DATA STRUCTURE

Data : Anything to give information is called data.

Ex:- Student Name, Student Roll no.

Structure : Representation of data is called structure.

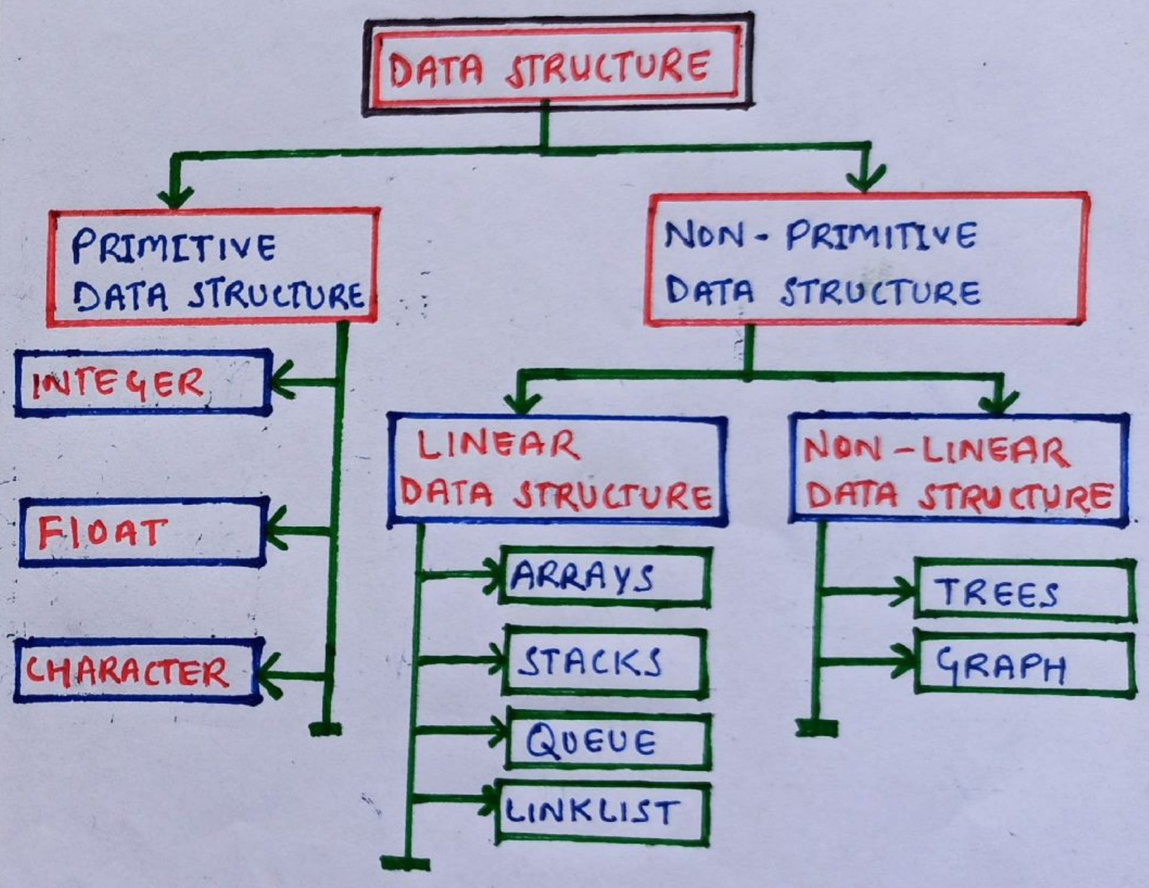
Ex:- Graph, Arrays, List.

Data structure :

- Data structure = Data + structure.
- Data structure is a way to store and Organize data so that it can be used efficiently (better way).
- Data structure is a way of organizing all data items and relationship to each other.

Types of Data structure :

There are mainly two types of data structure.



Primitive data structure : These are basic structure and are directly operated by machine instruction.

Ex:- integer, float, character.

Non-Primitive data structure:

These are derived from the primitive data structure. its a collection of same type or different type primitive data structure.
Ex:- Arrays, stack, trees.

Data Structure Operation

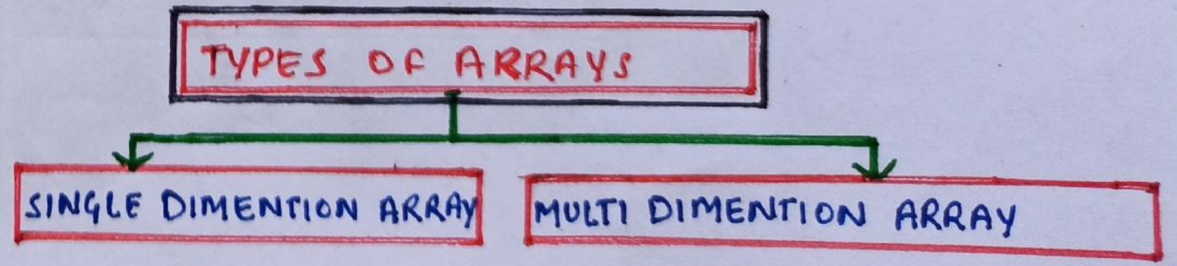
The data which is stored in our data structure are processed by some set of operation

1). **Inserting:** Add a new data in the data structure.
2). **Deleting:** Remove a data from the data structure.
3). **Sorting:** Arrange data increasing or decreasing order.
4). **Searching:** Find the location of data in data structure.

5). **Merging:** Combining the data of two different stored files into a single stored file.
6). **Traversing:** Accessing each data exactly one in the data structure so that each data items is traversed or visited.

Arrays

- An Array can be defined as an infinite collection of homogeneous (similar type) elements.
- Array are always stored in consecutive (specific) memory location.
- Array can be stored multple values which can be referenced by a single name.



1). Single Dimentional Arrays :

It is also Known as One Dimentional (1D) Array.

- It's use only one subscript to define the elements of Arrays.

[row] [column]

Declaration :

data-type var_name[expression];

Ex :- int num[10];
char c [5];

(expression → size)

Initializing One-Dimentional Array :

Data-type var_name[Expression] = {values};

Ex :- int num[10] = {1,2,3,4,5,6,7,8,9,10};
char a[5] = {'A', 'B', 'C', 'D', 'E'};

2). Multi-Dimentional Arrays :

Multi-Dimentional Arrays use more then one subscript to describe the Arrays elements.

[][][]---

Two Dimentional Arrays :

It's use two [row][column] subscript, one subscript to represent row value and second subscript to represent column value.

It mainly use for matrix Representation.

Declaration two Dimentional Arrays :

Data-type var-name[rows][column]

Ex :- int num[3][2]

Initialization 2-D Arrays :

data-type var-name[rows][column] = {values};

Ex ⇒ int num[3][2] = {1,2,3,4,5,6};

or

int num[][] = {1,2,3,4,5,6};

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}_{3\times 2}$$

num[0,0] = 1
num[01] = 2
num[10] = 3
num[11] = 4
num[20] = 5
num[22] = 6

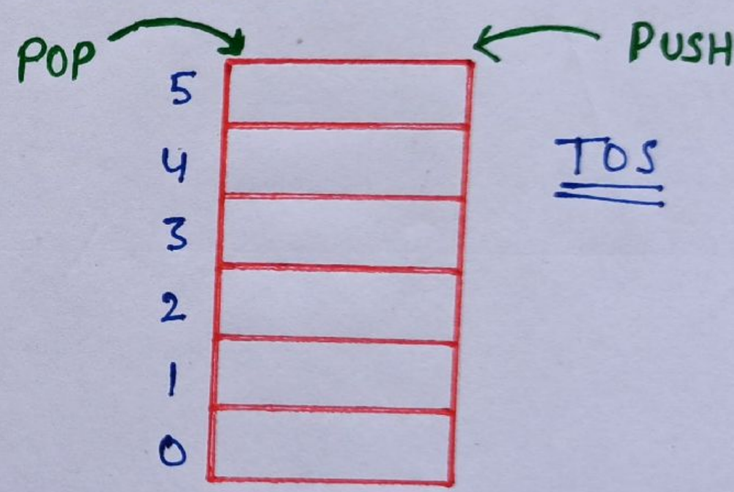
Write a program to read & write one Dimentional Array.

```
include <stdio.h>  ──→ Standard input out Printf Scanf
include <conio.h>  ──→ Console input out clrs(rc), getch()

void main()
{       int a[10], i;
        clrs(rc);
Printf("Enter the Array Elements");
   for(i=0; , i<=9; , i++)
   {  scanf("%d", &a[i]);
   }
 Printf("the entered Array is");
   for(i=0; i<=9; i++)
   {
   Printf("%d\n", a[i]);
   }
    getch();
}
```


Stacks (Data Structure)

- Stack 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 Top of stack (TOS)



- The last added element will be the first to be Removed from the stack.

This is the reason stack is called Last-in-first-out (LIFO) type of list.

Operations on stack

There are two operation of stack.

1). **Push Operation :** • The process of adding a new element to the top of stack is called Push Operation.

- Every new element is adding to stack top is incremented by **one**
- In case the array is fall and no new element can be added it's called **Stack full** or **Stack Overflow** condition.

2). **POP Operation :** • The process of deleting an element from the top of stack called POP operation

- After every POP operation the stack (TOP) is decremented by **One**.
- If there is no element on the stack and the POP is performed then this will result into **stack underflow** condition.

Stack Operation & Algorithm

Stack has two Operation.

1). PUSH operation.
2). POP Operation.

1). **PUSH Operation :** • The process of adding a new element of the top of stack is called PUSH Operation

- Every PUSH operation Top is incremented by one.

```
TOP = TOP + 1
```

- In case the Array is full no new element is added. this condition is called **Stack full** or **Stack Overflow** condition.

Algorithm for Inserting an item into the stack (PUSH Operation).

PUSH (stack [max size], item)

Step 1: initialize

set top = -1

Step 2: Repeat steps 3 to 5 until Top < maxsize-1

Step 3: Read Item.

Step 4: Set top = top + 1

Step 5: Set stack [Top] = item

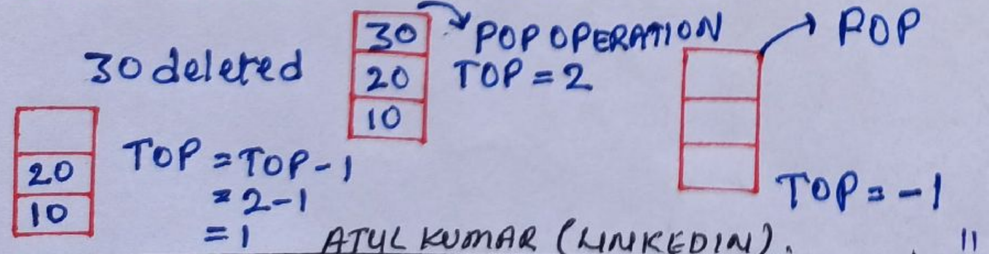
Step 6: Print "Stack overflow"

2. POP Operation :

- The process of deleting an element from the top of stack is called **POP Operation.**
- After every POP Operation the stack TOP is decremented by one.

TOP = TOP - 1

- If there is no element on the stack and the POP operation is performed then this will result into **STACK UNDERFLOW.** condition.



Algorithm for deleting on item from the stack (POP)

POP (stack [max size], item)

Step 1: Repeat steps 2 to 4 until Top ≥ 0.

Step 2: Set item = stack [TOP].

Step 3: Set top = top - 1

Step 4: Print, No. deleted is, Item

Step 5: Print stack under flows.

Stacks (Prefix & Postfix)

1). **Infix Notation :** Where the operator is written in between the operands.

Ex :- A + B + operator A, B operands.

2). **Prefix Notation :** In this operator is written before the operands.

It is also known as Polish Notation.

Ex :- + A B

3). **Postfix Notation :** In this operator is written after the operands.

It is also known as Suffix Notation.

Ex :- A B +

Q⇒ Convert the following Infix to Prefix and postfix for (A+B) * C/D + E^F/G

Prefix ⇒ (A+B) * C/D + E^F/G

+ AB * C/D + E^F/G

Let +AB = R_1

R_1 * C/D + E^F/G

R_1 * C/D + ^EF/G

Let ⇒ ^EF = R_2

R_1 * C/D + R_2/G

R_1 * /CD + R_2/G

Let ⇒ /CD = R_3

R_1 * R_3 + R_2/G

R_1 * R_3 + /R_2G

Let ⇒ /R_2G = R_4

R_1 * R_3 + R_4

* R_1 R_3 + R_4

Let * R_1 R_3 = R_5

R_5 + R_4

+ R_5 R_4

Now Enter the value of R_5, R_4, R_3, R_2, R_1

+ * R_1 R_3 / R_2 G

+ * +AB/CD/^EFG

Postfix ⇒ (A+B) * C/D + E^F/G

(AB+) * C/D + E^F/G

Let AB + = R_1

R_1 * C/D + E^F/G

R_1 * C/D + (EF^)/G

Let EF^ = R_2

R_1 * C/D + R_2/G

R_1 * (CD/) + R_2/G

Let CD/ = R_3

R_1 * R_3 + R_2/G

R_1 * R_3 + (R_2G/)

Let R_2G/ = R_4

R_1 * R_3 + R_4

(R_1 R_3 *) + R_4

Let R_1 R_3 * = R_5

R_5 + R_4

R_5 R_4 +

Now enter the value of R_5, R_4, R_3, R_2, R_1

R_5 R_4 +

R_1 R_3 * R_4 +

AB + CD/ * R_2 G/ +

AB + CD/ * (EF^) G/ +

Postfix expression.

Prefix and Postfix using tabular form

Ex :- Convert (A + B * C) into prefix and postfix using tabular form

to convert in Prefix following operation

Program

1). Reverse the input string
2). Perform tabular method and find postfix Expression.
3). Reverse this postfix Expression string to find the Prefix.

Ex :- A + B * C

first to add branches

(A + B * C)

Reverse string

(C * B + A)

Priority
^ → highest
*, / → 2 highest
+ - → Lowest Property.

Tabular form Symbol Scanned	Stack	Postfix Expression
(	(	-
C	(	C
*	(*	CB
B	(*	CB*
+	(+	CB*A
A	(+	CB*A+
)	-xy	

So the postfix Expression CB*A+. Now reverse this Expression to get the prefix so prefix is +A*BC Prefix

Convert postfix ⇒ Direct perform tabular form (A + B * C)

Symbol Scanned	Stack	Postfix Expression
(	(	-
A	(	A
+	(+	AB
B	(+	AB
*	(+*	ABC
C	(+*	ABC*
)	lr	ABC*+

Postfix Expression = ABC*+

QUEUES

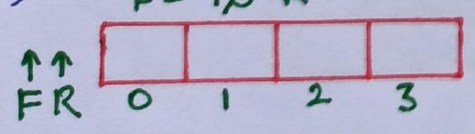
- Queue is a Non-Primitive Linar data structure.
- It is an homogeneous collection of elements in which new elements are added at one End called the Rear End, and the existing element are deleted from other End called the Front End.
- The first added element will be the first to be remove from the queue. that is the reason queue is called (FIFO) First-in-First out type list.
- In queue every Insert operation Rear is incremented by one.

$$R = R+1$$

and every deleted operation front is incremented by one.

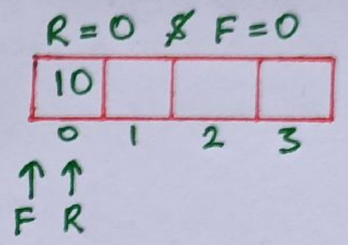
$$F = F+1$$

Ex ⇒

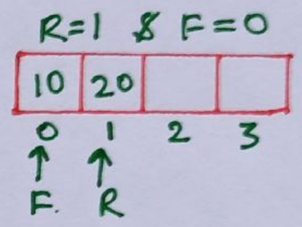


Empty queue

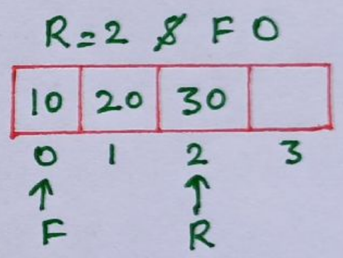
Insert 10



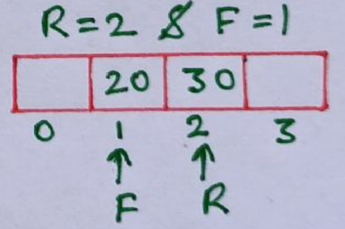
Insert 20



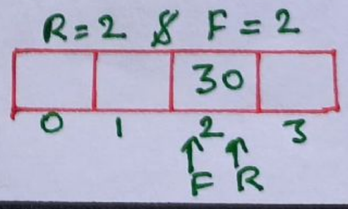
Insert 30



delete element. First delete 10



delete second element.



Operation on Queue

1). To insert an Element in a Queue :

Algo : QINSERT [QUEUE[maxsize], Item]

Step 1: Initialization
set front = -1
set Rear = -1

Step 2: Repeat steps 3 to 5 until
Rear < maxsize -1

Step 3: Read item

Step 4: if front == -1 then
front = 0
Rear = 0
else
Rear = Rear + 1

Step 5: set QUEUE [Rear] = item

Step 6: Print, Queue is overflow.

2). To delete an element from the Queue:

QDELETE (Queue[maxsize], item)

Step 1: Repeat step 2 to 4 until
front >= 0

Step 2: Set item = Queue [front]

Step 3: If front == Rear
set front = -1
set Rear = -1
else
front = front + 1

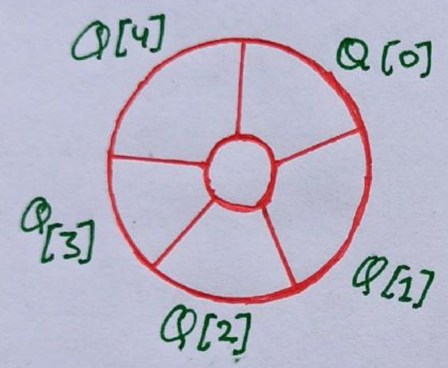
Step 4: Print, No. Deleted is, item

Step 5: Print "Queue is Empty or underflow.

CIRCULAR QUEUE

A Circwar queue is one In which the insertion of a new element is done at the very first location of the queue if the last location of queue is full.

F = R = -1



A Circwar queue overcome the problem of unutilized space in Linar queues implemented as arrays.

Circwar queue has following condition:

1). Front will always be pointing to the first element.

2). If front = Rear the queue will be empty.

3). Each time a new element is inserted into the queue the Rear is incremented by one.

Rear = Rear + 1

4). Each time an element is deleted from the queue the value of front is Incremented by one.

Front = Front + 1

Insert an element in circular queue:

Algo ⇒ QINSERT (QUEUE[maxsize], Item)

Step 1 ⇒ If (front = = (Rear+1) %. maxsize)

write queue is over flow & Exit.

Else: take the value

if (front = = -1)

set front = 0

Rear = 0

Else

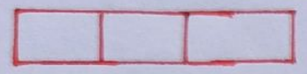
Rear = ((Rear+1) %. maxsize)

[Assign value] Queue[Rear] = value.

[End if].

Queue (Data Structure)

Operation on Queue

Ex:-  10, 20, 30, 40.

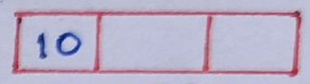
maxsize = 3

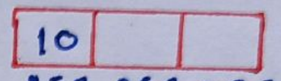
1). Front = ~~-1~~ Empty queue
Rear = -1

- 3 to 5 step Repeat
  R < maxsize -1
  -1 < 3-1
  -1 < 2 true
  ↙ ↓ ↘
  3 4 5

- Read item
  Read 10
- F = = -1
  -1 = = -1 true
  F = 0
  R = 0

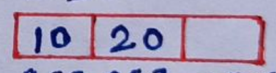
- Set q[0] = item
  q[0] = 10



Queue 

F = 0, R = 0
Rear < maxsize -1
0 < 3 -1
0 < 2 true
Read 20

- if F = = -1
  0 = = -1 false
  Else
  R = R + 1
  R = 0 + 1
  R = 1
- q[1] = 20



F = 0, R = 1

2) Rear < maxsize -1
1 < 3 -1
1 < 2 true
Read 30
if f = = -1
0 = = -1 false
Else
R = R + 1
R = 1 + 1 = 2

- set q[2] = 30

F = 0, R = 2

case1 Rear < maxsize -1
2 < 3 -1
2 < 2 false

- Queue is overflow.

Delete an Element in Circwar queue :

Algo : QDELETE (Queue [maxsize], Item)

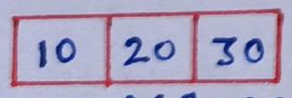
1). if (front = -1
write queue underflow and Exit
Else: item = Queue [front]
if (front = = Rear)
set front = -1
set Rear = -1
Else: front = ((front + 1) % maxsize)
[end if statement]
→ item deleted.

2). Exit.

QUEUE (Data structure)

Delete operation on Queue.

Ex :-



maxsize = 3

F=0 , R = 2

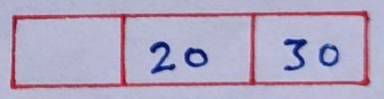
Case 1). 1). f >=0

0 >= 0 true.

2) set item = q [0]
item = 10

3). f = = R
0 = = 2 false
Else
f = f + 1
F = 0 + 1 = 1

4). item is deleted
10 is deleted.



F = 1 R = 2

F = 1 R = 2

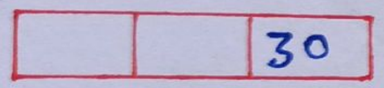
Case 2).

1). F >= 0
    1 >= 0 true

2). item = q[1]
    item = 20

3). if f == R
    1 == 2 False
    else
        F = F+1
        F = 1+1 = 2

4). Item is deleted
    20 is deleted.



F=2 R=2

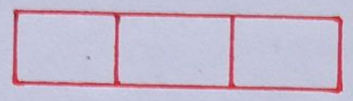
Case 3).

1). F >= 0
    2 >= 0 true

2). item = q[2]
    item = 30

3). if F == R
    2 == 2 true
    set f = -1
    R = -1

4). Item is deleted.



F = -1
R = -1

Case 4).

F >= 0
-1 >= 0 False

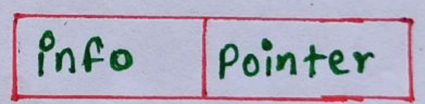
Step 5: Queue is empty is Underflow.

Linked Lists.

- A Linked List is a linear data structure, in which the elements are not stored at contiguous memory location.

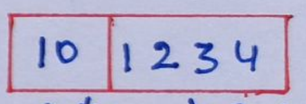
- A Linked List is a dynamic data structure. The No. of nodes in a list is not fixed and can grow and shrink on demand.

- Each Element is called a node which has two parts.
  Info part which stores the information and Pointer which point to the next element.

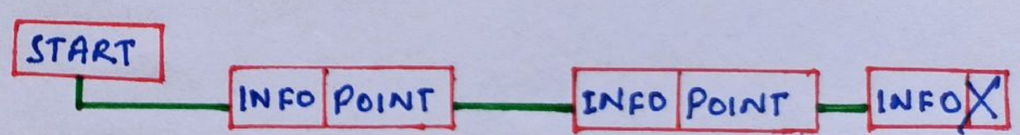


Node

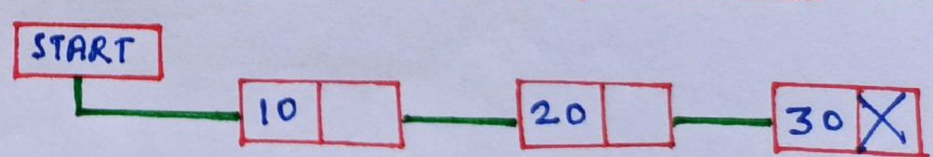
Ex:-



Ex:-



Ex:-



Advantages of Linked Lists

1). **Linked List are dynamic data structure :**
That is, they can grow and shrink during the execution of a program.

2). **Efficient memory utilization :**
Here, memory is not pre-allocated. Memory is allocated whenever it's required. And it's deallocated (Removed) when it's no longer needed.

3). **Insertion and deletions are easier & efficient**
It provide flexibility in inserting a data item at a specified position and deletion of a data item from the given position.

4). Many complex Applications can be easily carried out with linked lists.

Operation on Linked List :

The Basic operation to be performed on the Linked Lists are :-

1). **Creation :** This operation are used to create a Linked List. In this node is created and Linked to the another node.

2). **Insertion :** This operation is used to Insert a new node in the Linked List. A new node may be Inserted.
→ At the beginning of a Linked List.
→ At the End of a Linked List.
→ At the specified position in a Linked List.

3). **Deletion :** This Operation is used to delete an item (a node) from the Linked List. A node may be deleted from.
→ Beginning of a Linked List.
→ End of a Linked List.
→ Specified position in the List.

4). **Traversing :** It is a process of going through all the nodes of a linked list from one end to the other end.

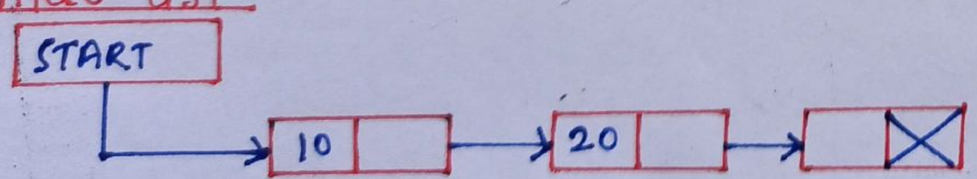
5). **Concatenation :** It's the process of the joining the second list to the end of the first list.

6). **Display :** This operation is used to print each and every nodes information.

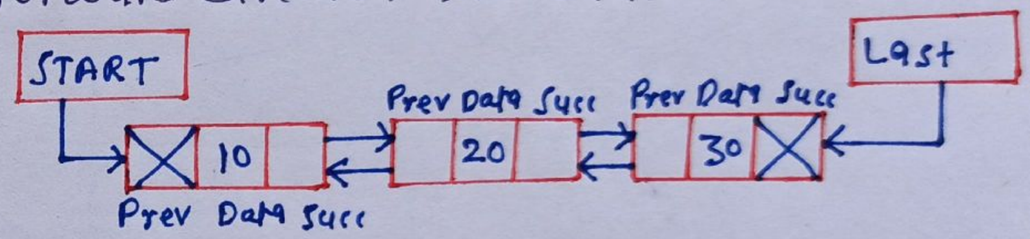
Types of Linked List.

- Basically, there are four type of linked list.

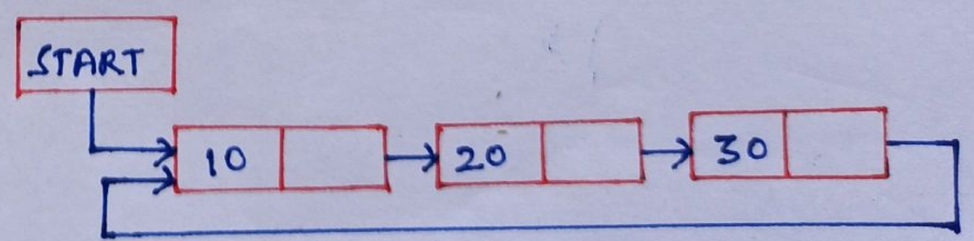
1). **Singly-Linked List :** It's one in which all nodes are linked together in some sequential manner. it is also called Linear Linked list



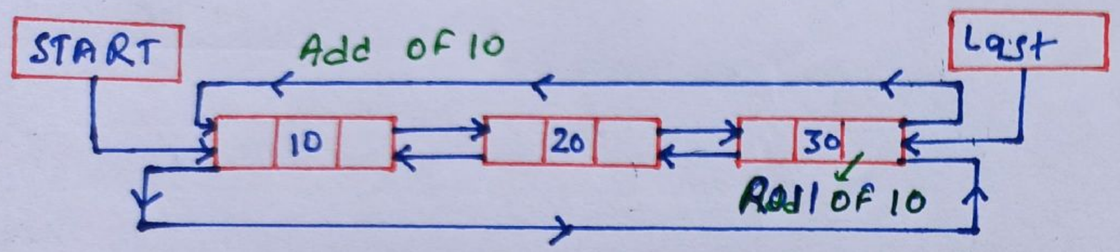
2). **Doubly-Linked List :** It's one in which all nodes are linked together by multiple links which help in accessing both the successor node (Next node) and predecessor node (Previous node) within the list. This help to traverse the list in the forward direction and backward direction.



3). **Circular Linked List :** It's one which has no beginning and no end. A singly linked list can be made a circular linked list by simply sorting the address of the very first node in the link field of the last node.



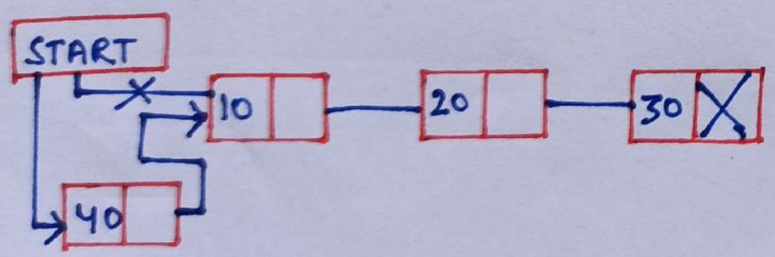
4). **Circular doubly Linked List :** It's one which both the successor pointer and predecessor pointer in a circular manner.



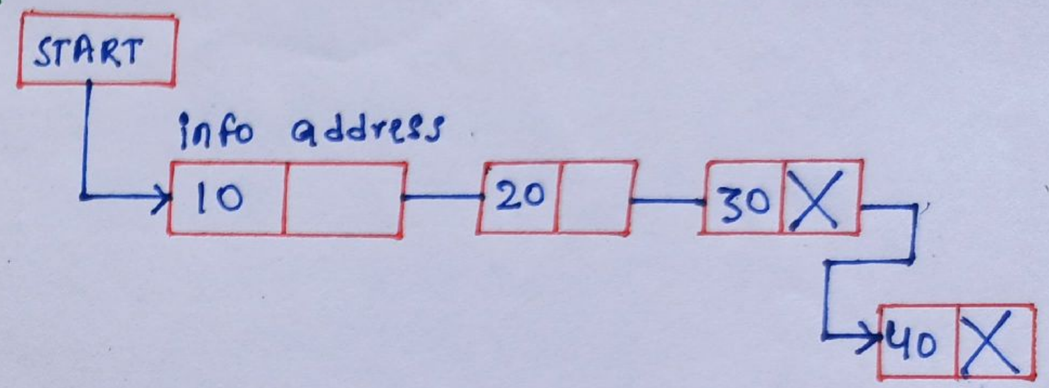
Inserting of Nodes in Linked List

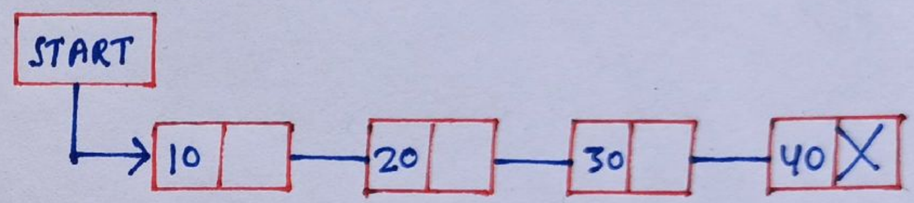
1). Inserting at the beginning of the list.
2). Inserting at the End of the list.
3). Inserting at the specified position within the list.

1).

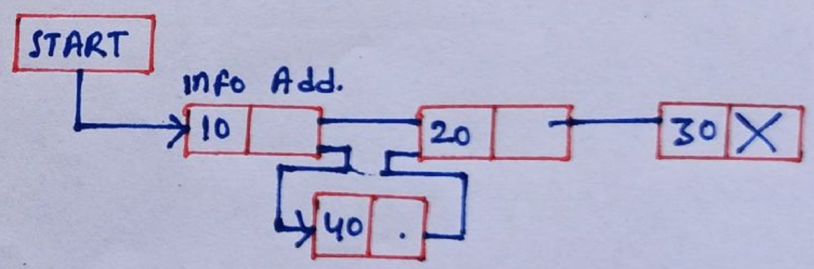


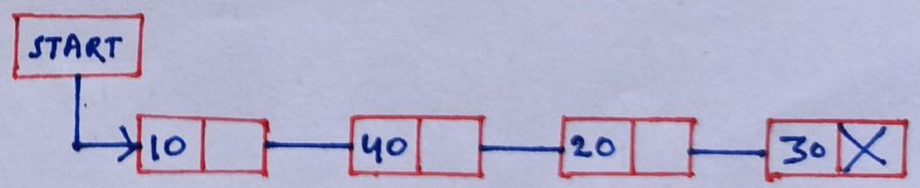
2).





3).



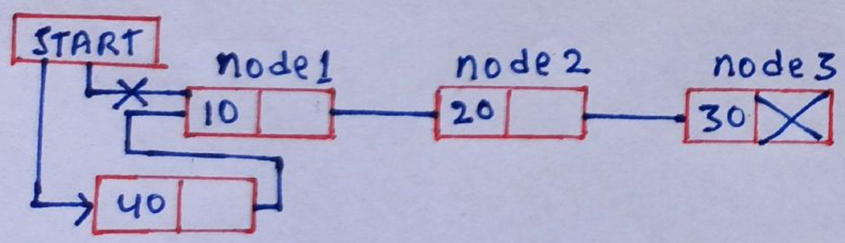


LINKED LIST

Inserting a node at the Beginning in Linked List.

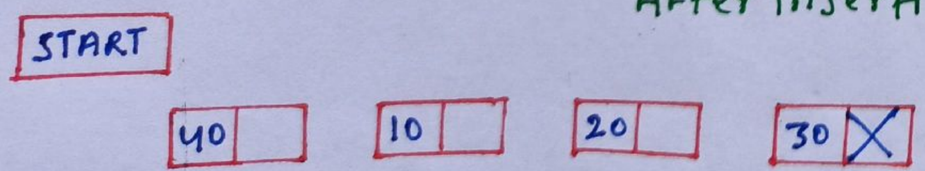
Algorithm ⇒

```
        INSERT_FIRST (START, ITEM)
Step 1: [Check for overflow]
        If Ptr = NULL then
            Print overflow
            Exit
        Else
            PTR = (Node *) malloc (size of (Node))
            // Create new node from memory and
               assign its address to PTR.
Step 2: Set PTR → INFO = Item.
Step 3: Set PTR → Next = START
Step 4: Set START = PTR
```



new node.

After Insertion



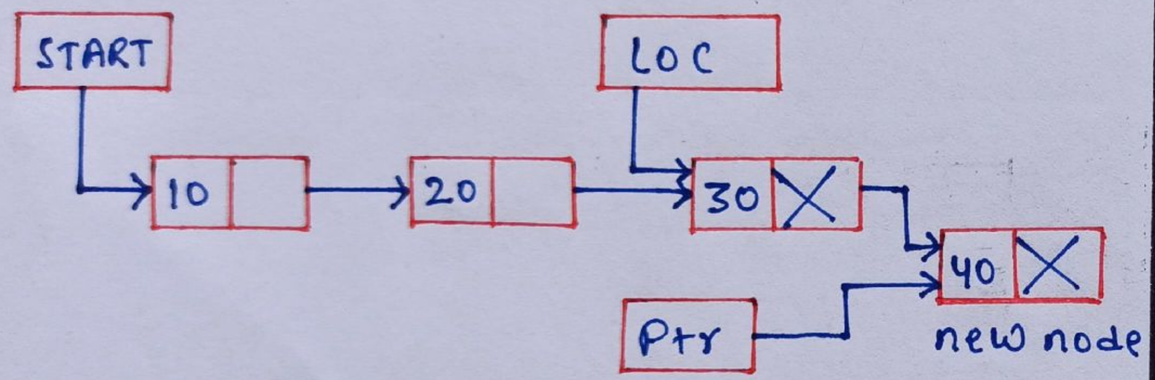
LINKED LIST

Insert a Node at the End in singly linked.

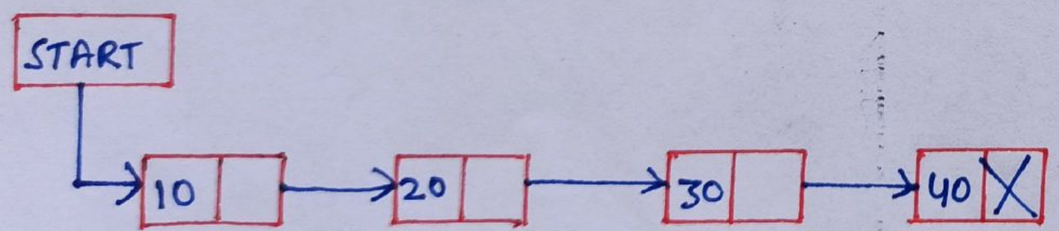
Algorithm ⇒

```
        INSERT_LAST (START, ITEM)
Step 1: Check for overflow
        If Ptr = NULL then
            Print overflow
            Exit
        PTR = (Node *) malloc (size of
                               (Node));
Step 2: Set PTR → Info = Item;
Step 3: Set PTR → Next = NULL;
Step 4: If start = NULL and then
            set START = Ptr;
        Else,
Step 5: set LOC = start
```

Step 6: Repeat step 7 until Loc → Next != NULL
Step 7: Set Loc = Loc → Next ;
Step 8: Set Loc → Next = Ptr;



After Insertion.



LINKED LIST

Inserting a node at the specific position in singly linked list

Algorithm ⇒

Insert_Location (START, Item, LOC)

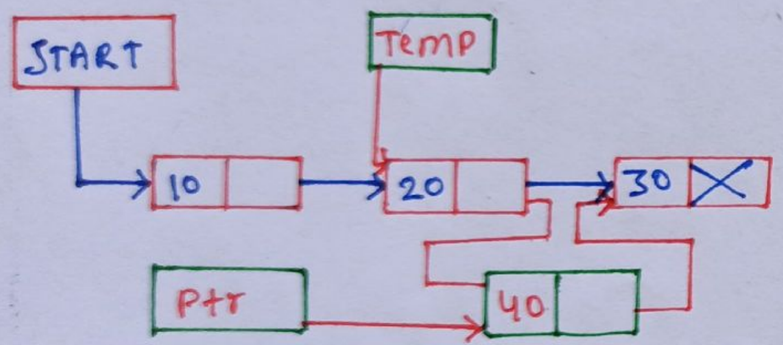
Step 1: Check for overflow
If Ptr == NULL then
Print overflow
Exit
Else
Ptr = (Node*) malloc (size of (Node))

Step 2: Set Ptr → Info = Item

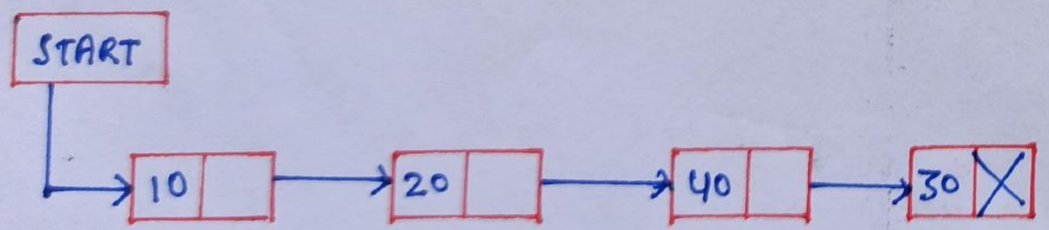
Step 3: If start = NULL then
set start = Ptr
set Ptr → Next = NULL

Step 4: Initialize the counter I and pointers
set I = 0
set temp = start

Step 5:- Repeat steps 6 and 7 until I < Loc
Step 6: Set temp = temp → Next
Step 7: Set I = I + h
Step 8: Set Ptr → Next = temp → Next
Step 9: Set temp → Next = Ptr



After Insertion



Deleting Node in Linked List

Deleting a node from the Linked List has three instances.

1 ⇒ Deleting the first node of the Linked List.

2 ⇒ Deleting the Last node of the Linked List.

3 ⇒ Deleting the node from specified position of the Linked List.

Linked List Deleting Nodes

Deleting the first Node in singly Linked List

Algorithms ⇒

Deleted first (START)

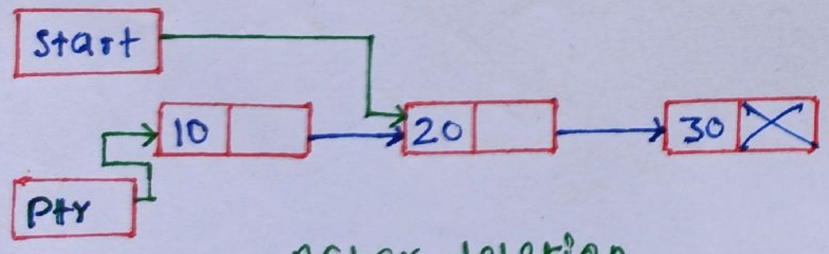
Step 1: Check for underflow.
If start = NULL, then
print Linked list Empty
Exit.

Step 2: Set PTR = START

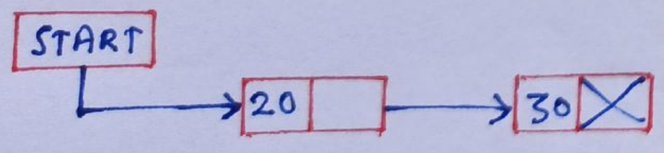
Step 3: Set START = START → Next

Step 4: Print Element deleted is Ptr → info

Step 5: free (Ptr)



After deletion



Linked List Deleting Nodes

Deleting the last node in singly linked List

Algorithm ⇒

Deleting (START)

Step 1: Check for Underflow
If start = NULL then
Print Linked List is Empty
Exit

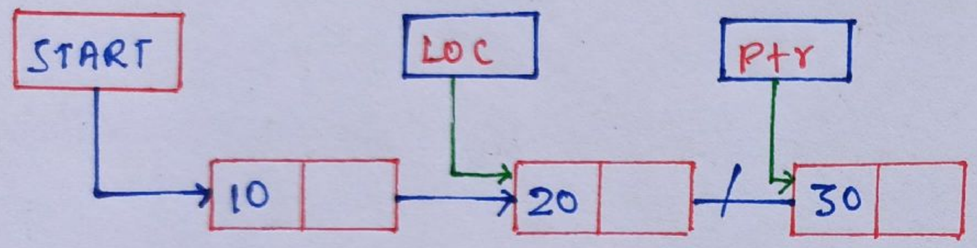
Step 2: If start → Next = NULL then
Set Ptr = start
Set start = NULL
Print element deleted is = PTR → Info
free (PTR)
End if

Step 3: Set PTR = START

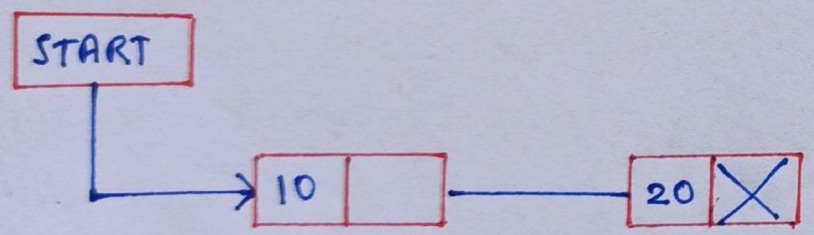
Step 4: Repeat step 5 and 6 until
PTR → Next ! = NULL.

Step 5: Set Loc = PTR

Step 6: Set PTR = PTR → Next
Step 7: Set LOC → Next = NULL
Step 8: Free (PTR).



After deletion



LINKED LIST DELETING NODES

Deleting the Nodes from Specified position in Singly Linked list.

Algorithm ⇒

Delete - Location (START, LOC)

Step 1: Check for underflow
if PTR = NULL then
Print underflow
Exit

Step 2: Initialize the counter I and pointers.
Set I = 0;
Set Ptr = start;

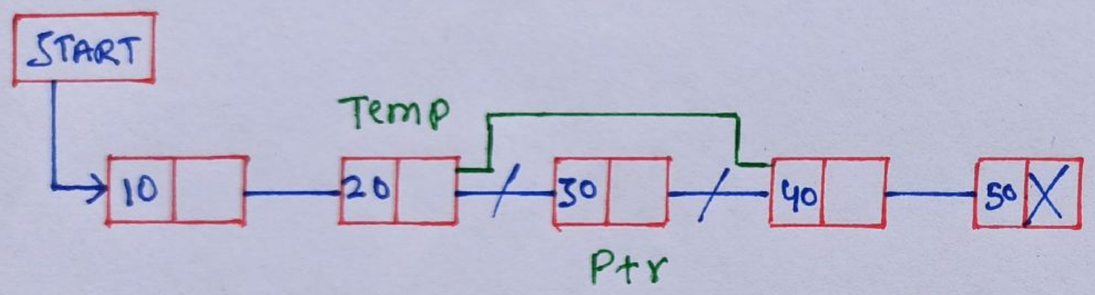
Step 3: Repeat step 4 to 6 until
I < LOC

Step 4: Set temp = PTR

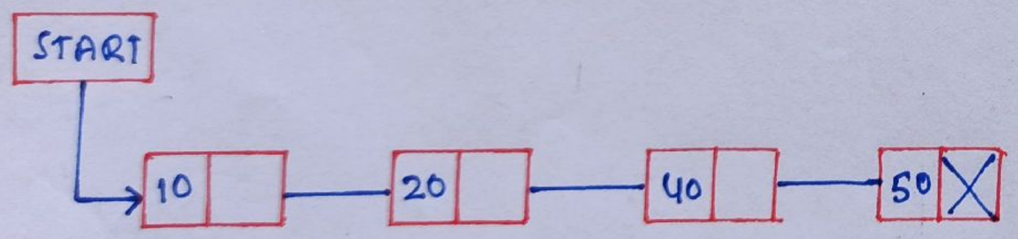
Step 5: SET PTR = PTR → Next

Step 6: Set I = I + 1.

Step 7: Print Element deleted is
= Ptr → Info
Step 8: Set Temp → Next = Ptr → Next
Step 9: free (Ptr)



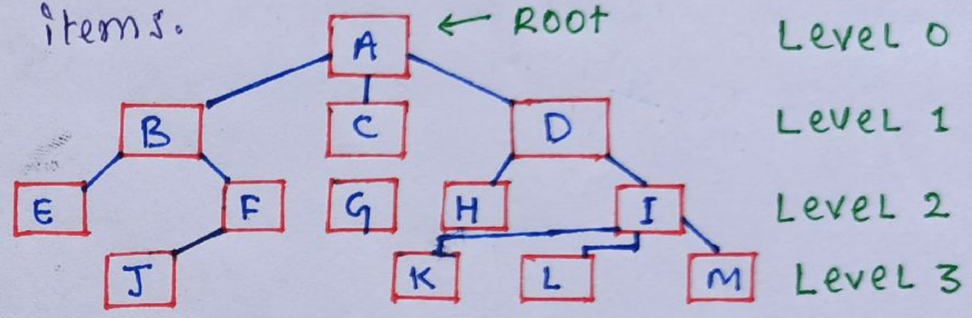
After deletion.



TREES IN DATA STRUCTURE

Tree : A Tree 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.



Tree Terminology : Tree has different terminology such as :-

1). **Root :** It is specially designed data item in a tree. It is the first in the hierarchical Arrangement of data item.

2). **Node :** Each data item in a tree is called a node. In the given Tree there are 13 nodes such as :- A, B, C, D, E, F, G, H, I, J, K, L, M.

3). **Degree of a node :** It is the no. of subtrees of a node in a given tree.

The degree of A = 3
The degree of C = 1
The degree of L = 0

4). **Degree of a tree :** It is the maximum degree of nodes in a given tree. In the given tree the node A and node I has maximum degree(3). so the degree of tree is 3.

5). **Terminal node :** A node with degree zero is called terminal node. In given tree - E, J, G, H, K, L and M are terminal node.

6). **Non-Terminal Node :** Any node whose degree is not zero is called non-terminal node. In given tree - A, B, C, D, F, I are non-terminal node.

7). **Siblings :** The child nodes of a given parent node are called Siblings. They are also called brothers.
In the given table.

- B, C, D are siblings of parent node A.
- H & I are siblings of parent node D.

8). **Level :** The entire tree structure is Levelled in such a way that the root node is always at level 0.

9). **Edge :** It is a connecting line of two nodes. that is, the line drawn from one node to another node is called an Edge.

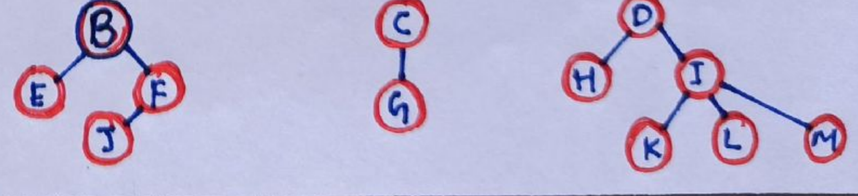
10). **Path :** It is a sequence of consecutive edges from the sourc node to the destination node. In the given tree the path b/w A and J is as.

(A, B) (B, F) and (F, J)
A → B → F → J

11). **Depth :** It is the maximum level of any node in a given tree. In the given tree, the root node A has the maximum level.

12). **Forest :** It is a set of disjoint trees. In a given tree if you remove its root node then it becomes a forest. In the given tree, there is forest with three tree. such as.

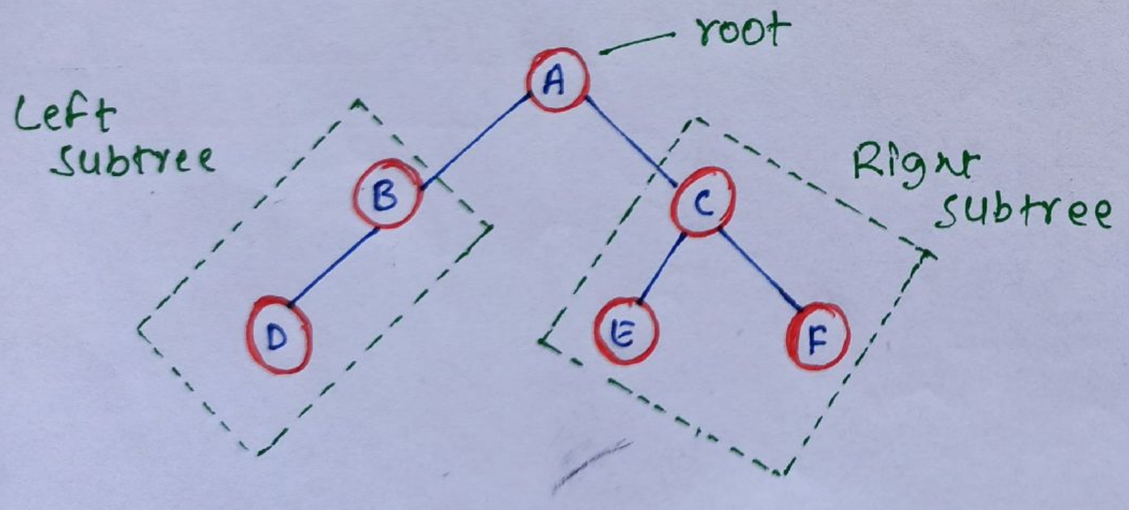
After removing root A. forest is



BINARY TREES

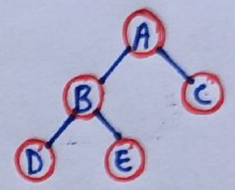
- Binary tree is a finite set of data item which is either empty of consists of a single item called root and two disjoint binary tree called the Left subtree and right subtree.

- In Binary tree, Every node can have maximum of 2 childern which are known as Left child and Right child.

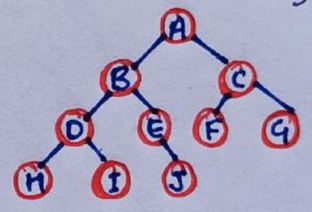


TYPES OF BINARY TREES

1). **Full Binary tree:** A Binary tree is full if every node has 0 or 2 child.



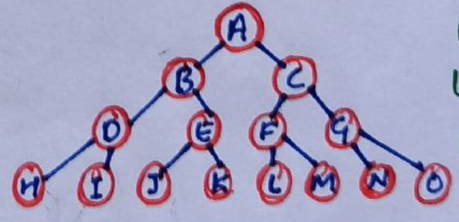
2). **Complete Binary tree:** A Binary tree is complete binary tree if all levels are completely filled except possibly the last level and the last level has all keys as left as possible.



Level 0 $2^0 = 1$
Level 1 $2^1 = 2$
Level 2 $2^2 = 4$
Level 3 $2^3 = 8$

3). **Perfect Binary Tree:** A tree in which all internal nodes has two childern and all leaves are at same level.

In which all level has 2^n child.



Level 0 → $2^0 = 1$
Level 1 → $2^1 = 2$
Level 2 → $2^2 = 4$
Level 3 → $2^3 = 8$

2^n → level

Traversal of a Binary Tree

It is a way in which each node in the tree is visited exactly once in a systematic manner.
There are three ways which we use to traverse a tree –

	Node ↑	Left ↑	Right ↑
1 - Pre Order traversal	(N	L	R)
2 - In order traversal	(L	N	R)
3 - Post order traversal	(L	R	N)

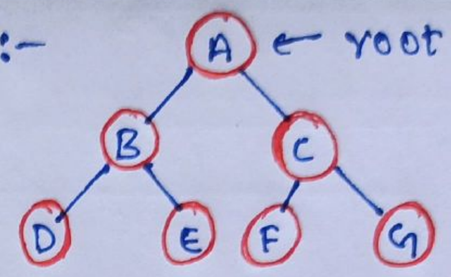
1- **Pre order Traversal :** In this Traversal method, the root (N) node is visited first, then the Left (L) subtree and finally the right (R) subtree.

Algorithm ⇒

Until all nodes are traversed –

Step 1: Visit root node.
Step 2: Recursively traverse Left subtree.
Step 3: Recursively traverse Right subtree.

Ex :-



Pre – Order traversal is →

A, B, E, D, C, F, G.

2). **Inorder Traversal :** In this traversal method, the Left (L) subtree is visited first, then the root (N) and later the right (R) subtree.

Algorithm ⇒

Until all nodes are traversed –

Step 1 : Recursively traverse left subtree.
Step 2: Visite root node.
Step 3 : Recursively traverse Right subtree.

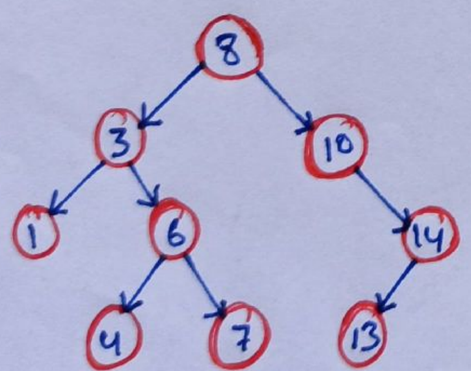
Binary Search tree (BST)

- Binary search tree is a node-based binary tree data structure which has the following Rules :-

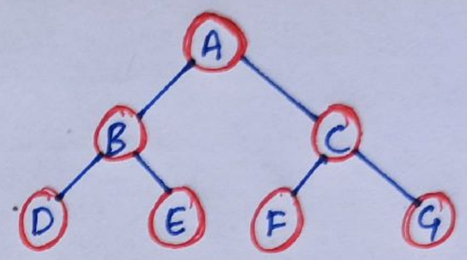
1). The value of the key in the left child or left subtree is less than the value of root.

2). The value of the key in the right child or right subtree is more than or equal to the root.

3). The right and left subtree each must also be a binary search tree (BST).



Ex :-



Inorder Traversal is —

D, B, E, A, F, C, G.

3). **Post-Order Traversal :** In this method the root node is visited last, hence the name first we traverse Left (L) subtree, then the right (R) subtree and finally the root (N) node.

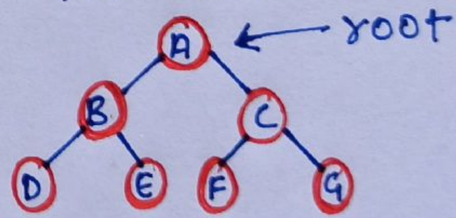
Algorithm ⇒

Step 1: Recursively traverse Left subtree.

Step 2: Recursively traverse right subtree.

Step 3: visit root node.

Ex :-



post order Traversal is —

D, E, B, F, G, C, A

DIFFERENCE BETWEEN STACK and QUEUE

STACK

1). It represents the collection of elements in Last in fist out (LIFO) order.
2). Objects are inserted and removed at the same end called Top of stack (Tos).
3). Insert operation is called push operation.
4). Delete operation is called POP operation.
5). In stack there is no wastage of memory space.
6). Plate counter at marriage Reception is an Example of stack.

QUEUE

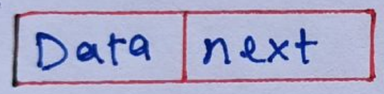
1). It represents the collection of elements in fist in fist out (FIFO) order.
2). Object are inserted and removed from different ends called front and rear ends.
3). Insert Operation is called Enqueue Operation.
4). Delete Operation is called Dequeue operation.
5). In Queue there is a wastage of memory space.
6). Students standing in a line at fees counter is an example of Queue.

DIFFERENCE BETWEEN SINGLY & DOUBLY LINKED LIST

SINGLY LINKED LIST

1). Singly Linked List has nodes with data field and next link field (forward link).

Ex :-



2). It allows traversal only in one way.

3). It requires one list pointer variable (start).

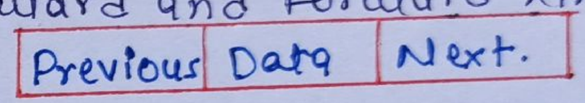
4). It Occupies less memory.

5). Complexity of Insertion and Deletion at known position is O(n)

DOUBLY LINKED LIST

1). Doubly Linked List has nodes with data field and two pointer field. (Backward and forward link).

Ex :-



2). It allows a two way traversal.

3). It requires two list pointer variable (start and last).

4). It occupies more memory.

5). Complexity of Insertion and Deletion at known position is O(1).

DIFFERENCE BETWEEN LINEAR & NON-LINEAR DATA STRUCTURE

LINEAR DATA STRUCTURE	NON-LINEAR DATA STRUCTURE
1). In this data structure, The elements are organized in a sequence such as :- Ex :- Array, stack, queue etc..	1). In this data structure data is Organized without any sequence. Ex :- Tree, Graph etc.
2). In Linear data structure single level is involved.	2). In Non-Linear Data structure multiple levels are involved.
3). It is easy to implement.	3). It is difficult to implement.
4). Data elements can be traversed in a Single Run only.	4). Data elements can't be traversed in a single Run only.
5). Memory is not utilized in a efficient way.	5). Memory utilization in an efficient way.
6). Application of Linear D.S. are mainly in Application softwere development.	6). Application of non-Linear D.S. are in Artificial Intelligence and Image Processing.

DIFFERENCE BETWEEN ARRAY AND LINKED LIST

ARRAY

1). Size of an Array is fixed.

2). Array is a collection of Homogeneous (similar) data type.

3). Memory is allocated from stack

4). Array work with static data structure.

5). Elements are stored in contiguous memory location.

6). Array elements are independent to each other.

7). Array take more time.
(Insertion & Deletion)

LINKED LIST

1). Size of a List is not fixed.

2). Linked List is a collection of node (data & address)

3). Memory is allocated from heap.

4). Linked List work with dynamic data structure.

5). Elements can be stored anywhere in the memory.

6). Linked list elements are depend to each other.

7). Linked - List take less time.
(Insertion & Deletion)

DIFFERENCE BETWEEN TREE AND GRAPH

TREE

1). Tree is a collection of nodes and Edges.
EX :- T = {node, Edges}

2). There is a unique node called root in tree.

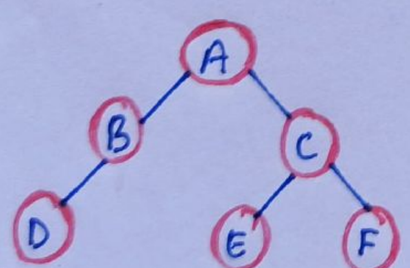
3). There will not be any cycle/loops.

4). Represents data in the form of a tree structure, in a hierarchical manner.

5). In tree only one path between two nodes.

6). In this preorder, In order and preorder Traversal.

Ex :-



GRAPH

1). Graph is a collection of vertices/nodes and Edges.
EX :- G = {V, E}

2). There is no unique node.

3). There can be loops/cycle.

4). Represents data similar to a network.

5). In Graph One or more then one path between two nodes.

6). In this BFS and DFS traversal.

Ex :-