** into **prefix** and **postfix notation**.

(A+B)*C/D+E^F/G Infix notation

Postfix Notation:-

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

⇒ (AB+)*(C/D)+(EF^)/G

⇒ (AB+)*(CD/)+((EF^)G/)

⇒ ((AB+) (CD/)*)+((EF^)G/)

⇒ ((AB+) (CD/)*) ((EF^)G/)+

⇒ AB+CD/*EF^G/+

Prefix Notation:-

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

⇒ (+AB)*(C/D)+(^EF)/G

⇒ (+AB)*(/CD)+(/(^EF)G)

⇒ (*(+AB/CD))+(/^EFG)

⇒ + *+AB/CD/^EFG

Example:-5 Express following **infix notation** into **prefix** and **postfix notation**.

A/B^C-D **Infix notation**

Postfix Notation:-

(A/(BC^))-D

⇒ ((A (BC^)/)-D

⇒ A BC^/D-

Prefix Notation:-

(A/(^BC))-D

⇒ (/A^BC)-D

⇒ -/A^BCD

Example:-6 Express following **infix notation** into **prefix** and **postfix notation**.

Not Highest Priority

And ↓

Or Lowest Priority

((NOT A) OR (NOT B) AND (NOT C))

Prefix Notation:-

((A NOT) OR (B NOT) AND (C NOT))

⇒ ((A NOT) OR (AND (B NOT) (C NOT)))

⇒ OR ((A NOT) (AND (B NOT) (C NOT)))

⇒ OR A NOT AND B NOT C NOT

Postfix Notation:-

((NOT A) OR (NOT B) AND (NOT C))

⇒ ((NOT A) OR ((NOT B) (NOT C) AND))

⇒ ((NOT A) ((NOT B) (NOT C) AND) OR))

⇒ NOT A NOT B NOT C AND OR

Queues

It is a **non primitive linear data structure**. It is **homogeneous collection of elements** in which **new elements** are **added** at **one end** called the **Rear end**, and the **existing elements** are **deleted from other end** called **Front end**. A queue is logically a First In First Out (**FIFO**).

Example: - **Insertion** of elements in queue.

Rear = Rear + 1

F = −1, R = −1

F = 0, R = **0**

3					

F = 0, R = **1**

3	5				

F = 0, R = **2**

3	5	7			

F = 0, R = **3**

3	5	7	9		

F = 0, R = **4**

3	5	7	9	11	

F = 0, R = **5**

3	5	7	9	11	13

Example: - **Deletion** of elements in queue.

Front = Front + 1

F = 0, R = **3**

	5	7	9	11	13

F = 1, R = **5**

		7	9	11	13

F = 2, R = **7**

			9	11	13

F = 3, R = **9**

				11	13

F = 4, R =11

					13

F = 5, R = **13**

Function for Insertion in Queue:-

```
void insert(int item)
{
if (rear>max_size)
{
Printf("Queue is over flow");
break;
}
else
{
rear = rear + 1;
queue(rear) = item;
}
}
```

Function for Deletion in Queue:-

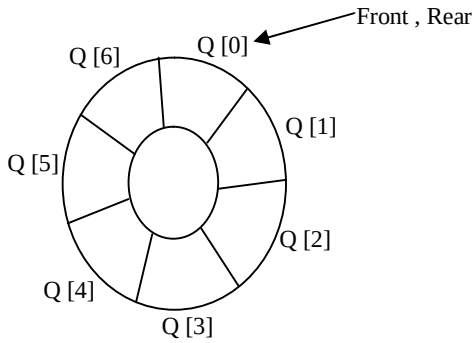
```
void del()
{
if(front<=0)
printf("\n Queue is empty");
break;
}
else
{
item = queue(front);
front = front + 1;
printf("Deleted item = %d",item);
}
}
```

Types of Queue ⇨ There are two types. **1.** Circular Queue **2.** Double Ended Queue

1. Circular Queue: - In circular queue **the insertion** of a new element is **done** at the very **first location** of the queue, if the last location of the queue is **full.**

Assumption:-

a) **Front** will **always be pointing** to the **first element**.
b) If **front** = **rear** then **queue will be empty**.
c) Each time a **new element is inserted** into the **queue**. The rear is **incremented** by one. **Rear = Rear + 1**
d) Each time an **element is deleted from** the queue. The value of front **is incremented** by one. **Front = Front + 1**



Logic for adding **element in circular Queue:-**

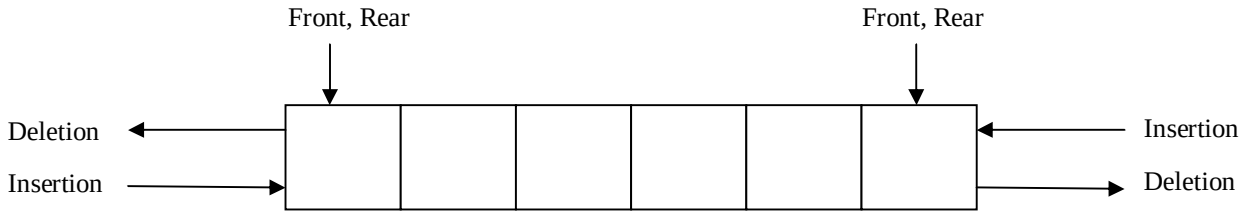
Rear = (Rear + 1)% max_size

Queue [Rear] = Value

Logic for deletion **element in circular Queue:-**

Front= (Front + 1) % max_size

2. Double Ended Queue: - It is also a homogeneous list of element in which **insertion** and **deletion operation** is performed **from both the ends**.



Function of Double Queue:-

(a) **Insertion function:-**

```
dqinsert(int q[15], int front, int rear, int item)
{
if(front==0)
{
printf("Queue is full");
return;
}
front = front – 1;
q[front] = item;
}
```

(b) **Deletion function:-**

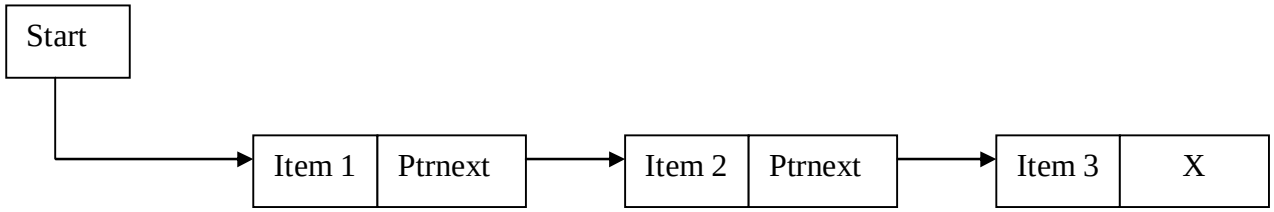
```
dqdelete (int q[15], int front, int rear, int item)
{
```

```
if(front==rear)
{
printf("Queue is empty");
return;
}
rear = rear – 1;
item = q[item];
}
```

15. Lists

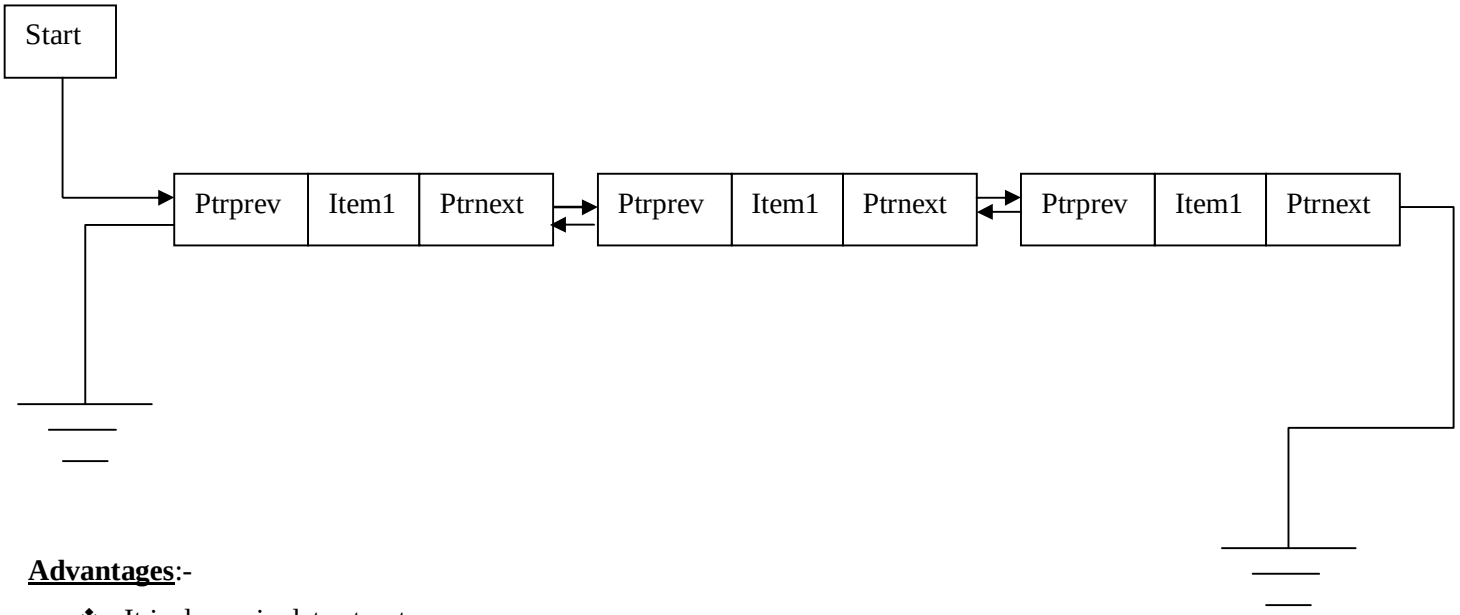
Linked List ⇨ It is **special list** of some data elements **linked** to one another.

Example 1:-



X → Null Pointer

Example 2:-



Advantages:-

- ❖ It is dynamic data structure.
- ❖ Efficinent memory utilization
- ❖ Insertion and deletions are easier and efficient.

Disadvantages:-

- ❖ More Memory
- ❖ Access to an arbitrary data item is little bit cumbersome and also time consuming

Applications Of Linked list:-

- ❖ Linked Lists can be used to implement Stacks , Queues.
- ❖ Linked Lists can also be used to implement Graphs. (Adjacency list representation of Graph).
- ❖ Implementing Hash Tables :- Each Bucket of the hash table can itself be a linked list. (Open chain hashing).
- ❖ Undo functionality in Photoshop or Word . Linked list of states.
- ❖ A polynomial can be represented in an array or in a linked list by simply storing the coefficient and exponent of each term.
- ❖ However, for any polynomial operation , such as addition or multiplication of polynomials , linked list representation is more easier to deal with.

- ❖ Linked lists are useful for dynamic memory allocation.
- ❖ The real life application where the circular linked list is used is our Personal Computers, where multiple applications are running.

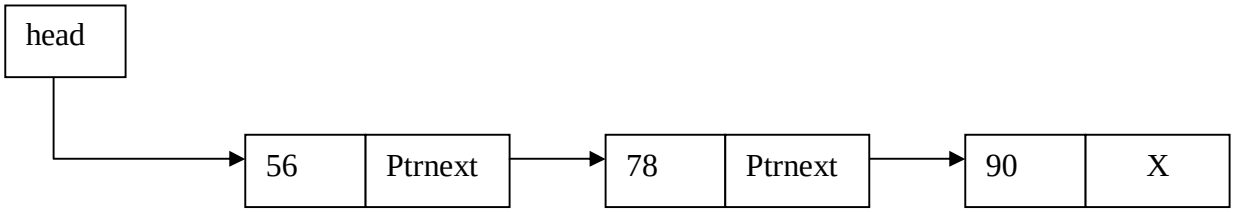
Representation of Linear Linked list:-

```
struct node
{
int a;
struct node *Ptrnext;
};
typedef struct node Node_Obj;
Node_Obj *start;
```

Type of Linked list:-

- ❖ Singly Linked list
- ❖ Doubly Linked List
- ❖ Circular Linked List
- ❖ Circular Doubly Linked list

1. Singly Linked List: - It is also called **linear linked list**. It is represented as.



Structure definition of Singly Linked list:-

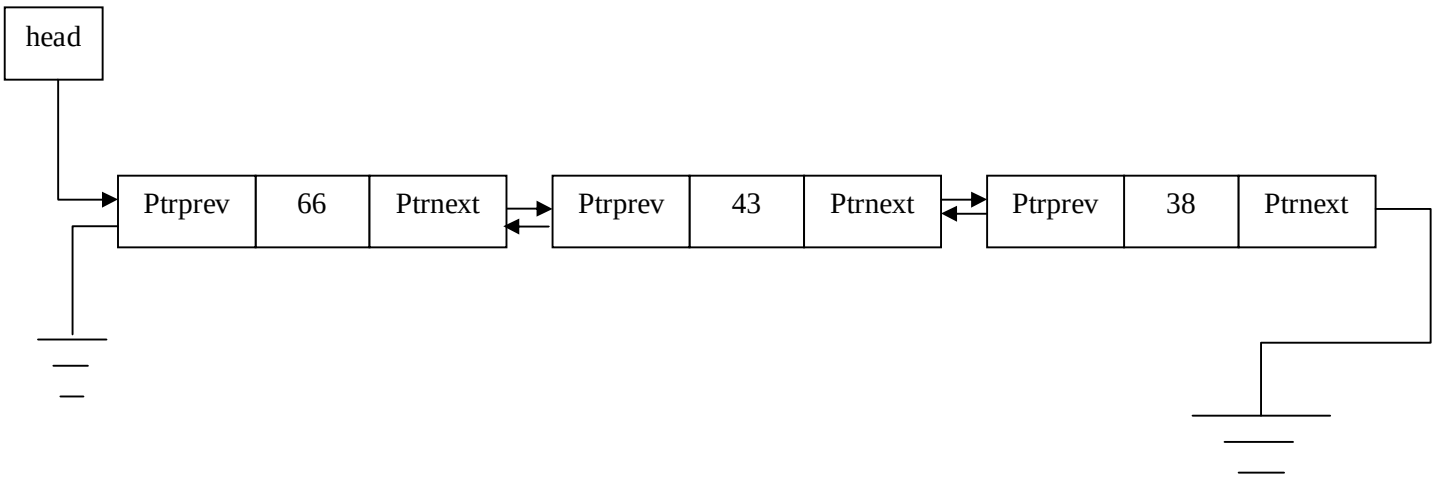
```
struct node
{
int num;
struct node *Ptrnext;
}
typedef struct node obj_node;
obj_node *head;
head=( obj_node *) malloc(size(obj_node));
```

Q. Write a program. to create a list-

```
#include<stdio.h>
#include<conio.h>
struct node
{
int data;
struct node *next;
}*start=NULL;
void create()
{
char ch;
 do
 {
  struct node *new_node,*current;
  new_node=(struct node *)malloc(sizeof(struct node));
```

```
  printf("\nEnter the data : ");
  scanf("%d",&new_node->data);
  new_node->next=NULL;
  if(start==NULL)
  {
  start=new_node;
  current=new_node;
  }
  else
  {
  current->next=new_node;
  current=new_node;
  }
 printf("\nDo you want to creat another : ");
 ch=getche();
 }while(ch!='n');
}
void display()
{
struct node *new_node;
 printf("The Linked List : n");
 new_node=start;
 while(new_node!=NULL)
{
   printf("%d--->",new_node->data);
new_node=new_node->next;
   }
  printf("NULL");
}
 void main()
{
 create();
 display();
}
```

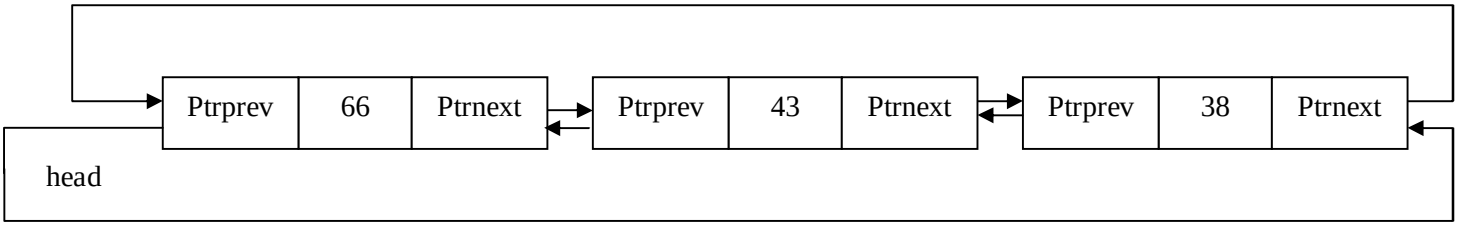
2. <u>Doubly Linked List</u>:-



Structure definition of Doubly Linked list:-

```
struct node
{
 int num;
 struct node *Ptrnext;
 struct node *Ptrprev;
};
typedef struct node obj_node;
```

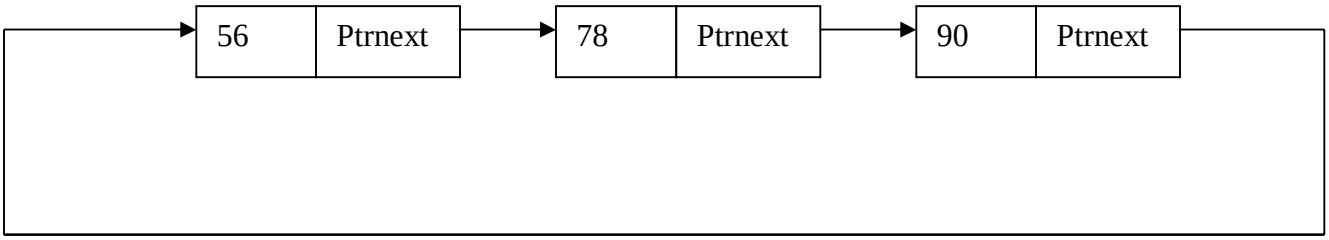
3. Circular Doubly Linked List:-



Structure definition of Circular Doubly Linked list:-

```
NODE *create_list()
{
NODE *head;
head=allocate_node();
head->LEFT=head->RIGHT=head;
head->num=0;
return head;
}
```

4. Circular Linked List:-



Operation on Linked list:-

- Creation of Linked list.
- Insertion of linked list.
  - At the beginning of linked list.
  - At the end of linked list.
  - At the specified position of linked list.
- Deletion of node from linked list.
  - Beginning of linked list.
  - End of linked list.
  - Specified position in the linked list.
- Traversing.
- Searching.
- Concatenation.
- Display.

Searching of Array Element

It is **used to find** the **location** where element is available or not. There are **two types** of searching techniques as given below.

- Linear or sequential search
- Binary Search

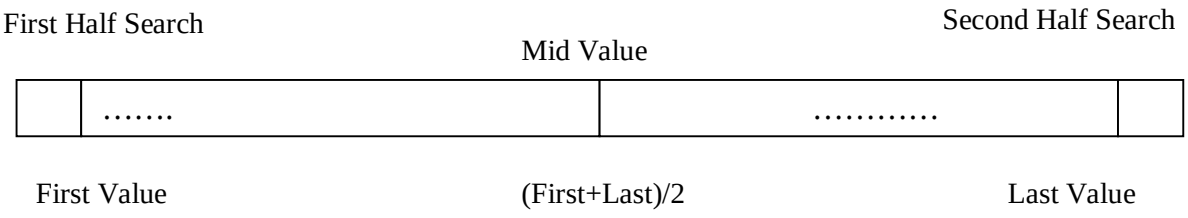
1. Linear or sequential search:-

```
#include<stdio.h>
main()
{
int m[50],n,i,item,loc=-1;
printf("\n Enter the number of element=");
scanf("%d",&n);
printf("\n Enter the numbers:\n");
for(i=0;i<=n-1;i++)
{
scanf("%d",&m[i]);
}
printf("\n Enter the number to be searched:");
scanf("%d",&item);
for(i=0;i<=n-1;i++)
{
if(item==m[i])
{
loc=i;
break;
}
}
if(loc>=0)
printf("\n%d is found in position %d",item,loc+1);
else
printf("\n Item does not exist");
}
```

2. Binary Search:-

It is an extremely efficient algorithm. This search technique searches the given item in **minimum possible comparisons**. To do binary search, **first we had to sort the array elements**. The logic behind this technique is given below:-

- First, find the middle element with an item.
- Compare the mid element with an item.
- There are **three cases**
  - if it is a desired element the search is successful
  - if it is less than desired item then search only the first half of the array.
  - if it is greater than the desired element search in the second half of the array



Program:-

```
#include<stdio.h>
main()
{
int arr[20], start, end, middle, n, i, item;
printf(" How many elements you want to enter in the array:");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("Enter element %d:",i+1);
scanf("%d",&arr[i]);
}
printf("Enter the element to be searched : ");
scanf("%d",&item);
start = 0;
end = n-1;
middle = (start+end)/2;
while(item!=arr[middle] && start<=end)
{
if(item>arr[middle])
start = middle+1;
else
end = middle-1;
middle = (start + end)/2;
}
if(item ==arr[middle])
{
printf("%d found at position %d\n",item,middle+1);
}
if(start>end)
printf("%d not found in array\n",item);
}
}
```

16. Files

It **deals** with storage and retrieval of data in a computer.

Terminology:-

1. **Field**: - It is an elementary data item characterized by its size, length and types.

   Example:-

Ename char[15]	:	Character type of size 15
int age	:	A numeric type

2. **Record**: - It is a **collection** of related fields that can be treated as unit from an application point of view.

   Example:-

Empno	**Ename**	**Age**	**Job**	**Sal**	**Deptno**
101	Amit	30	Steno	8000	10
102	Sudeep	32	Clerk	9000	20

3. **File**:- Data is organized for storage in files. A file is collection of similar, related records. It has identifying name.

4. **Index**: - It corresponds to a data file. Its records contain a key field and a pointer to that record of the data files which has the same value of the key field.

File Organization: -

The factor involved in selecting a particular File organization for uses are:

- Ease of retrieval
- Economy of storage
- Reliability
- Security
- Integrity

Types of File organization:-

- Sequential Files
- Relative Files
- Direct Files
- Indexed Sequential Files
- Index Files

1. **Sequential Files**: -Data records are stored in some specific sequence, that is, order of arrival value of key field. It cannot access randomly.

2. **Relative Files**: - Each data record has a fixed place in relative file. Each record must associated with it in an integer key value that will help identify this slot.

3. **Direct Files**: - These are similar to relative files, except that the key value need not be an integer. The user can specify keys which make sense to his application.

4. **Indexed Sequential Files**: - An index is added to the sequence file to provide random access. An overflow area needs to be maintained to permit insertion in sequence.
5. **Indexed Files: -** In this organization, no sequence is imposed on the storage of records in the data file, therefore, no overflow area is needed.

File Operation:-

- Creation of File
- Reading of File
- Updation of File
- Insertion in the file
- Deletion from file

Step In Processing Files

1. Opening a files **2.** Reading from or Writing onto a file **3.** Closing the file

1. Syntax for opening a file:-

File opening mode

Mode	Purpose
rt	**Open** a text file **for reading**. The file must **already exist**.
wt	Open a text file **for writing**, if file already exist. It contents will be discarded, if it does exist at will be created.
at	Open a text file **for extending**. Data will be **added** to the end of existing file. **If** it does not exist, it will be create.
rb	Open a **b**inary file for **r**eading. The file **must** already exist.
wb	Open **b**inary file for **w**riting.
ab	Open **b**inary file for **e**xtending.
rt+ or **r+t**	Open a text file for **both** reading & writing. The file **must** exist.
wt+ or **w+t**	Open a text file for **both** reading & writing. **If** it does not exist, it will be created.

How to open successful file in program:-

```
fptr=fopen("Name of file","Mode");
if(fptr=NULL)
{
printf("\n It can not open Defined file by user");
exit (1);
}
```

Or

```
if (fptr=fopen("Name of file","Mode")=NULL)
{
printf("\n It can not open Defined file by user");
exit(1);
}
```

2. Syntax for closing a file:-

```
fclose(fptr);
```

3. Reading and Writing of File:-

File Input/Output Function:-	fget**c**(), fput**c**()	**C**haracter input/output function
	fget**s**(), fput**s**()	**S**tring input/output function
	get**w**(), put**w**()	**I**nteger input/output function
	fscan**f**(), fput**f**()	**F**ormatted input/output function
	fread(), fwrite()	**R**ecord input/output function

(A) Reading and Writing Using character input/output function:-

(i) Syntax for writing one character at a time- fput**c**(ch,fptr)

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
main()
{
FILE*fptr;
char str;
if((fptr=fopen("d:\Patel.txt", "at"))==NULL)
{
printf("\n It can not open Defined file by user");
//exist(1);
}
printf("\n Enter character in file=");
while((str=getche())!='\r')
fputc(str,fptr);
fclose(fptr);
}
```

(ii) Syntax for reading one character at a time- **ch**=fget**c**(fptr);

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
main()
{
FILE*fptr;
char str;
if((fptr=fopen("d:\Patel.txt", "r"))==NULL)
{
printf("\n It can not open Defined file by user");
//exist(1);
}
printf("\n Enter character in file=");
while((str=fgetc(fptr))!=EOF)
putchar(str);
fclose(fptr);
}
```

(B) Reading and Writing using String I/O functions:-

(i) Syntax for writing to a file:- fput**s**(str,fptr)

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
main()
{
FILE *fptr;
char str[90];
if((fptr= fopen("Patel.txt","at"))==NULL)
        {
 printf("\n\n It can not open Defined file by user");
```

```
//exit(1);
}
printf("\n Enter a set of text string=");
while(strlen(gets(str))>0)
{
fputs(str,fptr);
fputs("\n",fptr);
}
 fclose(fptr);
}
```

(ii) Syntax for <u>reading</u> from a file:- fgets(str, n, fptr)

Where n is the maximum length of the input string.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
main()
{
 FILE *fptr;
char str[90];
if((fptr= fopen("Patel.txt","r"))==NULL)
{
printf("\n\n It can not open Defined file by user");
//exit(1);
}
printf ("\n Text string from existing file=\n\n");
while(fgets(str,90,fptr)!=NULL)
 puts(str);
fclose(fptr);
}
```

(C) Reading and Writing <u>using formatting</u> I/O functions:-

(i) Syntax for writing to a file:- fprintf (fptr,"Format string",data_items);

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
main()
{
FILE *fptr;
char name[30];
char fname[30];
char choice;
if((fptr= fopen("Patel.txt","w"))==NULL)
{
 printf("\n\n It can not open Defined file by user");
//exit(1);
}
 while(1)
{
printf ("\n Enter name of Student=");
scanf("%s",&name) ;
printf("\n Enter Father name of Student=");
scanf("%s",&fname);
```

```
fprintf(fptr,"\n*****************************************");
fprintf(fptr,"\n\t%s\t\t%s",name,fname);
printf("\n Do you want to add any more records (press n for exit and y for yes)?");
choice=getche();
if(choice=='n' || choice=='N')
break;
}
 fclose(fptr);
}
```

(ii) Syntax for reading to a file:- fscanf(fptr,"Format string",data_items);

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
main()
{
FILE *fptr;
char name[30];
char fname[30];
if((fptr= fopen("Patel.txt","r"))==NULL)
{
printf("\n\n It can not open Defined file by user");
//exit(1);
}
while(fscanf(fptr,"%s%s",&name,&fname)!=EOF)
printf("\t%s\t%s\n",name,fname);
fclose(fptr);
}
```

Program for creating linked list:-

```
#include<stdio.h>
#include<conio.h>
#include<alloc.h>
struct node
{
int data;
struct node *next;
}*start=NULL;
void create()
{
char ch;
 do
 {
  struct node *new_node,*current;
  new_node=(struct node *)malloc(sizeof(struct node));
  printf("nEnter the data : ");
  scanf("%d",&new_node->data);
  new_node->next=NULL;
  if(start==NULL)
  {
  start=new_node;
  current=new_node;
  }
  else
  {
```

```
  current->next=new_node;
  current=new_node;
  }

 printf("\nDo you want to creat another  press y/n: ");
 ch=getche();
 }while(ch!='n');
}
void display()
{
struct node *new_node;
 printf("The Linked List : n");
 new_node=start;
 while(new_node!=NULL)
      {
   printf("%d--->",new_node->data);
   new_node=new_node->next;
      }
  printf("NULL");
}
void main()
{
create();
display();
}
```