LEARNING MATERIAL

ON

DATA STRUCTURES

(FOR 3RD SEMISTER CSE)

PREPARED BY: -

SWARNALATA SAHOO

GOVT POLYTECHNIC, DHENKANAL

CHAPTER-01
INTRODUCTION

Introduction: -

I. Data is Raw fact or an organized fact that need to be process.
II. Collection of an organized facts before processing refers to data.

Ex: -
$$\begin{tikzcd} 1 \arrow[d] & +2 \arrow[d] & =3 = \arrow[d] \arrow[r] & \text{information} \\ \text{Data} & \text{process} & \text{Data} & \end{tikzcd}$$

INFORMATION

When data is processed organized to make it useful it is one information.

Data type

- Datatype define the definition of data.
- The data type are mainly categorize into 2 types

I. Built-in data type or Primary data type
II. Derive data type or user defined data type

Built-in data type

I. This is also known as primary datatype.
II. This type of data type are pre-defined and as a fixed set of rule for declaration.
III. Example of such data type :-

1. Integer datatype
2. Float datatype
3. Character datatype
4. Boolean data type

Derived data type

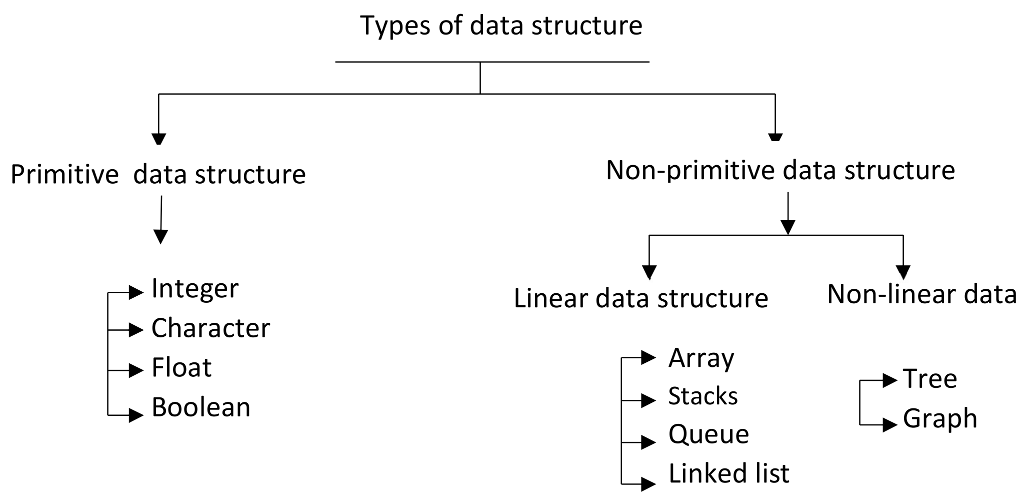
- This is also known as user define data type. This data type can be implemented independently within a language.
- Example of such data type :-

I. Array
II. String
III. Structure
IV. Union etc.

DATA STRUCTURE

1) Data structure is a specialize way for organizing and storing data in memory .
2) So, that one can perform operation on it.
3) Data structure is

- How to represent data.
- What relationship data element s have among themselves.
- How to access those data elements.



Linear data structure

- **i.** The arrangements of data elements are in a sequential format .So one is connected to one element .
- **ii.** In this data structure one element is connected to another element in linear form.

Non-linear data structure

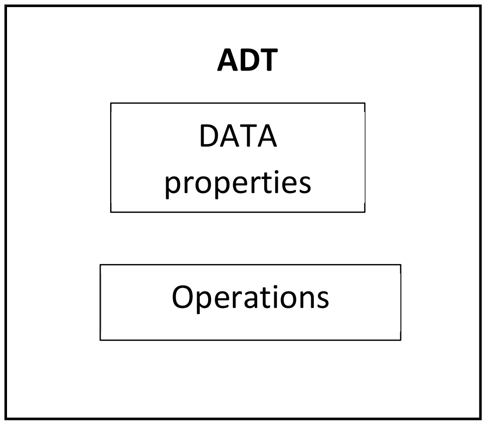
- i. In this data structure one element is connected to 'n' number of elements .

Data structure operations :-

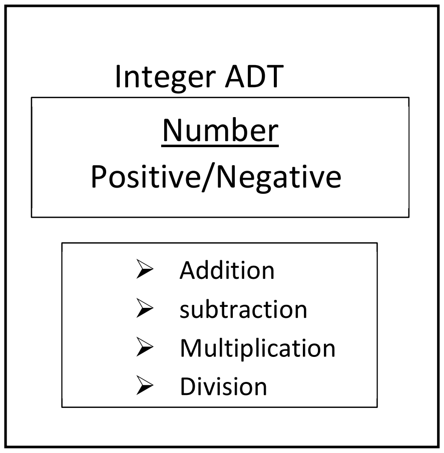
- ➢ The following 4 operations play a measure role in this text
  - (1) Traversing – Accessing each record exactly once so that certain items in the record may be processed (these accessing & processing is sometimes called visiting the records)
  - 2) Searching – finding the location of the record with a given key value.
  - 3) Inserting – Adding a new record to the structure.
  - 4) Deleting – Removing a record from the structure.
- ➢ The following 2 operations which are used in special situation will also be consider .
  - 1) Sorting- Arranging the records in some logical order it may be ascending or descending
  - 2) Merging- Combining the records in two different sorted files into a single sorted file

Abstract data type

- **i.** **A**bstract data type (ADT) is a mathematical model with collection of operations define on that model .
- **ii.** Model of a data type –
  - ➢ Properties of data
  - ➢ Operations that can be perform on that data.

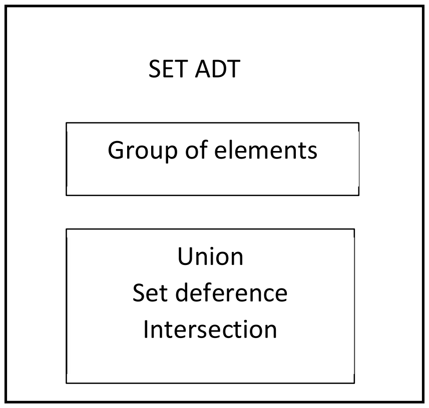


Example of ADT :- Integer



Example

SET UNION (set A , set B)



Algorithm & It's Complexity

Algorithm: -

- Algorithm is a step by step process to solve a problem of the computer programming language.
- Using an algorithm we can write any programming language such as :- C, C++, Java etc.
- The performance of an algorithm can be measure on the scales of time and space.
- If space complexity is more then time complexity will be more .If Space complexity is less then time complexity will be less.
- It mainly depends upon the space complexity as compare to time complexity

Time complexity

- Time complexity of an algorithm signifies the total time required by the program to run till its completion.
- The Time complexity of an algorithm is most commonly expressed using the "Big O Notation" if an asympototic notation to represent the time complexity .

Types of notation for time complexity

I. Big O – It denotes "fewer than or same as"
II. Big ω – It denotes "more than or same as"
III. Big θ--It denotes "same as"
IV. Little o – It denotes "fewer than"
V. Little ω - It denotes "More than"

Space complexity

- The amount of space required to execute a program.

Algorithm Analysis :-

Best case complexity

- The best case complexity of an algorithm is the function define by the minimum number of steps taken an any instance of size n.
- It represents the curb passing through the lowest point of each column.
- Big O (1)
- n =23

Average case complexity

- The average case complexity of an algorithm is the function define by the average number of steps taken an any instance of size n.
- Big O (n/2)
- n=36

Worst case complexity

- The worst case complexity of an algorithm is the function defined by the maximum number of steps taken an any instance of size n.
- Big O(n)
- n = 40

CHAPTER -2

STRING PROCESSING

Character	String
I. It is considering about 1 alphabet or 1 byte.	I. It is considering about more than 1 alphabet or 1 byte or more.
II. It is written in single quotation (‘ ’).	II. It is written in double quotation (“ “).
III. Character is consisting himself.	III. String is a collection of characters.
IV. In Array, character is defined with single character but at no null character will be obtained in the last end . $\overset{5000}{}$ **Ex:- ‘A’ =** $\boxed{A} \searrow$ Index or address.	IV. In Array, string is consisting of character and confined with null character at the end state. **Ex:-** “A” = $\overset{200}{\boxed{‘A’}}\ \overset{201}{\boxed{‘0’}}$
V. Every character can be a string.	V. Every string cannot be a character.

STRING: -

- A group of numbers can be stored in an array, group of characters can also be stored in an array. Such an array of character is called as string.

OR

- String is combination of character, numeric with special character.

Example:-

‘S’	**‘T’**	**‘R’**	**‘I’**	**‘N’**	**‘G’**	**‘\O’**
0	1	2	3	4	5	

a[2]=30

a[5]= $ array size

array name

- The number of characters in a string is called its length.

<u>Ex</u> ->6

- The string with zero character is called "empty string” or "null string ".

<u>Ex</u>:-

___	‘\0’

<u>CONCATENATION</u>: -

- Let S_1, S_2 be string, the string consisting of character of S_1 followed by character of S_2 is called Concatenation of S_1 and S_2 .
- It is denoted by S_1 // S_2 .

Ex :-

(i) “The”// ”end”

S_1 = The

S_2 = end

S_1 // S_2 = “The” // “end”= Theend.

(ii)”The”//”___ “//”End” = The End.

<u>Sub-string</u> :-

A string 'y' is called as sub-string of a string of ‘S' if they are exist ‘X' and 'Z'.

- If 'X' is an empty string then 'y' is called an initial sub-string of 'S' and 'Z' is an empty string then 'y' is called a terminal sub-string of ‘S’.

<u>Storing strings</u> :-

- Storing strings means how to store the string in different method like as:
  1. Fixed length structure.
  2. Variable length structures with fixed maximum.
  3. Linked storage structures.

1. Fixed length structure: -

- In fixed length storage each line print is viewed as a record where all records have the same length i.e; where each records accommodate the same number of character.

  That is; data/character=length size

Ex:- char a[6]

S	T	R	I	N	G

2. Variable length structures with fixed maximum: -

- The storage of variable length string is memory cells with fixed length that can be done 1 can used marker such as 2-dollar signs (‘$’ ‘$’) and two signal at the end of the strings.

Ex:-

data/character≠ length size
data/character<length size

Char a[7]

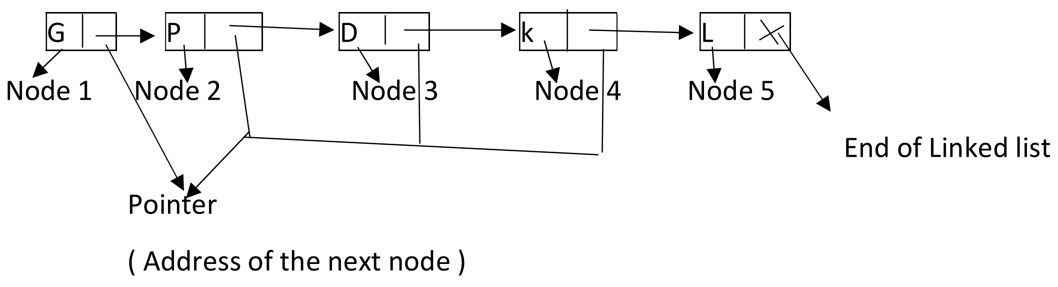
$R_{\$}$	e	s	h	m	$a_{\$}$	

3. Linked storage structure: -

- Computer are being used to frequently today forward processing i.e; for inputing processing and outputing printed matter.
- For more extensive word processing, application string are stored by means of linked list.

- **Linked list:-**Linked List is a linear collection of data elements called nodes in which the linear representation is given by links from one node to next node.

Ex:- “ GPDKL”



➢ <u>CHARACTER DATA TYPE</u> :-

- Character data types are of two types:-

1. Character (char)
2. Character Varying (Varchar).

Character	**Character varying**
➢ Character $\rightarrow$ data can be stored as fixed length.	➢ Character $\rightarrow$ data can be stored as variable length.
➢ Loss of memory is not obtaining by char.	➢ Loss of memory is obtaining by varchar.
➢ Character will take small size i.e; 0 to 255.	➢ Varchar will take size upto 0 to 65,535.
➢ In case of character it is not flexible.	➢ In case of varchar it is flexible .
➢ Ex :- Char a[5] | G | P | D | K | L |	➢ Ex :- Char a[6] | C | S | E | | | |

➢ <u>STRING OPERATION</u>: -

1. <u>Length</u> ()

<u>Syntax</u>: - LENTH (STRING)
<u>Ex</u>- "COMPUTER APPLICATION"
$\downarrow$
LENTH("COMPUTER APPLICATION")
= 20

2. <u>Sub String</u> ( )

<u>Syntax</u>:- SUBSTRING(STRING,INITIAL,LENGTH)
<u>Ex</u>:- SUBSTRING("COMPUTER APPLICATION",0,8)
=COMPUTER

3. <u>Indexing</u> ( )

<u>Syntax</u>:- INDEX (STRING,PATTERN)
<u>Ex</u>:-INDEX("HE IS WEARING GLASSES",EAR)
= 8

4. <u>Concatenation</u> ( )

<u>Syntax</u>:-STRING 1 // STRING 2.
<u>Ex</u>:-"COMPUTER "//"APPLICATION"
= COMPUTERAPPLICATION.

5. Word Processing ( )

Insert Delete

a) Insert ( )

Syntax:-INSERT (STRING , POSITION , STRING).
Ex:-INSERT(“ABCDEIJKL”,5,FGH)
= ABCDEFGHIJKL

b) Delete ( )

Syntax:- DELETE(STRING,POSITION,LENGTH)
Ex-DELETE(“ABCDEFGH”,4,2)
=ABCDG

6. Replacement ( )

Syntax: -REPLACE (STRING, PATTERN1, PATTERN2)

Ex:-REPLACE(“ABCDIJKL”,D,N)
= ABCNIJKL

CHAPTER-3

ARRAY

Introduction about Array: -

Array is a collection of homogeneous data element of same datatype.

Syntax: Datatype array name [size]

ex: int a[5]

10	20	30	40	50

LINEAR ARRAY :-

- A linear array is a finite number 'n' of homogeneous data element. That is data elements of the same type such that-
    - I. The elements of the array are referenced respectively by an index set consisting of 'n' consecutive numbers.
    - II. The elements of the array are stored respectively in successive memory locations.
    - III. 'n' is the number of elements is called the length or size of the array.
    - IV. Index 'a' consists of the integer 0,1,2,3...n
    - V. Length = UB -LB +1

where,

UB= Upper Bound

LB= Lower Bound

REPRESENTATION TION OF LINEAR ARRAY IN MEMORY: -

- Memory of a computer is simply a sequence of Memory address location.
- Elements of a linear array as stored in Successive memory locations.
- The computer does not need keep track of the address of every element of array but need to keep track of the only the address of the first element called base address .
- In most programming languages the name of the array is associate with the starting address of the memory location.
- LOC (LA [K] Base (LA) + W(K-LB)

Where, K is index

LA = Linear Array.

W= size of Each Element.
LB = Lower bound.

(1) Int marks[]={99,67,78,56,88,90,34,85}
Calculate the address of marks [4] of the base address 1000 and W = 2 ?

LA[4]	=1000(+2(4-0)) =1000+8 =1008
LA[1]	=1000+2(1-0) =1002
LA[2]	=1000+2(2-0) =10004
LA[3]	=1000+2(3-0) =1006
LA[4]	=100+2(5-0) =1010
LA[6]	=1000+2(6-0) =1012
LA[7]	=1000+2(7-0) 1014

<u>Traversing linear array</u>: -

- Traversing an array means accessing each and every element of the array for a specific purpose.
- Traversing data element of an array can include Printing Every element, Counting the total number of elements or performing any process of those element.
- Since array is a linear data structure traversing it's element very simple & strict forward.
- It is the process visiting Each element of the array exactly once that is starting from 1st element up to the last element.
- Let,
  LA be a linear array stored in the memory of the computer we want to print each element of linear array.

<u>ALGORITHM</u>:-

step 1 - Traversing a LA (LB. .....UB)

Step 2 -(Initialize Counter) set k=LB

step 3 - Repeat steps 4&5 while (K<=UB)

step 4 - Visit LA [K]

Step 5 - set K=K+1

Step 6 - Exit

- **Write a programme to traverse an array:-**

```
#include<stdio.h>
#include<conio.h>
void main()
{
int in ,n ,a [10];
Clrscr ();
printf ("\n enter the length of array");
scanf("%d", &n);
printf ("Enter the elements");
For (i=0; i<=n; i++)
{
scanf("%d\n", &a[c]);
}
printf (" traversing of the array");
for (i=0; i<=n-1; i++)
{
printf ("%d\n", a[i]);
}
getch ();
}
```

<u>Inserting</u>:-

To insert the new element in the existing array following different position of the array used.

- At the end the array.
- the beginning of the array.
- At given position.
- in the shorted array.

At the end of the array:-

Algorithm :-

```
step - 1 – If  UB = max then array is overflow and stop.
step 2 =Read data
Step 3 =UB ← UB +1
Step 4 =A [UB] ← data
Step-5=stop
```

Program :-

```
#include <stdio.h>
#include <conio.h>
void main()
{
int i ,n ,a [10];
Clrscr();
printf ("\n enter the size of the array :");
scanf("%d", &n);
printf ("enter the elements:");
For (i=0; i<=n; i++)
{
scanf("%d", &a[i]);
}
for (i=0; i<=n-1;i++)
{
Printf ("over flow");
}
 getch();
}
```

OUTPUT:-

```
enter the size of the array:4
enter the elements:1,2,3,4,5
```

over flow over flow over flow over flow

At the beginning 0f the array:-

Algorithm:-

```
Step- 1 -if UB = Max -then Write array is overflow & Stop
Step-2 - READ =DATA
 Step-3- K ←UB
Step-4- Repeat step
Step-5 -A[K+1]←A[K][5]while k ≥ LB  K←K-1
 Step-6 -A[B]← DATA
 Step -7- STOP
```

Program :-

```
#include <stdio.h>
#include <conio.h>
void main()
{
int a [10],i, n;
Clrscr();
printf (" Enter the size of the array");
scanf("%d", &n);
If(n>=10)
{
printf (" over flow");
}
printf("Enter the element of array");
For (i=0; i<=n; i++)
scanf("%d", &a[i]);
for (i=n;i>0;i--)
a[i]=a[i-1];
printf (" Enter the new element");
scanf("%d",& a[0]);
printf ("after insert");

n++;

for (i=0;i<n;i++)

printf ("%d\t", a[i]);

getch();

}
```

OUTPUT:-

```
Enter the size of the array:3

Enter the elements:1,3,5

Enter the new element 6

Enter insert 6,1,3,5
```

At the given position :-

We want to insert an element in the middle of the array then half of the elements must be moved downward to new locations to accomorded the new element and help the order of the elements.

Algorithm:-

```
INSERT ( LA, N, K, ITEM )
( LA is the linear array with N elements and K ≤ N insert item LA[ K ] )
Step-1 :- j = N
Srep- 2:- Repeat step 3 & 4 while k <= j
Step- 3:- Move the elements downwards LA [ j + 1 ] = LA [ I ]
Step- 4:- j = j-1 ( end of step 2 loop )
Step- 5:- Set L[ K ] = item
Step- 6:- Set N = N+1
Step- 7:- Exit
```

Program: -

Sorted Array :-

```
Step- 1:- If UB= Max then write ‘ array ‘ is overflow and stop
Step- 2:- Read data as elements to be in sorted
Step- 3:- k ← LB
Step- 4:- Repeat step (5) while A(K) < data
Step- 5:- K ← K+1
Step- 6:- LOC ← K
          K ← UB
Step- 7:- Repeat step (8) while K ≥ LOC
Step- 8:- A ( K+1) ← A(K)
```

```
        K ← K – 1
Step- 9:- A (LOC) ← data
Step- 10:- STOP
```

<u>Deletion</u> :-

→ Link insertion deletion is also possible at any position in the array such as from the end ,from the beginning if specific elements is given

→ Before deleting the element first we have to check the under flow condition that is if N = 0 (N = UB-LB + 1), Array is underflow and stop

Algorithm:-

From the end

```
1. If N = 0 then write 'UNDERFLOW' and STOP
2. A(UB) ← NULL
3. UB ← UB-1
4. STOP
```

From the beginning

```
1. If N = 0 then write 'UNDERFLOW' and STOP
2. K ← LB
3. Repeat step(4) while K < UB
4. A(K) ← A( K+1 )
     K ← K+1
5. A(UB) ← NULL
   UB ← UB – 1
6. STOP
```

<u>Deletion of given element's</u>:-

<u>Algorithm</u>:-

```
1) If N=0 then array is UNDERFLOW & STOP
2) READ DATA as element to be deleted.
3) READ LOC as location at where deletion will be made.
4) K←LB
5) Repeat step (6) while A (K)=DATA
6) K←K+1
7) Repeat step (8) while K<UB
8) A(K)←A(K+1)
   K←K+1
9) A(UB)←NULL

   UB←UB-1

10) STOP
```

Multi dimensional array: -

(2d array)

A [2] [3]={60, 40, 30, 50, 60, 25}

= {60,40,30,50,60,25}

- Most programming languages allow 2D and 3D array that is array elements are referenced by 2 and 3 subscriptes.

2-Dimensional array:-

- 1 dimensional array are organized linearly in only one direction but at times we need to store data in the form of brief and tables.
- Here the Concept of single dimensional array is extended to incorrupted 2D data structure.
- A 2D array the specified using two subscripted denotes the row subscripted and Column subscriptes.

Declaration :-

Datatype array name [raw size] [column size];

int marks [2] [3]

- M×N array Contains M×N data element and each element is accessed using 2 subscripted ‘I’ and ‘j' where i < =m, i<=n.
- 2d array are called matrix in math and tables in business application the 2d array are called matrix array.

  Ex:-

  A 2d array 2x3 can be represented as

$$\begin{bmatrix} 3 & 2 & 1 \\ 6 & 3 & 5 \end{bmatrix}_{2\times 3}$$

a[1,1],a[1,2],a[1,3]

a[2,1],a[2,2],a[2,3]

Representation of 2D array in memory :-

Let A be a 2D m×n array ,the array will be represented in memory by a block of m×n sequential memory location array can be represented in memory in two ways.

I. Row measure order.
II. Column measure order.

Row measure order:-

The array is

array[1. . . . .m,1. . . . . n]

$$\begin{bmatrix} 1 - - - -n \\ 1 - - - -n \\ 1 - - - -n \\ . \\ . \\ . \\ 1 - - - -n \end{bmatrix}$$

The first way is the row measure order & the second way is the column measure order.

(0,0) (0,1) (0,2). . . . . . . (0,n-1)

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

(Row measure order)

(0,0)(0,1)(0,2). . . . . (0,m-1)

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

(Column measure order)

Memory store:-

Address [A(I , j)]=Basic address+w{n(i-1)+(j-1)}

When 'n'=number of column

W= number of

I & j are subscripts of the array element

<u>Ex</u>:-

Consider a 20×5 2d array marks which has its base address 1000 and the size of an element 2 . now compute the address of element make [18][4] assume that the element are stored in row measure order.

=1000+2{5(18-1)+(4-1)}

=1000+2{88}

=1000+176

=1176

Column measure order:-

If the array element are stored in column measure order.

Address [A(I ,j)]=Basic address +w{m(j-1)+(i-1)}

=1000+2{20(4-1)+(18-1)}

=1000+2{60+17}

=1000+154

=1154

Sparse matrix:-

The sparse matrix is a matrix that has large number of element with a '0' values as the dominating elements.

Ex:- $\begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 2 & 0 & 0 \end{bmatrix}$

Representation of matrix in memory :-

- The matrix consume a lot of memory then the matrix is sparse, the memory allotted to the matrix is wasted.
- In such a case to save valuable storage space we restore to triplet representation (i , j, value) to represented each non zero element of the sparse matrix.

Ex- $\begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 2 & 0 & 0 \end{bmatrix}$

i	j	Value
1	1	2
3	3	1
4	2	2

i = raw
j = column
Value = non zero elements

➢ Write a programme to print the element of 2D array.

```
#include <stdio.h>

#include <conio.h>
```

```
void main( )
{
Int a [2] [2]={(2,4), (3,19)}
For (i=0; i<2; i++)
{
printf ("\n");
}
For (j=0; j<2; j++)

printf ("%d\t", a[i] [j]);
}
getch( );
}
```

$$\begin{bmatrix} 2 & 4 \\ 3 & 19 \end{bmatrix}$$

➢ Write a programme to print the element of 2D array.

```
# include<stdio.h>
# include<conio.h>
void main( )
{
int  I , j .a[2],[3];
clrscr();
for(i=0;i<2; i++)
{
for( j=0;j<3;j++)
{
printf("Enter value for disp[%d][%d]=",i ,j);
scanf("%d", &a[i][j]);
}
}
printf("2D  array elements:\n");
for( i=0;i<2;i++)
{
for(j=0;j<3;j++)
{
printf( "\t%d",a[i][j]);
if(j==2)
{
printf("\n");
}
}
}
getch();
}
```

OUTPUT:-

```
Enter the value for disp[0][0]=2
Enter the value for disp[0][1]=52
Enter the value for disp[0][2]=12
Enter the value for disp[1][0]=41
Enter the value for disp[1][1]=23
Enter the value for disp[1][2]=11
Two dimensional array elements are:
2  52  12
41 23 11
```

CHAPTER-4

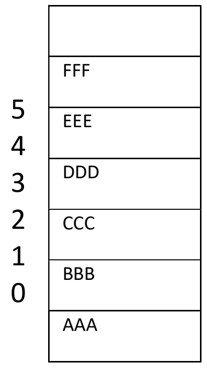
STACKS AND QUEUES

Stack:-

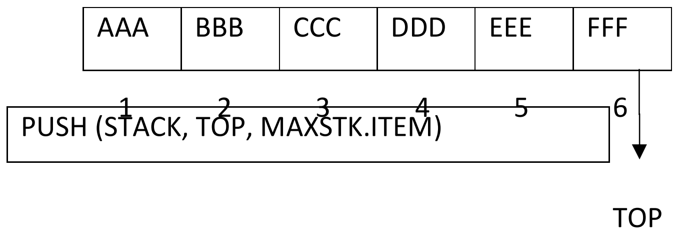
- A stack is a linear structures in which items may be added or removed only at one end.
- Example of such a structure – A stack of folded towels etc.
- That an item may be added on removed only from the top of any of the stacks. This means particular that the last item to be added to a stack is the first item to be removed. Accordingly stacks are also called last-in-first-out (LIFO).
- A stack is a list of elements in which an element may be inserted or deleted only at one end, called the top of the stack. This means in particular that elements are removed from a stack in the reverse order stack that in which they were inserted into the stack.
- Special terminology is used for two basic operations associated with stacks-
  (a)"push"- Push is the term used to insert an element into a stack.
  (b) "Pop"- Pop is the term used to delete an element into a stack.

  Ex-Suppose the following 6 dements are pushed in-order to an empty stack.
  STACK: AAA, BBB, CCC, DDD, EEE, FFF.

LIFO



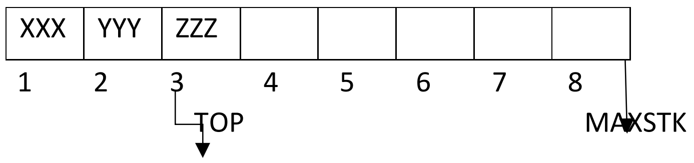
- That Insertions and deletions can occur only at the top of the stack.
- This means EEE cannot be deleted before FFF, DDD cannot be deleted before EEE and FFF-are deleted and so on.



(Diagrams of the stacks)

Array Representation of Stacks :-

- Stacks may be represented in the computer in various ways, usually by means of a one way list or a linear array ·Unless otherwise started or implied each of our stacks will be maintain by a linear array stack a pointer variable top, which contains the location of the top element of the stack and a variable MAXSTK. which gives the maximum number of elements that can be held by the stack.
- The condition TOP=0 or TOP= NULL will indicate that the stack is empty.
- Such an array representation of a stack, Since TOP=3, the stack has three elements, XXX.YYY.ZZZ and Since MAXSTK=8, there is room for five more stems in the stack.



- The operation of adding (pushing) an item in to a stack and the operation of removing (poping) an item of a stack may be implemented respectively by the following procedures called Push and Pop.
- In executing the procedure. Push one must first test whether there is room in the stack for the new item, if not then we have the condition known as Overflow condition.
- In executing the procedures pop one must first test whether there is an element in the stack to be deleted, if not then we have the condition known as Under flow condition.

Algorithm:-

Push Operations:-

- This procedure pushes an ITEM into stack.

```
Step 1- If TOP = MAXSTK then print Overflow and return.
Step 2- Set TOP=TOP+1
Step 3- Set STACK [TOP]=ITEM
Step 4-Return
```

- ## Pop Operation:-
    - This procedure deletes the top element of stack and assigns to the variable item.

      Step 1- If TOP=0 then print Underflow and return .

      Step 2- Set ITEM = STACK [TOP]

      Step 3 - Set TOP = TOP-1

      Step 4 – Return

- ## Arithmetic Expression:-
    - Let Q be an Arithmetic Expression involving constants and operations.
    - This section gives an algorithm which finds the value of Q by using reverse polish (Post fix) notation.
    - The binary operations in Q may have different levels of precedence. Specifically we assume the following three levels of precedence for the usual five binary operations.

	Precedence	Operators:-
1)	Highest:	Exponentiation (↑)
2)	Next highest:	Multiplication(*) and division (/)
3)	Lowest :	Addition (+) and Subtraction (-)

EX:-

Evaluate the following parentheses tree arithmetic expression.

2↑3+5*2↑2-12/6

=8+5*4-12/6

=8+20-2

=28-2

=26

Polish Notation:-

- For most common arithmatic operations, the operator symbol is placed between its two operands.
- This is called Infix Notation.
- For example A + B , C – D , E*F , G/H
- By using either parenthesis or some operator precedence convention such as,

  Ex:-

  (A + B) *C , A+(B*C)
- Accordingly, the order of the operators and operand in an arithmetic expression does not uniquely determine the order in which the operations are to be performed.
- Polish notation in which the operator symbol is placed before its two operands.

Ex:-

+ AB, - AB ,*CD , /GH

- We translate, step-by-step, the following infix expression Into polish notation using brackets [ ] to indicate a partial translation.

  (A+B)*C =[+AB]*C=*+ABC
  A+(B*C)=A+[*BC]=+A*BC
  (A+B)/(C-D)=[+AB]/[-CD]=/+AB-CD
- Reverse polish notation refers to the analogous notation in which the operator symbol is placed after its two operands.

Ex:-

AB+, CD, EF*, GH /

- One never needs parenthesis to determine the order of the operations in any arithmetic expression written in reverse polish notation.
- This notation is frequently called postfix (or suffix) notation.
- Whereas prefix notation is the term used for polish notation.

• <u>Evaluation of a postfix Expression</u>:-

- This algorithm finds the value of an arithmetic Expression P written in postfix notation.

<u>**Step 1**</u>**-** If Add a right parenthesis ")" at the end of P.

<u>**Step-2**</u>- Scan P from left to right and repeat steps 3&4 for each element of P untill the Sentinel ")" is encountered.

<u>**Step-3**</u> -If an operand is encountered, put it on STACK.

<u>**Step-4 -**</u> If an operator Ø is encountered, then

a) Remove the two top elements of STACK, when A is the top element and B is the next to top element.
b) Evaluate B ØA
c) place the result of (b) then back on STACK.

<u>**Step-5 -**</u> Set value equal to the top element on STACK.

<u>**Step-6**</u> – Exit

Ex:-
Consider the following arithmetic expression P written in postfix notation.
P: 5,6,2,+,*,12,4,/,-
=5,6,2,+,*,12,4,/,-)
=37

Symbol scanned	STACK
5	5
6	5 6
2	5 6 2
+	5 8
*	40
12	40 12
4	40 12 4
/	40 3
-	37
)	

- **Trans forming Infix Expression into Postfix:-**

Expression:-

Algorithm:-

Suppose Q is an arithmetic expression written in infix notation this algorithm finds the equivalent postfix expression P.

Step 1-Push left parentheses "(" on to STACK and add right parenthesis ")" to the end of Q.

Step 2- Scan Q from left to right and repeat steps 3 to 6. Each element of Q until the STACK is empty.

Step 3- If an operand is encountered, add it to P.

Step 4- If a left parentheses "(" is encountered, push it on to STACK.

Step 5-If an operator Ø is encountered then,

a) repeatedly pop from STACK and add it to P each operator which has the same precedence or higher precedence than Ø
b) Add Ø to STACK.

Step 6- If a right parenthesis ")" is encountered
a) repeatedly pep from STACK and add it top each operator until a left parenthesis is encountered.
b) Remove the left parenthesis "(".

Step 7-Exit

Ex:-

Consider the following arithmetic infix notation (Infix toto postfix)

Q: A+(B*C-(D/E↑F)*G)*H

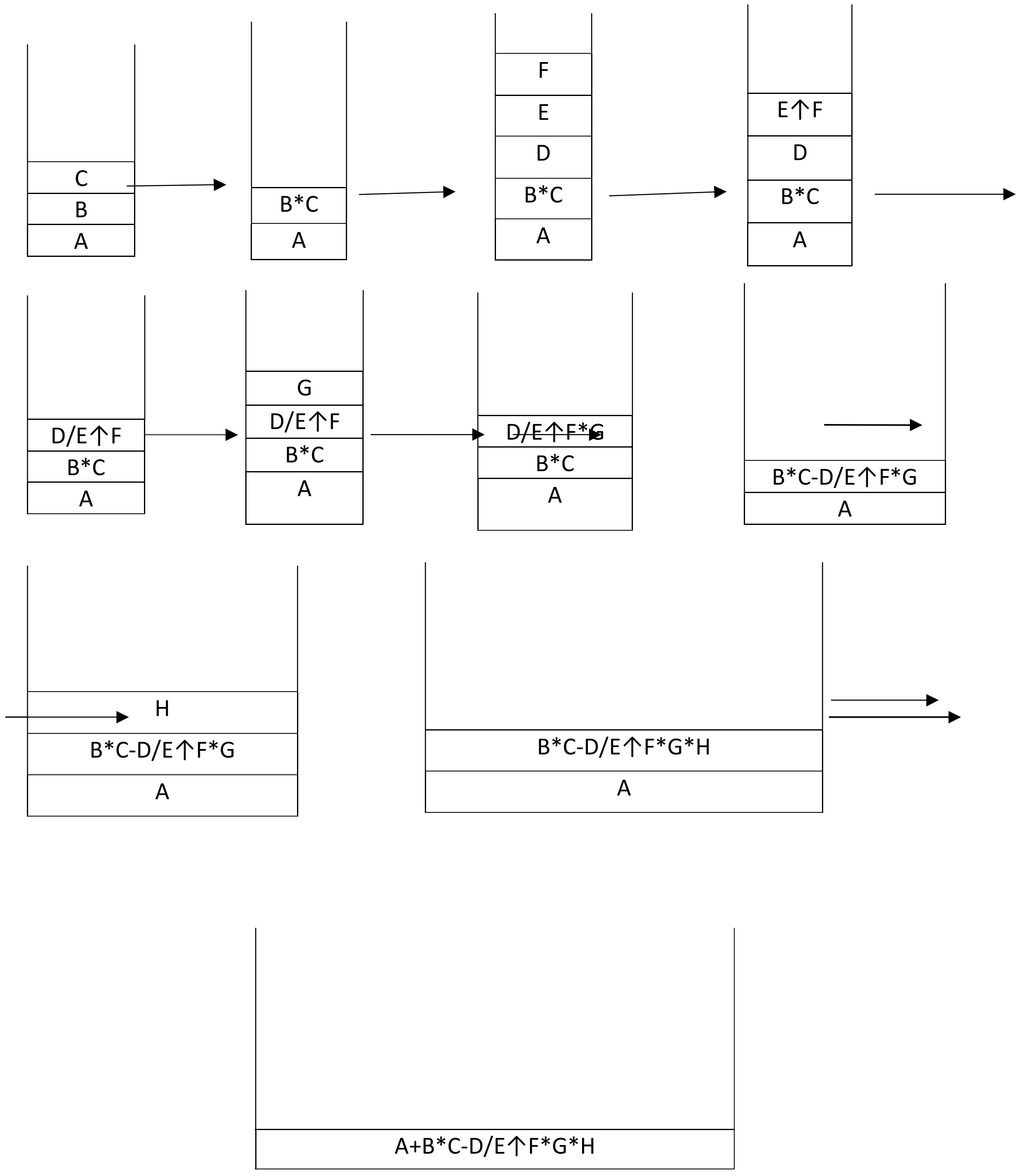
Q: A+(B*C-(D/E↑F)*G)*H)

Q	STACK	P
	(	
A	(	A
+	(+	A
(	(+(	A
B	(+(	AB
*	(+(*	AB
C	(+(*	ABC
-	(+(-	ABC*
(	(+(-(	ABC*
D	(+(-(	ABC*D
/	(+(-(/	ABC*D
E	(+(-(/	ABC*DE
↑	(+(-(/↑	ABC*DE
F	(+(-(/↑	ABC*DEF
)	(+(-	ABC*DEF↑/
*	(+(-*	ABC*DEF↑/
G	(+(-*	ABC*DEF↑/G
)	(+	ABC*DEF↑/G*-
*	(+*	ABC*DEF↑/G*-
H	(+*	ABC*DEF↑/G*-H
)		ABC*DEF↑/G*-H*+

- How to check this question is correct or not→

This is the conversion of postfix notation to infix notation

ABC*DEF↑/G*-H*+



Application of Stack:-

1)Expression Evaluation:-

- Stack is used to evaluate prefix, postfix and infix expressions.

2)Expression Conversion:-

- Stack is used to convert infix to postfix, infix to prefix form

3)Parentheses Checking:-

- Stack is used to check the proper opening and closing of parenthesis.

4)Page Visited History in a Web Browser

5) Undo sequence in a text editor.

6) Evaluation of arithmetic expression.

7) Recursion:-

- A function which calls itself is called Recursion.

I. Factorial function:-

- The product of the positive integers from one to end inclusive is called n factorial and is usually denoted by n!

  n! = 1, 2, 3, ...(n-2) (n-1`)^n
  0! =1
  1! =1
  4! =4×3×2×1
  4! =4×3!
  =4×3×2!
  =4×3×2×1!
  =4×3×2×1
  =24

  a) If n=0 , then n!=1
  b) If n>0 , then n!=n.(n-1)!

  Ex:-
  2!=2×(2-1)!
  =2×1!

Algorithm:-

The following are two procedures that each calculate n factorial.

1. Factorial (FACT,n)

```
Step 1- if N=0, then set FACT=1 and return.
Step 2-Set FACT=1
Step 3- Repeat for K = 1 to N
            Set FACT=K* FACT
Step 4- Return
```

- Factorial (FACT ,n)

```
Step-1- if N=0, then set  Fact=1 and return
Step-2 -Call Factorial (Fact, N-1)
Step-3- Set Fact = N* Fact
Step-4 -Return
N=5             5!
Factorial  (Fact, 4)
Factorial  (Fact,3)
Factorial  (fact,2)
Factorial  (fact, 1)

Factorial n*fact
Factorial =5*24
          =120
```

5!=5×4×3×2×1=120

II. Fibonacci Sequence:-

- It is usually denoted by (F_0, F_1, F_2 . . . . . . F_n) is as follows:- 0,1,1,2,3,5,8,13, 21, 34,55..

That is, Fo = 0 and F1 =1 and each succeeding term is the sum of two preceding terms.

- For example:-

The next two terms of the sequence are 34+55=89 and 55+89 = 144

$F_2=F_0+F_1$	$F_3=F_1+F_2$	$F_4=F_2+F_3$
$=0+1=1$	$=1+1=2$	$=1+2=3$

Def" of Fibonacci Sequence:-

a) if n=o or n=1, then F_n=n

b) if n>1 ,then, $F_n=f_n-2 + F_n-1$

Algorithm:-

$\boxed{\text{FIBONACCI (FIB, N)}}$

Step-1 if N=0 or N=1, then set FIB=N and return.

Step-2 Call FIBONACCI (FIBA, N-2)

Step-3 Call FIBONACCI (FIBB, N-1)

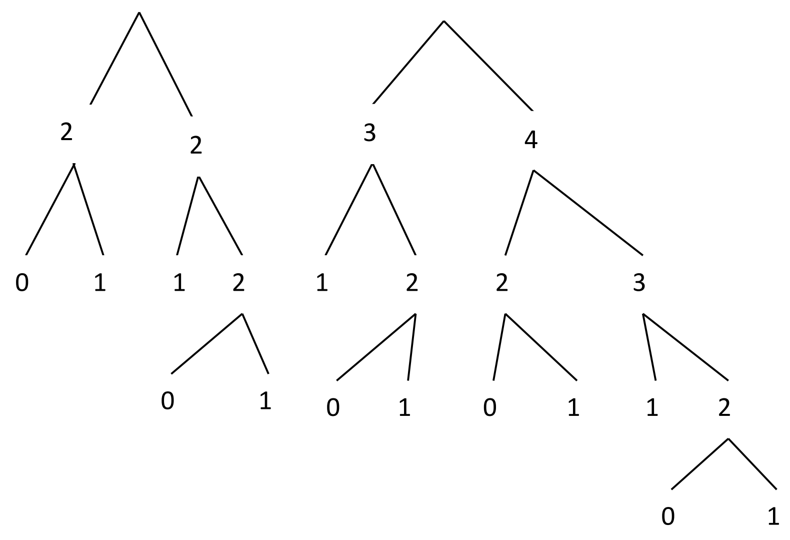
Step-4 Set FIB=FIBA+FIBB

Step-5 Return

Ex:-

FIB6

F6 =FIB4 + FIB5



FIB6=FIBA+FIBB

=3+5

=8

- **<u>Queues</u>:-**
  - A queue is a linear list in which items may be added only at one end and items may be removed only at the other end.
  - A queue is a linear list of elements in which deletions can take place only at one end called the front and insertions can the Rear.
  - The terms front and Rear are used in describe in a Linear list only when it is implemented as a queue.
  - Queues are also called first-in-first-out (FIFO) lists, since the first element in a queue will be the first element out of the queue. In other words the order in which elements enter a queue is the order in which they live.
    Ex:- (i)The People waiting in line at a bank from a queue where the first person in line is the person to be waited on and so on.
- **<u>The diagram of a queue with 4 elements</u>:-**

(A)

A.	B.	C.	D.

(B)

B	C	D

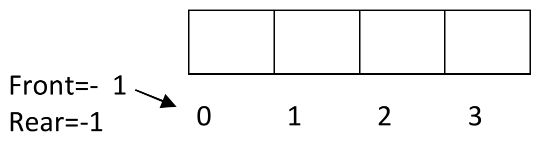
(C)

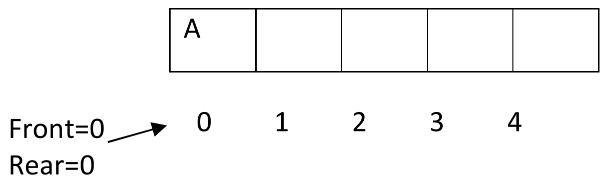
B	C	D	E	F

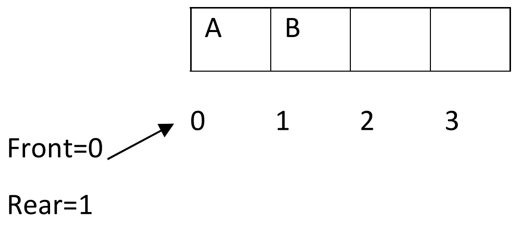
(D)

C	D	E	F

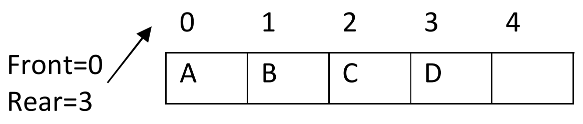
- The diagram of a queue with 4 elements where (A) is the front element and (D) is the rear element. The front and rear elements of the queue are also respectively the first and the last elements of the list. Suppose an element is deleted from the queue. Then it must be A.
- Figure B is now the front element.
- Next E is added to the queue and then F is added to the queue.
- Then they must be added at the rear of the queue as in figure C. F is now the rear element.
- Now suppose another element is deleted from the queue, then it must be B.
- In a date structure E will be deleted before F because it has been placed in the queue before F. However, E will have to wait until C and D are deleted.







Rear=Rear+1
=0+1
=1



Array Representation of Queues:-

- Queues may be represented in the computer in various ways ,usually by means at way list or linear arrays.
- Each of our queues will be maintained by a linear array queue and two pointer variables.
- Front containing the location of the front element of the queue and rear containing location of the rear element of the queue.
- The condition front = NULL will indicate that the queue is empty.
- The array will be started in memory using an array queue with N elements. It also indicates the way elements will be deleted from the queue and the way new elements will be added to the queue.
- Whenever an element is deleted from the queue the value of front is increased by 1 . This can be implemented by the assignment .
  FRONT = FRONT+1.
- Whenever an element is added to the queue the value of rear increased by 1. This can be implemented by the assignment.
  REAR =REAR +1

Ex(i)

A	B	C	D					
0	1	2	3	4	5	6	7	N-1

$\frac{F}{0}$ $\frac{R}{3}$

Max Q =N
Range of index=0-N-1

(ii) $\frac{F}{1}\frac{R}{3}$

	B	C	D						
0	1	2	3	4	5	6	7 . . .		. .N-1

(iii) $\frac{F}{0}\frac{R}{3}$

	B	C	D					
0	1	2	3	4	5	6	7 ..	. . N-1

(Array Representation of Queue)

- **Operation on a Queue:-**

1.Algorithm to insert an element in a Queue:-

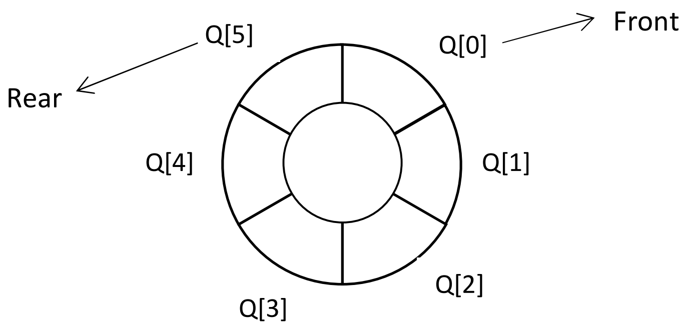
```
Queue Insert (Queue, N, FRONT, REAR, ITEM)

Step 1- If REAR = N-1
        Write Overflow
        Go to step 4
         [End of If]
Step2- If FRONT=-1 and REAR=-1
```

```
Set FRONT = REAR=0
ELSE
SET REAR = REAR+ 1
 [END OF IF]
Step 3 - Set QUEUE [REAR = ITEM]
Step 4- EXIT
```

2.Algorithm to delete an element from a queue:-

```
Step 1 - If FRONT=-1
Write UNDERFLOW
ELSE
SET VAL = QUEVE [FRONT]
 SET FRONT= FRONT +1
[END OF IF]
Step 2-Exit
```

- **Circular Queue:-**
  - As the name indicates a circular queue is not linear in structure but circular.



  - In a circular queue the elements Q[0], Q [1]. . . . . N-1 is represent in a circular.
  - A circular here is one in which the insertion of a new element is done at the very fast location of the queue.
  - After Inserting an element at last location queue Q[5] the next element will be inserted are very first location that is Q[o]. Circular queue very first is one in which the first element comes after the last element.

• INSERT AN ELEMENT IN A CIRCULAR QUEUE:-

```
Step-1 - IF FRONT=0 and REAR=N-1
        Write OVERFLOW
      Goto Step-4
           [ END OF IF]
Step 2 -IF FRONT =-1 and REAR=-1
            SET FRONT= REAR=0
ELSE LF
        REAR=N-1 and FRONT ≠0
            SET REAR=0
ELSE
     SET REAR=REAR+1
         [END OF IF]
Step 3- SET QUEUE [REAR] = ITEM
Step 4- EXIT
```

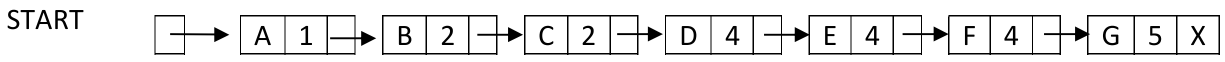
4

• DELETE AN ELEMENT IN A CIRCULAR QUEUE :-

```
Step-1 IF FRONT=-1
          Write UNDERFLOW
            Goto Step-4
Step- 2 -SET VAL=QUEUE [FRONT]
Step-3 -IF FRONT=REAR
ELSE IF
          FRONT=N-1
        SET FRONT=0
ELSE
       SET FRONT=FRONT+1
        [END OF IF]
Step -4  - EXIT
```

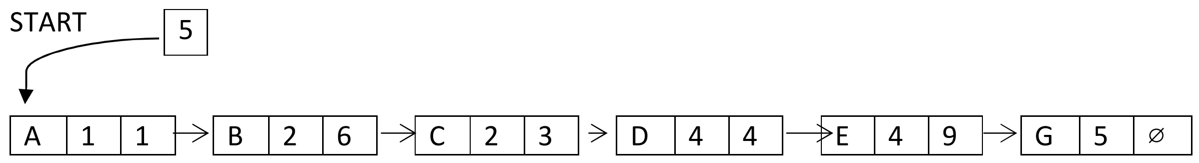
- **<u>Priority Queues</u>:-**
  - ➢ A priority queue is a collection of elements such that each element has been assigned a priority and such that the order in which elements are deleted and processed comes from following rules.
    - i. An element of higher priority is processed before any element of lower priority .
    - ii. Two elements with the same priority are processed according to the order in which they were added to the queue.

- **<u>One -way. List Representation of a Priority Queue</u>:-**
  - ➢ Each node the list will contain three items of information: an information field (INFO), a priority a number (PRN)and a link number.
  - ➢ A node X precedes a node y in the list-
  - ➢ When X has higher priority than Y. or
  - ➢ When both have the same priority but X was added to the list before y. This means that the order in the one way list corresponds to the order of the priority queue.

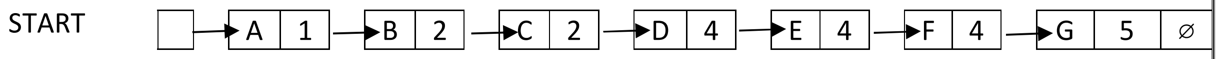


Ex:

	INFO	PRN	LINK
1	B	2	6
2			7
3	D	4	4
4	E	4	9
5	A	1	1
6	C	2	3
7			10
8	G	5	0
9	F	4	8
10			11
11			12
12			0



- **<u>Array Representation of a Priority Queue</u>→**



FRONT
3
1
0
5
4

REAR
3
2
0
1
4

$$\begin{array}{c|cccccc|}
 & 1 & 2 & 3 & 4 & 5 & 6 \\
1 & & & A & & & \\
2 & B & C & & & & \\
3 & & & & & & \\
4 & F & & & & D & E \\
5 & & & & G & &
\end{array}$$

CHAPTER-5

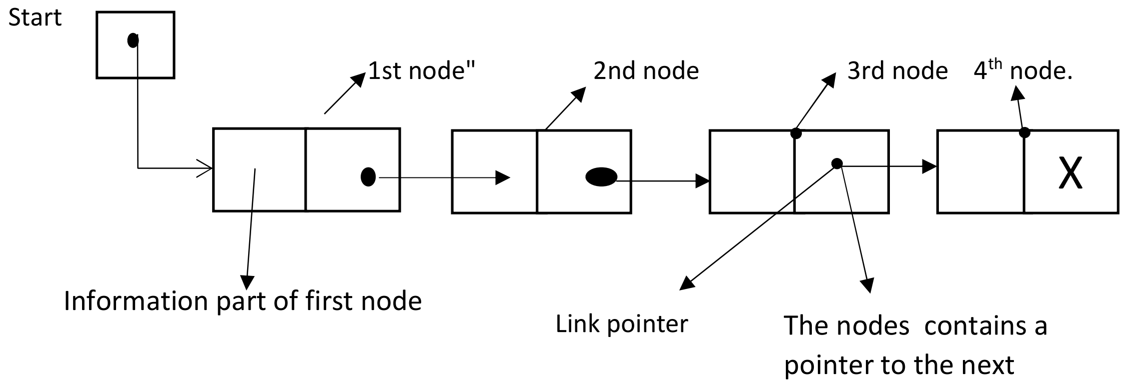
LINKED LIST

*Introduction:-

- Linked List is a linear collection of data elements called Nodes in which the linear representation is given by links from one node to next node.
- It is a data structure which in turn can be used to implement other data structure.
- It acts as a building block to implement data structure such as stack, and other queue variation.
- It can provide as a train or sequence of nodes in which each contains one or more data fields and a pointer to the next node.
- A Linked List in which every nodes contains two part –

I. Integer / Information part

II. Pointer to the next node


Start
1st node"
2nd node
3rd node
4th node.
X
Information part of first node
Link pointer
The nodes contains a pointer to the next

- A Null pointer is represented by {x} while programming we usually define Null.
- Hence a null pointer denotes the end of the list.
- Since in a linked list every nodes contains a pointer to another nodes which is of the same type, it is also called a self referencial data type.

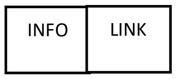
(chain representation of pointer)

- Linked List contains a pointer variable start which contains the address of the first node in the list.
- We can traverse the entire list using start which contains the address of the first node. The next part of the first node in the term stores the address of its succeeding node. Using this technique the individual nodes of the list with from a chain of nodes.

• <u>Representation of Linked List in memory</u> :-

- Let LIST be a Linked List. Then LIST will be maintain in memory unless otherwise specified or implied as follows-

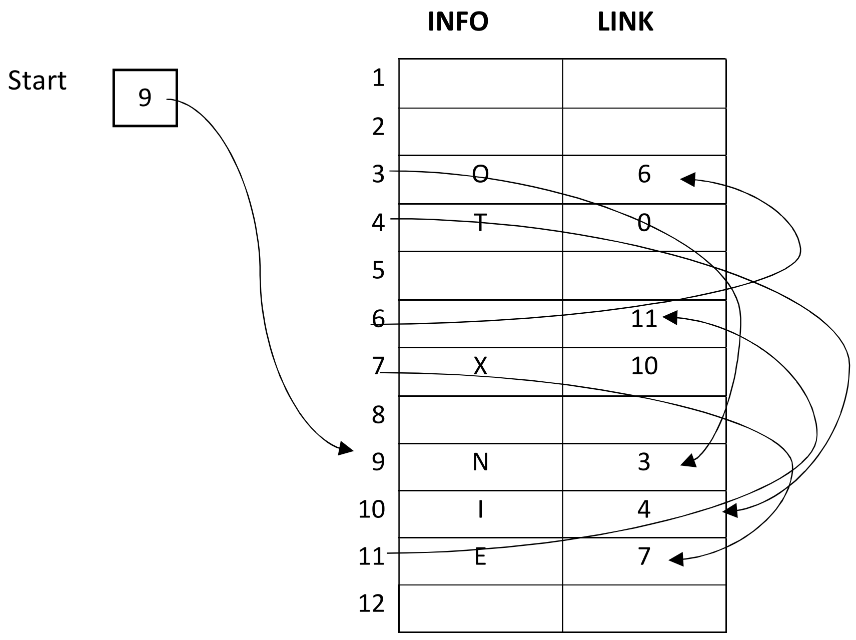
I. First of all LIST required two linear array which is INFO and LINK such that



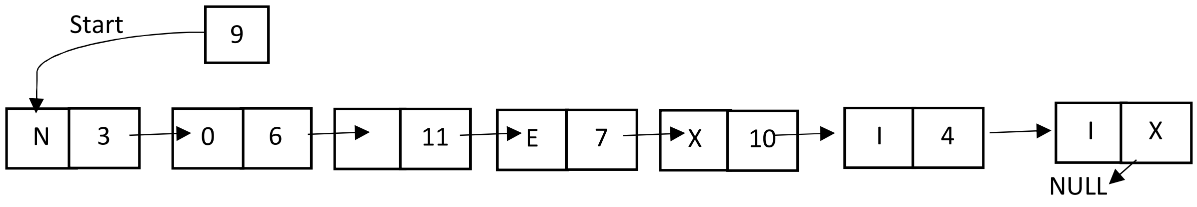
INFO [K]
LINK [K] contains respectively.

The information part and the next pointer field of a node linked list

II. List requires a variable name such as start which contains the location of the beginning of
the list and a next pointer denoted by null which indicates the end of the list.



	INFO	**LINK**
1		
2		
3	O	6
4	T	0
5		
6		11
7	X	10
8		
9	N	3
10	I	4
11	E	7
12		



- **Traversing a Linked List:-**
    - In a linked list INFO is the Information part, LINK is the address of the next element. We take PTR IS a pointer variable which points to the node that is currently processed.
    - Here START is a pointer which points to the first element of the LIST.
    - Initially we assign the value of START to pointer so PTR also points to the first element of the LIST.
    - For processing the next element we assign the address of the next element to PTR.

PTR:= START (Initially)

PTR:= PTR → Link (for processing next element)

- Now PTR has the address of the next element we can traverse. Each element of linked list through this untill PTR has null address so the linked list can be traversed as-
  While PTR:≠Null
  PTR:=PTR→ LINK/
  (LINK [PTR])

Algorithm:-

Let LIST be a linked list in memory. This Algorithm traverses LIST applying an operation process to each element of LIST. The variable PTR points to the node currently being processed

Step 1 - Set PTR: =START (initialises pointer PTR)

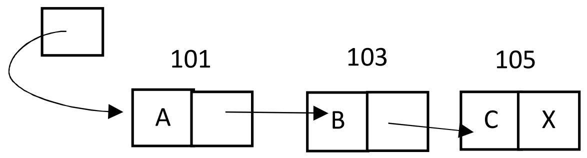
Step-2 -Repeat Steps 3 & 4 while P:≠ NULL

Step-3 -Apply Process to INFO (PTR)

Step-4 -Set PTR LINK := [PTR]

Step- 5 -Exit.

START



- **<u>Searching into a Linked List</u> :-**
  - ➢ Searching refers to search an element in linked list for searching the element we first traverse the linked list and with traversing the we compare the INFO part of each element with the given element.

- **<u>List is Unsorted</u>:-**

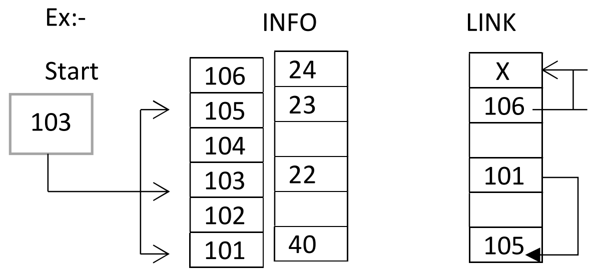
<u>Algorithm</u>:-

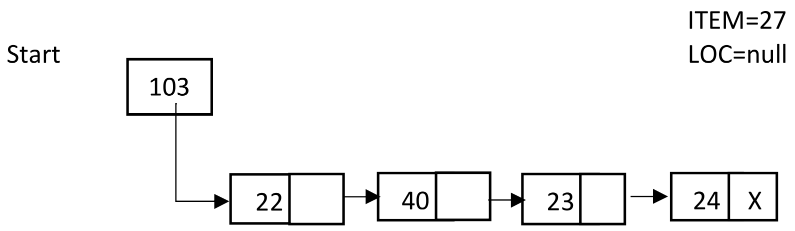
SEARCH (INFO, LINK, START, ITEM, LOC)

➢ LIST is a linked list in memory.
➢ This algorithm finds the location (LOC) of the node where ITEM first appears in LIST or sets LOC=NULL

```
Step 1 –Set  PTR :=START
Step 2 -Repeat Step 3 while PTR ≠NULL
Step 3 - If ITEM=INFO [PTR], then
              Set LOC:=PTR & Exit
        Else
              Set PTR :=  LINK [PTR] (PTR now point to next node)
Step 4-Set LOC := NULL (Search is unsuccessfull)
Step 5-Exit
```

Ex:-





- **List is Sorted**:-

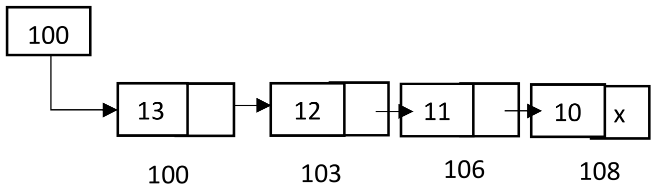
Algorithm:-

SEARCH (INFO, LINK, START, ITEM, LOC)

- LIST is a sorted list in memory
- This algorithm finds the location (LOC) of the node where ITEM first appears in LIST or sets

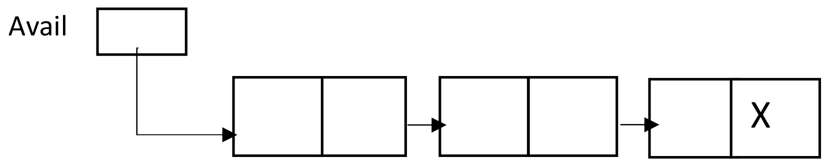
LOC=NULL.

Ex-> Start



```
Step-1 Set PTR := START
Step-2 Repeat Step 3 while PTR ≠ NULL
Step-3 If ITEM<INFO [PTR] then
        Set PTR := LINK [PTR]
       Else if
       ITEM:= INFO [PTR] then
        Set LOC=PTR and Exit (Search is successful)
        Else
        Set LOC=NULL and Exit (ITEM now exceeds  INFO[PTR])
Step-4   Set LOC: = NULL
Step-5   Exit
```

- **Garbage Collection**:-
  - Some memory space becomes re-usalable because a node is deleted from a list or an entire list is deleted from a program we want the available for future rule.
  - One way to bring this to immediately the space Into the free storage list.
  - The Operating System of a computer may periodically collect all the deleted space or unused   space on to the free storage list.



(Unused memory cells)

- The technique which does this collection is called Garbage Collection.
- Garbage collection usually takes place in two steps:

I. Tag memory cells which are currently in use.

II. Untagged space is collected onto the free storage list.

- **<u>Insention into a Linked List</u>:-**

<u>Insertion Algorithms</u>:-

- Algorithms which insert nodes into Linked List come up various situations. We discuss two of here-

The first one inserts a node at the beginning of the list, the second one inserts a node after the node with a given location.

(i)<u>Inserting at the beginning of a list</u>

<u>Algorithm</u>-

INSFIRST (INFO, LINK, START, AVAIL, ITEM)

<u>Step-1</u> If AVAIL: NULL then Write Overflow and Exit.

<u>Step-2</u> Set NEW:= AVAIL and AVAIL:=LINK [AVAIL].

(Remove first node from AVAIL. List)

<u>Step-3</u> Set INFO[NEW]:=ITEM (Copies new data into new node).

<u>Step-4</u> Set LINK [NEW] := START (New node now points to original first node)

<u>Step-5</u> Set START := NEW (Changes START so it points to to the new node)

<u>Step-6</u> Exit.

(ii)<u>Inserting After a given node</u>-

- Suppose we are given the value of Loc where either LOC is the location of a node A is a Linked List or LOC=NULL. The following is an algorithm which inserts ITEM into List so that ITEM follows node A or when LOC=NULL so that ITEM is the first node.

Let N denote the new o node (whose LOC is new)

If LOC= NULL then, N is inserted as the first node is List otherwise we l et node N point to node B by the assignment..

LINK [NEW] = LINK [LOC].

and we let node A point to the new node N by the assignment.

LINK [LOC]=NEW

Algorithm-

INS LOC (INFO, LINK, START, AVAIL, LOC, ITEM)

- This Algorithm inserts ITEM so that ITEM follows the node with Location (LOC) or inserts the ITEM as the first node when LOC = NULL

```
Step-1 AVAIL:=NULL
            Write Overflow or Exit
Step-2 Set NEW := AVAIL
    and AVAIL := LINK [AVAIL]
(Remove first node from AVAIL List)
Step-3 Set INFO [NEW] := ITEM (Coples new data into new node)
Step-4 If LOC=NULL, then (insert as first node)
      Set LINK [NEW]:= START and
      START : =NEW
      Else: [Insert after node with location (LOC)].
      Set LINK[NEW]:= LINK [LOC]
        And LINK [LOC]:=NEW
             (End of if structure)
Step-5 Exit
```

- **Deletion from a Linked List:-**

Algorithm-

DEL (INFO,LINK, START, AVAIL, LOC, LOCP)

This Algorithm deletes the node in with Location (LOC). LOCP is the location of the node which precedes N or when N is the first node, LOCP=NULL.

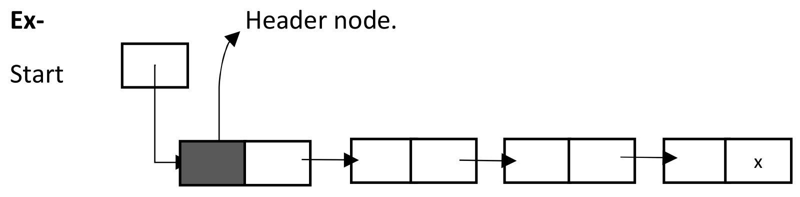
```
Step-1- If LOCP := NULL then
        Set START:=LINK [START] (delete first node)
          else
            Set LINK [LOC P]:=LINK [LOC] (deletes Node N)
Step-2-Set LINK[LOC]:= AVAIL and
         AVAIL:= LOC (Deleted node to the AVAIL List)
Step-3-Exit
```

- **Header Linked List:-**
  - A header linked list is a linked list which always contains a special node called the header node at the beginning of the list
  - The following are two kinds of widely used header list-

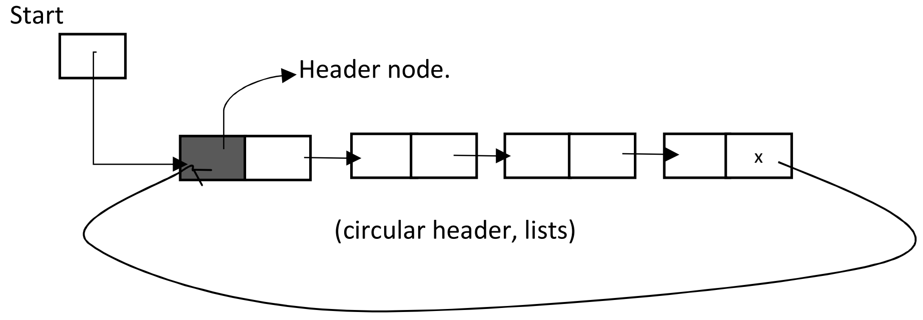
    I. **Grounded Header List**-> A grounded header list is a header list where the last node contains the null pointer.

    II. **Circular Header List** -> A circular header list is a header list where the last node points back to the header node

Ex-



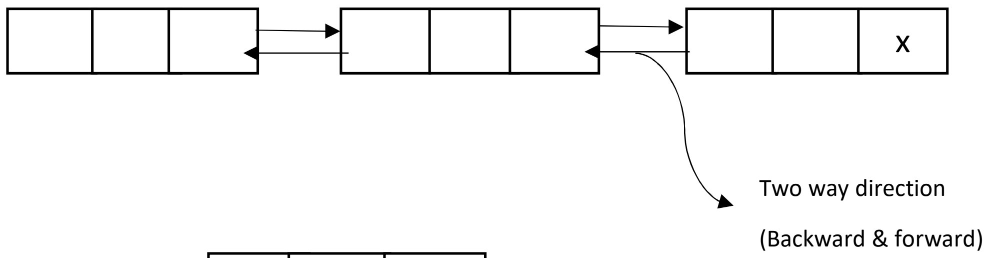
(grounded theadon list)

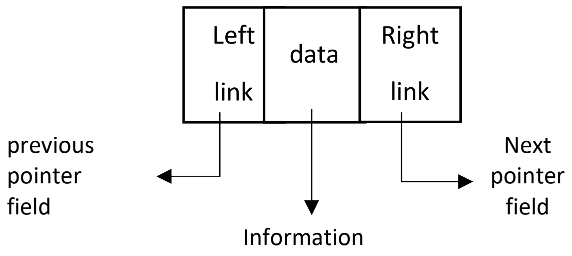


(circular header, lists)

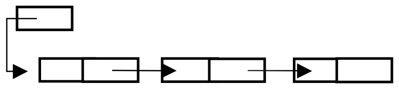
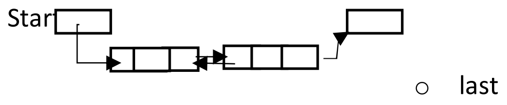
  - The list pointer start always points to the header node.
  - Accordingly LINK [START] = NULL indicates that a grounded header list is empty and LINK [START] = START indicates that a circular header list is empty.

- **Double Linked List:-**
  - In single linked list one can move starting from the first node to any ride in one direction only (left to right). So, single linked list also Known as one way list.
  - Double linked list is two way list.
  - In double linked list we can move either direction.
  - In double linked list a node has atleast three field say data, left link and right link.


x
Two way direction
(Backward & forward)


Left link
data
Right link
previous pointer field
Information
Next pointer field

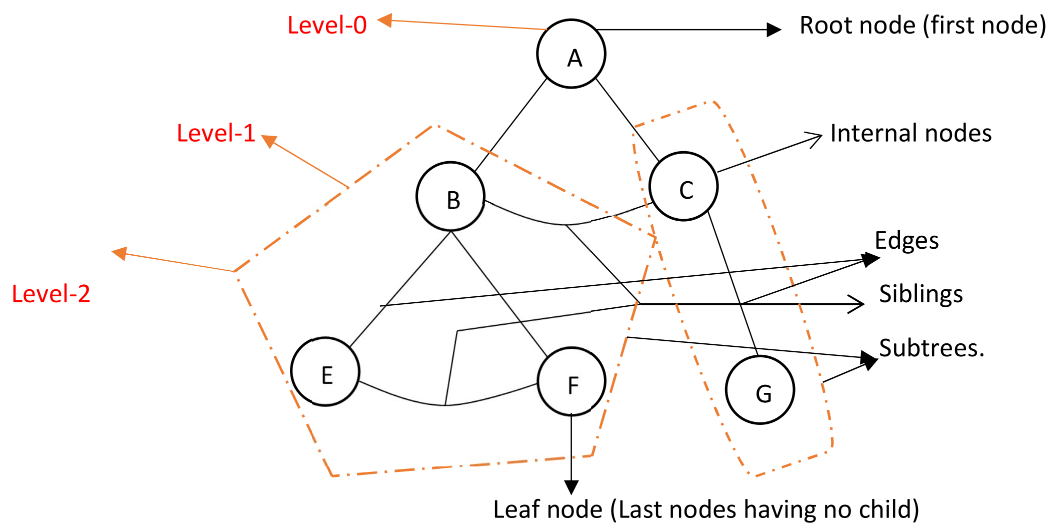
• Difference between Single & double linked lists:-

Single Linked List	Double Linked List
➢ It has nodes with data field and next linked field. (forward linked list)	➢ It has nodes with data field and two pointer field. (Backward & Forward pointer field).
➢ It requires one list pointer variable 'start' start 	➢ It requires two list pointer variable start and last or first and last. 
➢ In single linked list the traversal can be done in one direction.	➢ In double linked list traversal can be done in two directions.
➢ It occupies less memory	➢ It occupies more memory
➢ Less efficient access to the element	➢ More efficient access to the element.

Chapter-6

TREE

- **Basic Terminology of Tree :-**
  - ➢ Tree is a non-linear data structure which shows parents-child relationship among the nodes.
  - ➢ It organises data in hierarchical manner.
- **Tree Terminology**:-
- **Node-**> Each element of the tree is a node.



•**Edges**→ The lines which connects one node to another node is called Edge.

•**Degree of node**→ It is the number of child.

Ex- degree of B = E, F →2.

degree of C = G →1.

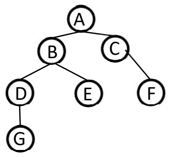
degree of G=O.

•**<u>Degree of tree</u>**→ It is the maximum number of nodes ore child.

Ex- B & C→2.

•<u>Height</u> →Bottom to Top (Longest path).

Ex-



Height of A = 3

Height of C= F to C →1

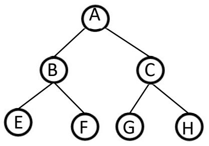
Height of F = 0

- **<u>Parent Node & Child Node</u>-**

Parent nodes $\left\{\begin{matrix} A \to B \,\&\, C \\ B \to E \,\&\, F \\ C \to G \end{matrix}\right\}$ Child Nodes

- **<u>Depth Level</u>** → Root node to node (similar as Level)

Ex-



Depth of E=2.

(E TO A)

Depth of B=1

(B to A)

•**<u>Siblings</u>** → E&F (Parrent same)

B & C (Parent same) } ıen there is the nodes having parent it is known as Siblings

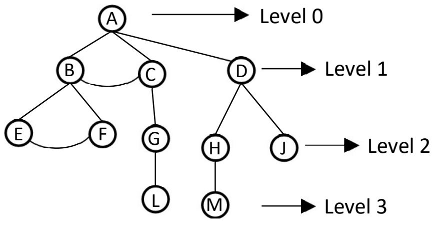
•**<u>Predecessor</u>** → C node→A

B node→ A

•<u>Successor→</u> C node → G

B node → E&F

Ex-



1) **Nodes -** A, B, C, D, E, F, G, H, J, L,M.
2) **Root Node -** A
3) **Leaf Node -** E, F, L, M, J
4) **Internal Node -**B, C, D, E, F, G, H, J.
5) **Degree of Node -** A = 3 , B = 2 , c = 1 , D = 2 , G = 1, H = 1 ,
L = 0 , M = 0
6) **Degree of tree -**B, C, D = 3
7) **Height-**
8) **Parent Node & Child Node-**

$$\textbf{Parent nodes,}\left\{\begin{matrix} \boldsymbol{A \rightarrow\ B, C, D} \\ \boldsymbol{B \rightarrow E, F(2)} \\ \boldsymbol{C \rightarrow G\ (1)} \\ \boldsymbol{D \rightarrow H, J(2)} \\ \boldsymbol{G \rightarrow L\ (1)} \\ \boldsymbol{H \rightarrow M.(1)} \end{matrix}\right\}\textbf{no of child nodes}$$

9) **Levels** -Level 0, Level 1, Level 2, Level 3.
10) **Depth** – 3
11) **Predecessor-**

B→A	E→B	H→D	M=H
C→A	F→B	J→D	
D→A	G→C	L→G	

12)Successor-

A→B,C,D

B→E, F

C→G

G→L

D→H, J

H→M.

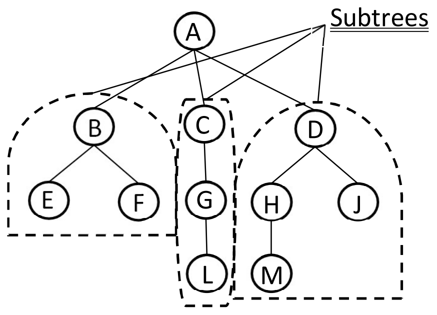
12) **Siblings**-

A → B & C

B → E & F

D → H & F

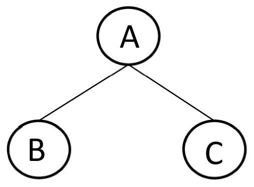
13) **Subtree**-



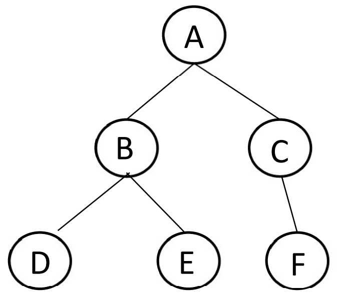
Binary Tree:-

- Binary tree is a special tree data structure in which every node has at most 2 child.

Ex-



- There has 2 child or 0 or 1

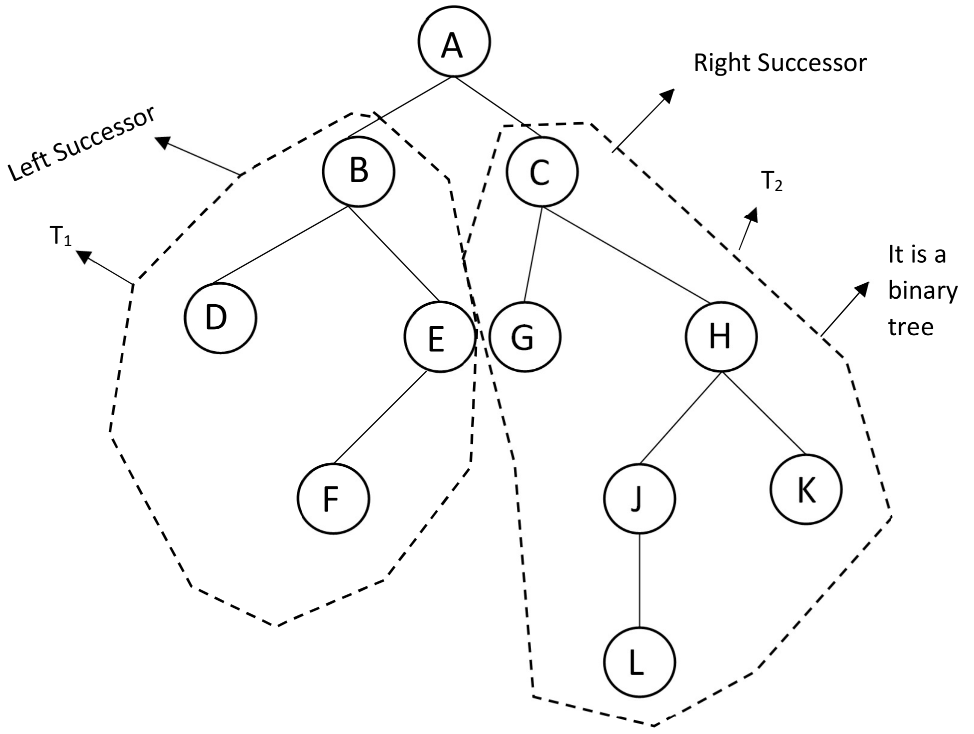


- A Binary Tree (T) is defined as finite set of elements called nodes.

(a) T is empty called the Null tree or Empty tree.

(b) T contains a distinguished node R, called the Root of T and the remaining nodes of T form an ordered pair of disjoint binary trees T, & T_2

- If T does contain a Root R then the two trees T_1& T_2 are respectively, the left and right sub-trees of R..
- If T_1 is non-empty then it's Root is called the left successor of R. Similarly if T_2 is non-empty then its Root is called the right successor of R.



(a) B is a left left successor and C is a sight successor of the node A

(b) The left subtree of the node B,D,E,F and the right subtree of the Root A consists of the node: C, G, H, J, K, L,

(c) Any node N in a binary tree. T has either 0, 1, or 2 successors. The nodes A, B, C and H have 2 successors, the node E and j have only one successor and the node D, F, G, L., K has no successor are called Terminal node or leaf node.

- **<u>Representation of Binary Tree in Memory</u>:-**
    - Binary Tree can be represented by two ways in memory

i. Linked Representation.
ii. Sequencial Representation.

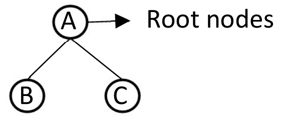
1) **<u>Linked Representation</u>** :-

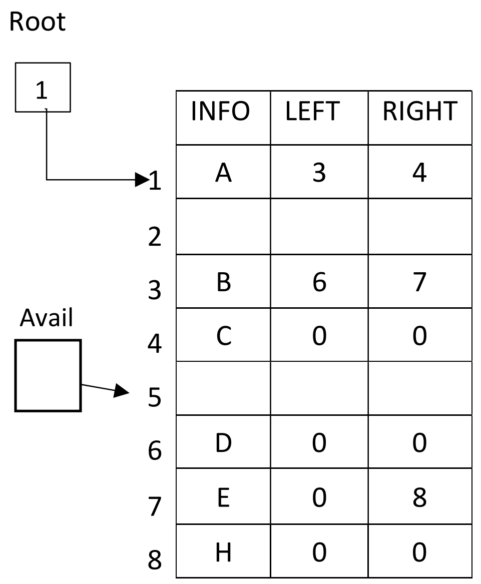
- In this representation a binary tree T is represented by linked list in memory
- T will be maintained in memory by means of a linked representation which uses three parallel arrays, INFO, LEFT and RIGHT and a pointer variable ROOT as follows-

(1) First of all each node N of T will correspond to a location K such that:
  (a) INFO [K] contain the data at the node N.
  (b) LEFT [K] contains the location of the left child of the node N.
  (c) RIGHT [K] contains the locations of the right child of the node N

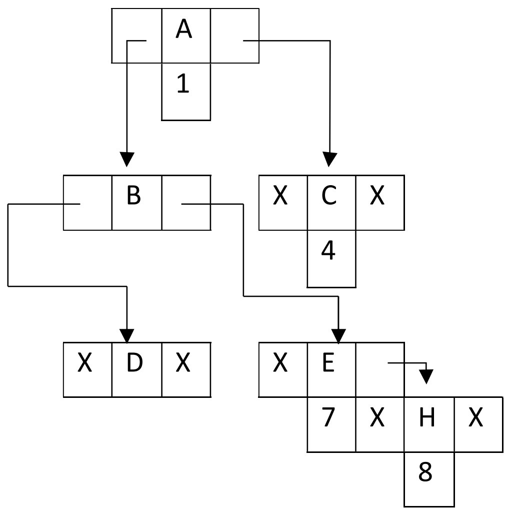
(2) ROOT will contain the location of ROOT R of T. If any subtree is empty then the corresponding pointer will contain the Null value, if the tree T itself is empty then ROOT will contain the Null value.

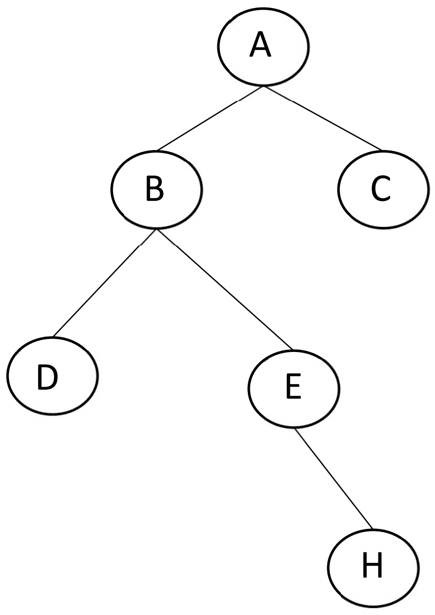
EX-





	INFO	LEFT	RIGHT
1	A	3	4
2			
3	B	6	7
4	C	0	0
5			
6	D	0	0
7	E	0	8
8	H	0	0

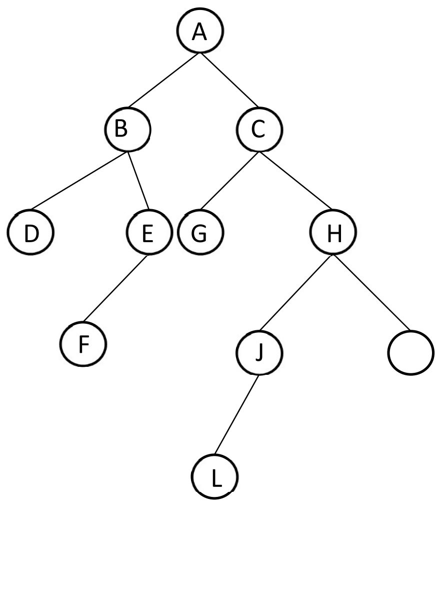




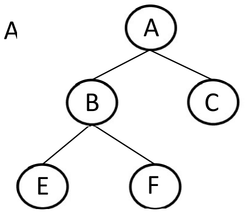
2) Sequencial Representation:-

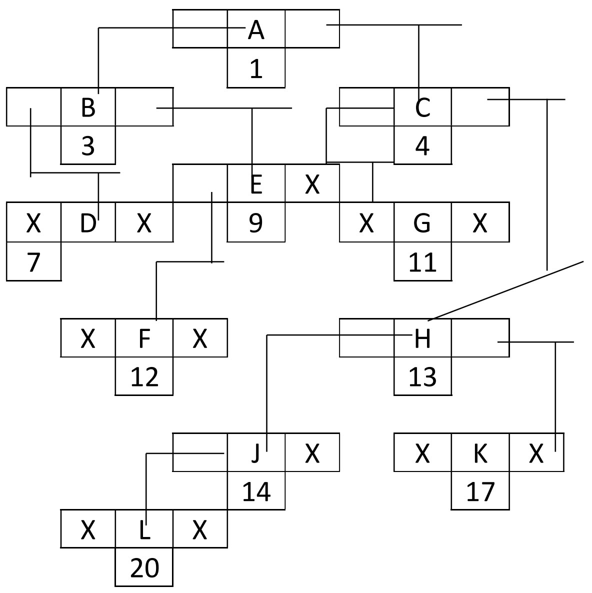
- This representation uses only a single array Tree as follows:-
  - (a) The Root R of T is stored in Tree [1].
  - (b) If a node N occupies TREE [K], then it's left child is stored in TREE [2*K] and its right child is stored in TREE [2*K+1].
  - (c) Again Null is used to indicate an empty subtree In particular TREE [1] = Null indicates that the tree is empty.

Solve this using Link Representation:-



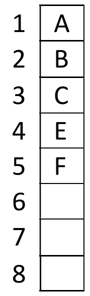
TREE [K]
TREE[1]element A
=NULL
The left child will stored in
TREE [2*K]
=2*1
=2 (element B)
The right child will stored
TREE [2*K+1]
= [2*1+1]
= [241]
=3 ->(element C)
TREE [K]
TREE [2]





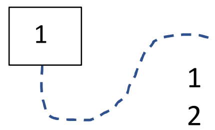
The right child will stored in->
TREE [2*K+1]
=[2*1+1]
=[241]
=3 ->(element c)
TREE [K]
TREE [2]

The left child will stored in



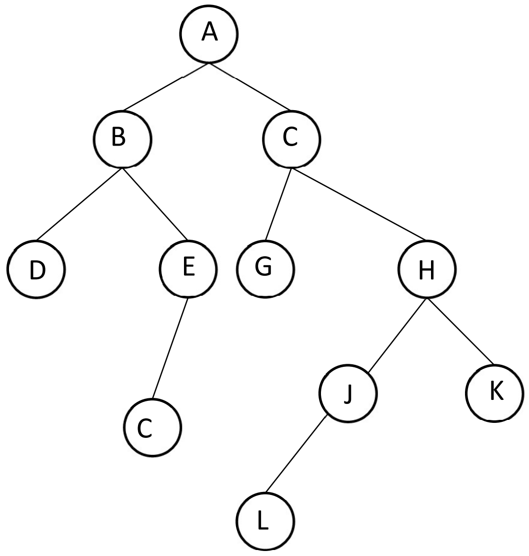
TREE [2K]
2*2
=4 ->(element E)
The right child stored in-
TREE [2K+1]
2*2+1
4+1
=5 ->(element F)

ROOT



	INFO	LEFT	RIGHT
1	A	3	4
2			
3	B	7	9
4	C	11	13
5			
6			
7	D	0	0
8			
9	E	12	0
10			
11	G	0	0
12	F	0	0
13	H	14	17
14	J	20	0
15			
16			
17	K	0	0
18			
19			
20	L	0	0

Q.**Solve this using Sequential Representation**.



1	A
2	B
3	C
4	D
5	E
6	G
7	H
8	
9	
10	F
11	
12	
13	
14	J
15	K
16	
17	
18	
18	
19	
20	
21	
22	
23	
24	
25	
26	
27	
28	L

TREE [K]

TREE [1]-element A

<u>The left child will stored in</u>-

TREE [2*K]

=2*1

=2 $\rightarrow$ element B

<u>Right child</u>-

TREE [2*K+1]

=2*1+1

=2+1

=3 $\longrightarrow$ Element c

<u>Left child</u>-

TREE [2 * K] 2*2=4 $\longrightarrow$ element D

<u>Right Child</u>-

TREE [2*K+1]

= [2*2+1]

=4+1

=5 $\rightarrow$ element E

TREE [K]

TREE [5]

<u>Left child</u> →
TREE [2*K]
=2*5
=10 → F
TREE [K]
TREE [3] → C
<u>Left child</u> ⟶
TREE [2*K]
=2*3
6 → G

<u>Right child</u>-
TREE [2*K+1]
=2*3+1
=7 → H
TREE [K]
TREE[7] → H

<u>Left child</u>-
TREE [2*K]=2*7
=14 → j

<u>Right child</u>-
Tree[2*K+1]
=2*7+1
=15 →K
Tree[K]
Tree[14] →J

<u>Left child</u>-
Tree[2*K]
=2*14
=28 → L

• <u>Binary Tree Traversal</u>:-

- ➢ Traversing a tree means accessing every node once at a time. There are 3 ways of traversing a binary free.

(a) **Pre order →** Root Left subtree Right subtree
(b) **Inorder→** Left subtree Root Right subtree
(c) **Post order→** Left subtrice Right subtree Root.

(1) <u>Pre order</u>:-

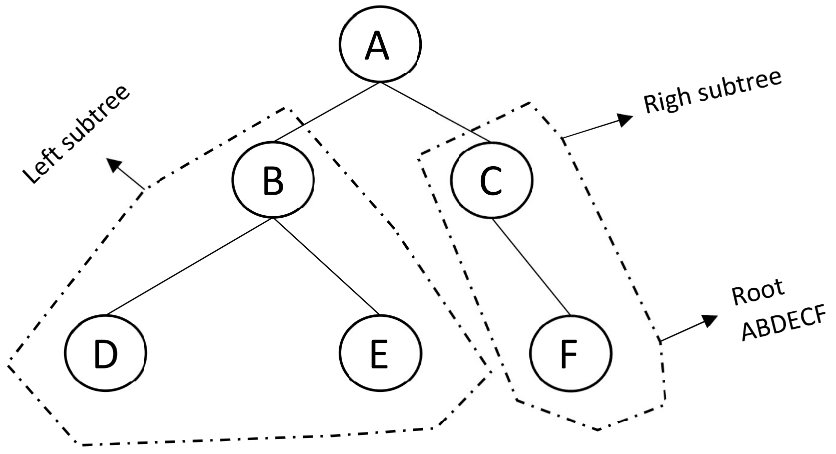
<u>Algorithm</u>-

<u>Step-1</u>→ Process the root R.

<u>Step-2</u>→ Traverse the Left subtree of R in preorder.

<u>Step-3</u>→ Traverse the right subtree

EX-



(2) <u>In order</u>:-

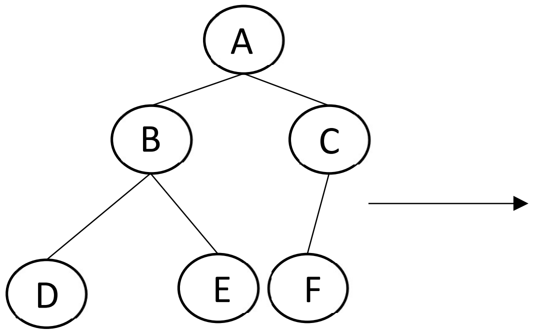
<u>Algorithm</u>:-

<u>Step-1</u> →Traverse the left subtree of R in inorder.

<u>Step-2</u> →Process the root R.

<u>Step-3</u> →Traverse the right subtree of R in inorder.

EX-



Left Root right

DBEAFC

(3) Post order:

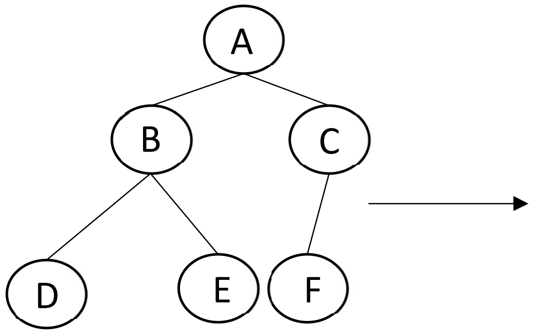
Algorithm:-

Step-1→ Traverse the left subtree of R in post order.

Step-2 → Traverse the right subtree of R in post order.

Step-3 → Process the root R.

EX:-



Left Right Root

DEBFCA

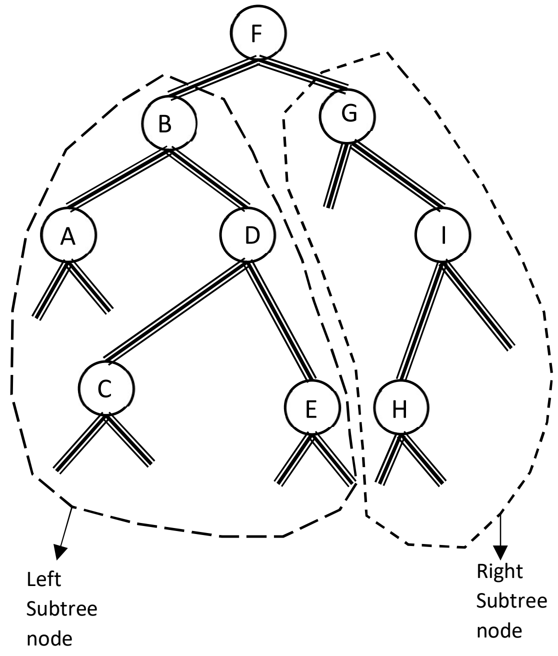
- **One shortcut method:-**

Scanning Method- (How to check the correct answer after solving by the Algorithm steps)

PRE ORDER-1st time access node

INORDER- 2^{ND} time access node

POST ORDER-3^{rd} time access node



Result

PRE-FBADCEGIH

IN-ABCDEFGHI

POST-ACEDBHIGF

R_{00} L R

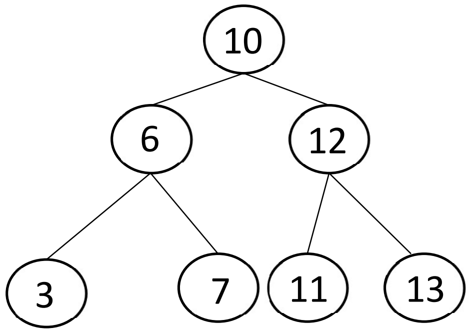
L R_{00} R

L R R_{00}

Same answer will obtain by Algorithm rules

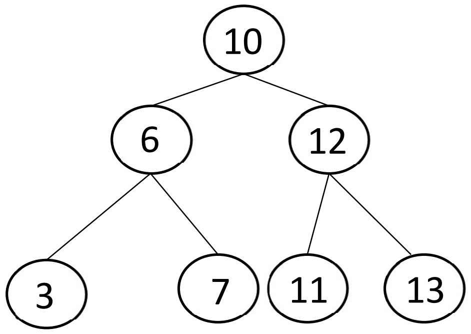
- **<u>Binary Search Tree</u>:-**
  - A Binary Search Tree is a binary tree in which
    i) All nodes of left subtree are less than root node.
    ii) All nodes of right subtree are greater than the root node.

EX-



- **<u>Searching</u>->**
  - Suppose T is a Binary Search Tree. An ITEM of information is given the following algorithm finds the location of ITEM in the Binary Tree T.
  - Compare ITEM with the root node N of the tree-
    i> If ITEM <N, proceed to the left child of N.
    ii> If ITEM >N, proceed to the right child of N.

Ex:-



Suppose search ITEM= 7 then, we will first see towards the root node i.e. 10 so, 7<10, then if 74<0 then according to the above cond^n ,we will proceed to the left child then there is. 6 which is the parent node then 7>6 then if 7>6, according to the above cond" we will shift to right node then, 7 = 7). Then the searching is stopped. Because we were searching for the ITEM=7, and we found that'

• <u>Insertion of Binary Search Tree</u> :-

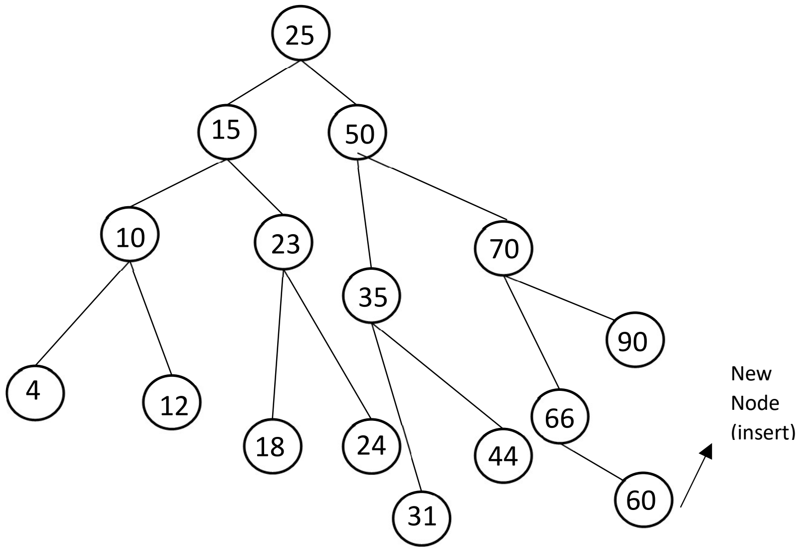
- ➢ Always insert new node as leaf node.
- ➢ Start at root node as current node.
- ➢ if new node's key < current's key

(i) If current node has a left child search left

(ii) Else add new node as current's left child:

- ➢ if new node's Key > current's Key
  - (i) If current node has a right child, search right.
  - (ii) Else add new node as current's right child.

EX-



According to the above rules, new node = 60 and the current nade 25 Then we insert at the last or after the leaf node.

• <u>Deletion of Binary Search Tree</u> :-

- ➢ Suppose T is a Binary Search tree

and suppose on ITEM of information is given.

- ➢ These section gives an algorithm which

deletes ITEM from the Tree T.

→ No chidren (leaf node)
→One child
→Two child

(Non-leaf node)/
(internal node)

- ➢ The way N is deleted from the Tree depends primary on the no. of children node N.
- ➢ There are three cases-

Case(i)-N has no children.

Then N is deleted from T by simply replacing the Location of N in the parent node P[N] by the Null painter.

Case (ii)- N has exactly one child.

Then N is deleted from T by simply replacing the location of N in P[N] by the location of the only child of N.

Case (iii)- N has two children.

Let S[N] denote the inorder successor of N. (The reader can verify that S[N] desnot have a left child).

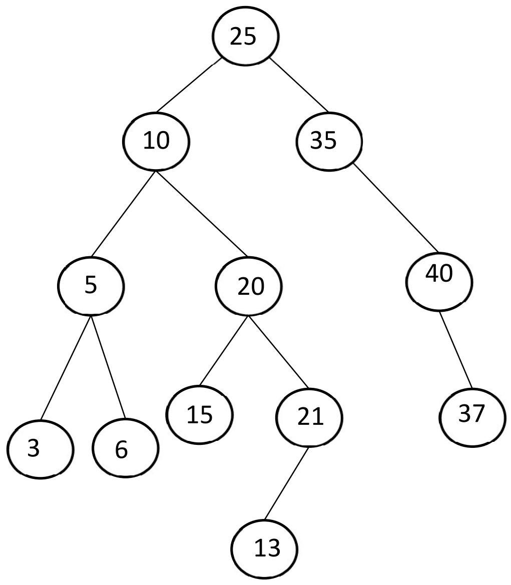
Then N is deleted from T-by first deleting S[N] from T (by using Case (i) or Case (ii) and then replacing node N in T by the node S[N].

Algorithm-

- Perform Search for value x
- If x is a leaf, delete x
- Else internal node has to be delete.

(a) Replace with largest value y on left subtree or smallest value z on right subtree.

(b) Delete replacement value (y and x) from subtree.


25
10
35
5
20
40
3
6
15
21
37
13