Data Structure

Data ⇒ Anything to give information is called data.

Ex ⇒ Student Name, Student Roll no.

Structure ⇒ Representation of data is called structure.

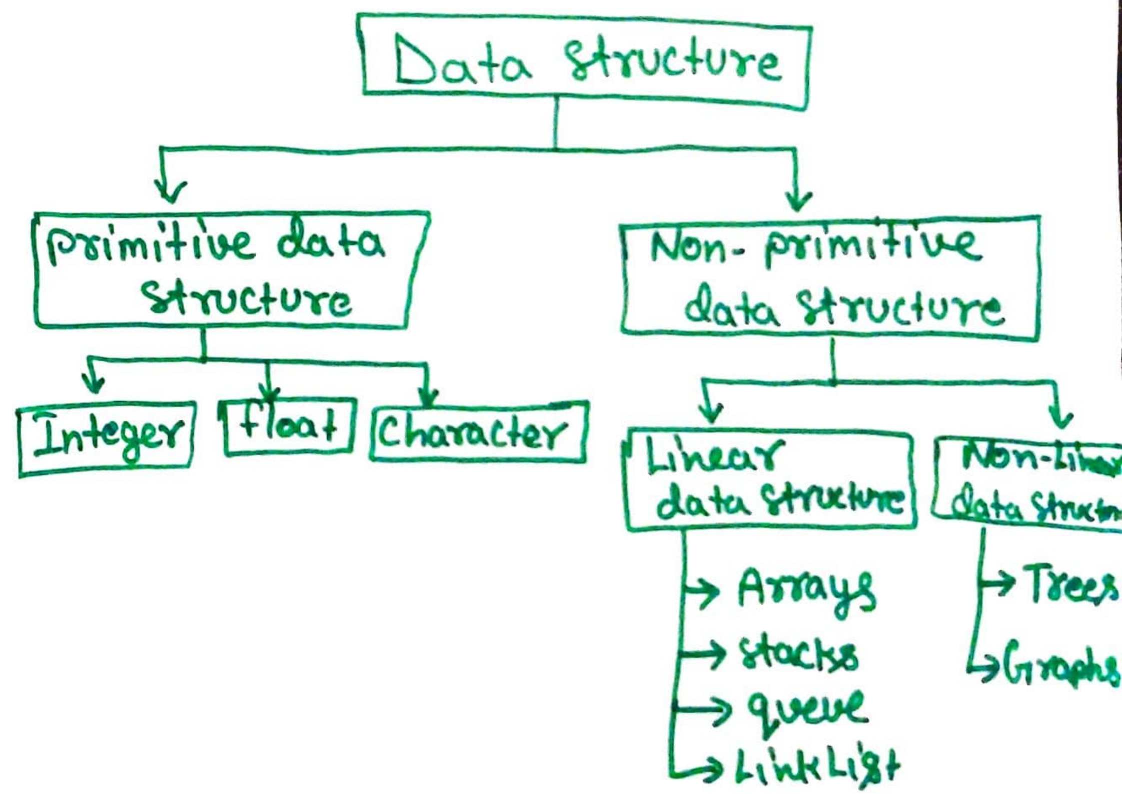
Ex ⇒ graph, Arrays, List.

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. it's 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

i) Insertion ⇒ Add a new data in the data structure

ii) Deleting ⇒ Remove a data from the data structure

iii) Sorting ⇒ Arrange data in increasing or decreasing order.

iv) Searching ⇒ find the location of data in data structure.

v) merging ⇒ Combining the data of two different sorted files into a single sorted file.

vi) Traversing ⇒ Accessing each data exactly one in the data structure so that each data item 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 Store multiple values which can be referenced by a single name.

Types of Arrays

- Single Dimensional Arrays
- Multi Dimensional Arrays

1) Single Dimensional Arrays ⇒ • It's also known as One Dimensional (1D) Array.

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

[row] [col]

Declaration ⇒

data-type var_name [Expression];
(Expression → size)

Ex ⇒

```
int num[10];
char c[5];
```

Initializing one-Dimensional 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-Dimensional Arrays ⇒ multidimensional Arrays use more then one Subscript to describe the Arrays Elements. [ ][ ][ ] ---

Two Dimensional Arrays ⇒ It's use two Subscript, one Subscript [row] [colum] to represent row value and second Subscript to represent Column value. It mainly use for matrix Representation.

Declaration two-Dimensional Arrays ⇒

data-type var-name [rows] [colums];

Ex ⇒

```
int num[3][2];
```

Initialization 2-D Arrays ⇒

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

Ex ⇒

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

or

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

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}_{3\times2}$$

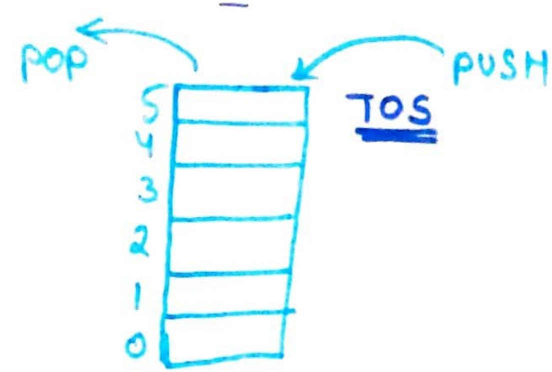
num[0,0] = 1
nu[01] = 2
num[10] = 3
num[11] = 4
num[20] = 5
nu[21] = 6

Write a program to read & write one Dimensional Array.

```
# include <stdio.h>  — standard input output (printf, scanf)
# include <conio.h>  → console input output (clrscr(), getch())
void main()
{
    int a[10], i;
    clrscr();
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 full 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 is 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 [maxSize], 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

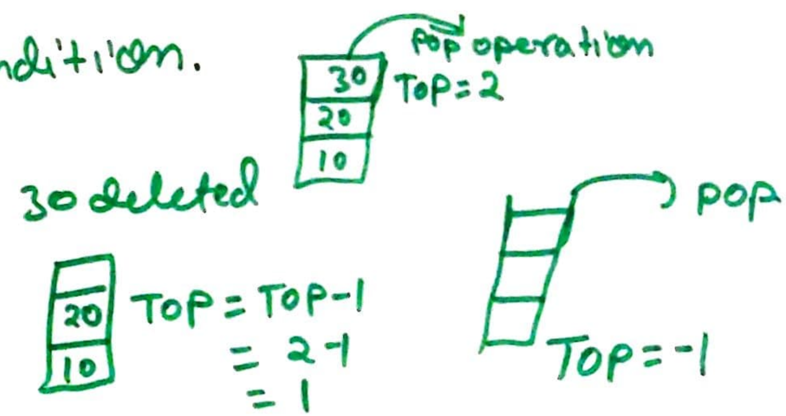
Step 6: print "Stack overflow"

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 an item from the Stack (POP)

pop (Stack [maxSize], item)

Step1: Repeat Steps 2 to 4 until Top ≥ 0

Step2: Set item = Stack [TOP]

Step3: Set top = top - 1

Step4: print, No. deleted is, Item

Step5: Print stack under flows.

Stacks (prefix & postfix)

Stack Notation ⇒ There are three stack Notation.

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) Posfix 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

$R_1 * C/D + E \wedge F/G$

$R_1 * C/D + \wedge EF/G$

Let ⇒ $\wedge EF = R_2$

$R_1 * C/D + R_2/G$

$R_1 * /CD + R_2/G$

Let ⇒ $/CD = R_3$

$R_1 * R_3 + R_2/G$

$R_1 * R_3 + /R_2G$

Let ⇒ $/R_2G = R_4$

$R_1 * R_3 + R_4$

$* R_1R_3 + R_4$

Let $* R_1R_3 = R_5$

$R_5 + R_4$

$+ R_5R_4$

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

$+ * R_1R_3 / R_2G$

$+ * + AB / CD / \wedge EFG$ ✓

postfix ⇒ $(A+B) * C/D + E \wedge F/G$

$AB+ * C/D + E \wedge F/G$

Let $AB+ = R_1$

$R_1 * C/D + E \wedge F/G$

$R_1 * C/D + EF\wedge /G$

Let $EF\wedge = R_2$

$R_1 * C/D + R_2/G$

$R_1 * CD/ + R_2/G$

Let $CD/ = R_3$

$R_1 * R_3 + R_2/G$

$R_1 * R_3 + R_2G/$

Let $R_2G/ = R_4$

$R_1 * R_3 + R_4$

$R_1R_3* + R_4$

Let $R_1R_3* = R_5$

$R_5 + R_4$

R_5R_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 perform

1) Reverse the input string
2) perform tabular method and find postfix expression.
3) Reverse this postfix Expression string to find the prefix.

Priority
- ^ → highest
- *, / → 2 highest
- + - → Lowest priority

Ex ⇒ A + B*C

first to Add bracktes

(A+ B*C)

Reverse string

(C*B+A)

Tabular form.

Symbol Scanned	stack	postfix Expression
(	(	—
C	(	C
*	(*	C
B	(*	CB
+	(+	CB*
A	(+	CB*A
)	—	CB*A+

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

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

Symbol Scanned	Stack	postfix Expression
	(	—
(	(	A
A	(+	A
+	(+	AB
B	(+*	AB
*	(+*	ABC
C		ABC*+
)		

postfix Expression = **ABC*+**

Queues

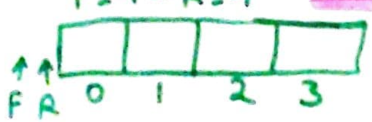
- Queue is a Non-primitive Linear data structure.
- It is an homogeneous Collection of Elements in which new Elements are added at one End called the Rear End, and the Existing Element are deleted from other End called the front End.
- The first added Element will be the first to be remove from the queue. 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

$$F = F+1$$

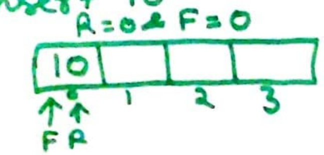
Ex ⇒ $F=-1$ & $R=-1$



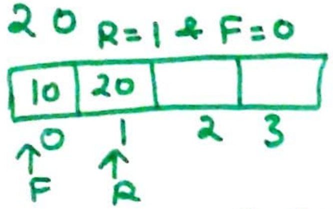
Empty queue

22

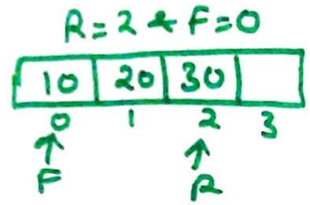
insert 10



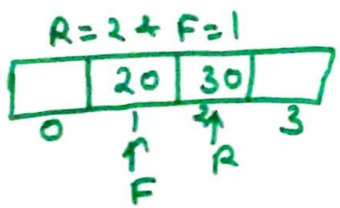
insert 20



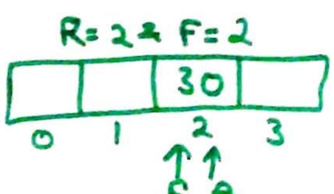
insert 30



deleted Element. first delete 10



deleted second element.



23

Operation on Queue

1) To insert an Element in a Queue ⇒

Algo ⇒

QINSERT [QUEUE[maxSize], ITEM]

Step1 : 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

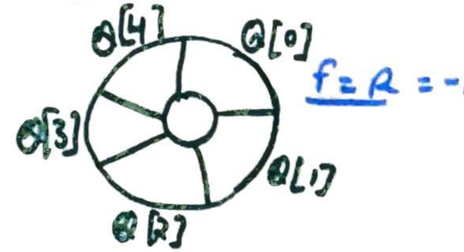
Else
    front = front + 1

Step 4: print, No. Deleted is, item

Step 5: Print "Queue is Empty or underflow".

CIRCULAR QUEUE

A Circular 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.



A Circular queue overcome the problem of unutilized space in linear queues implemented as arrays.

Circular 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 overflow & 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]

Step 2 ⇒ Exit

Queue (Data Structure)

Operation on Queue

10, 20, 30, 40

Ex ⇒

maxsize = 3

1) front = -1 ✓ Empty queue
Rear = -1

2) 3 to 5 Step Repeat

R < maxsize - 1

-1 < 3-1

-1 < 2 true

3 4 5

3) Read item

Read 10

4) f == -1

-1 == -1 true

f = 0

R = 0

5) Set q[0] = item

q[0] = 10

q[0]	q[1]	q[2]
10		

queue

10		
q[0]	q[1]	q[2]

f=0 R=0

Rear < maxSize-1

0 < 3-1

0 < 2 true

Read 20

4) if f==-1

0==-1 false

Else

R=R+1

R=0+1

R=1

5) q[1] = 20

10	20	
q[0]	q[1]	q[2]

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

5) set q[2] = 30

10	20	30
q[0]	q[1]	q[2]

f=0 R=2

Case Rear < maxSize-1

2 < 3-1

2 < 2 false

6) <u>queue is overflow</u>

DELETE an Element in Circular 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

q[0]	q[1]	q[2]
10	20	30

F = 0 R = 2

Case ↳

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

q[0]	q[1]	q[2]
	20	30

F = 1 R = 2

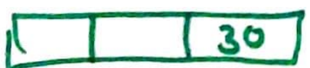
F = 1 R = 2

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

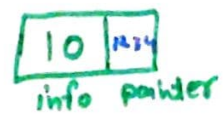
Step 5: queue is Empty
queue is <u>Underflow</u>.

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 <u>node</u>, 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 Lists 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 notonger needed.

3) **Insertion and deletions are Easier and 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's a process of going through all the nodes of a Linked List from one End to the other End.

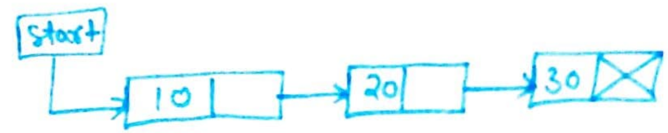
5) **Concatenation** ÷ It's the process of joining the second List to the End of the first list.

6) **Display** ÷ This operation is used to print Each and Every node's information.

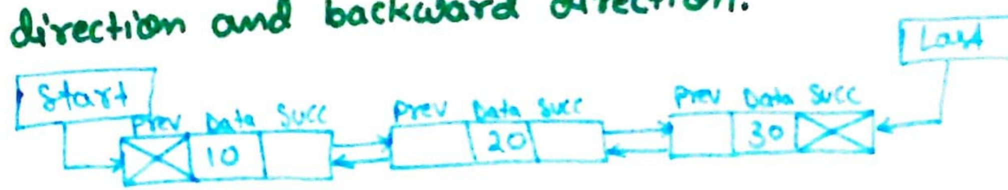
Types of Linked List

- Basically, there are four types of Linked List.

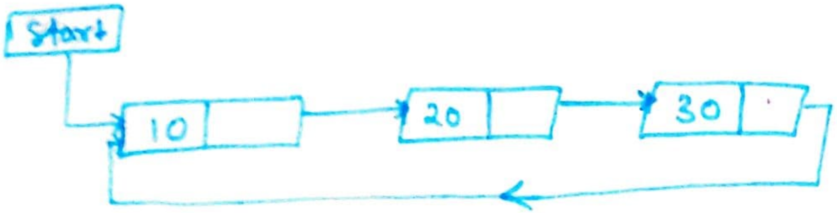
1 ⇒ **Singly-Linked List** ⇒ It's one in which all nodes are Linked together in some sequential manner. It's also called Linear Linked List.



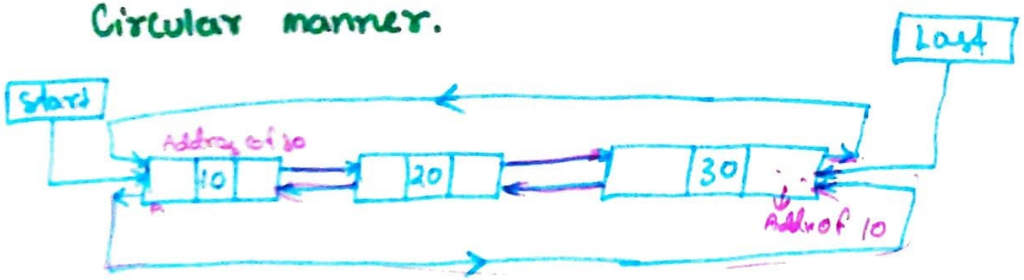
2 ⇒ **doubly-Linked List** ⇒ it's one in which all nodes are linked together by multiple Links which help in Accessing both the successor node (Next node) and predecessor node (previous node) within the List. This helps to traverse the List in the forward direction and backward direction.



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 has both the Successor pointer and predecessor pointer in a Circular manner.



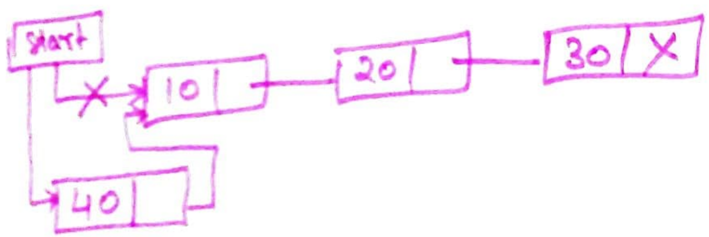
Inserting Nodes in Linked List

1) Inserting at the beginning of the list.

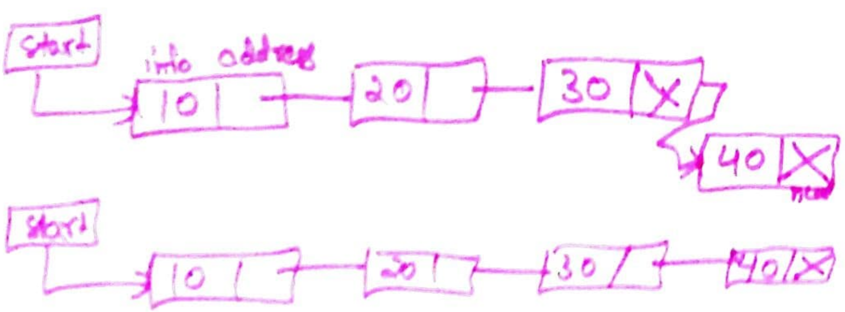
2) Inserting at the End of the list

3) Inserting at the Specified position within the list.

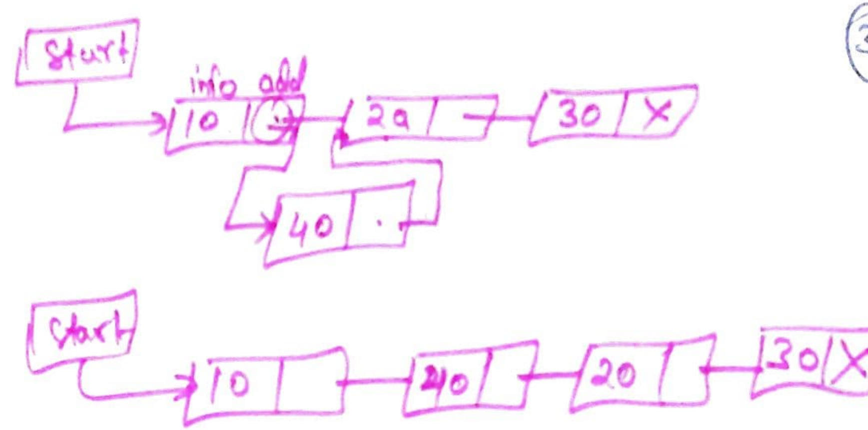
1)



2)



3)



LINKED LIST

Inserting A Node AT the Beginning in Linked List (40)

Algorithm ⇒

INSERT_FIRST(START, ITEM)

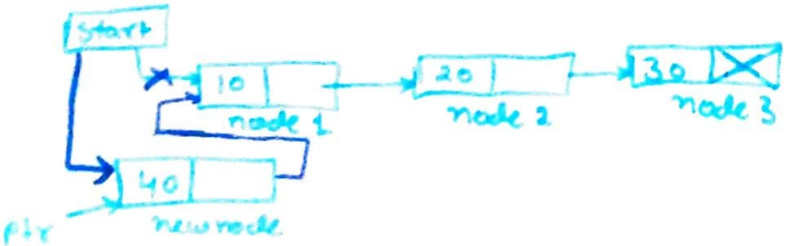
Step 1: [Check for overflow]

    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.

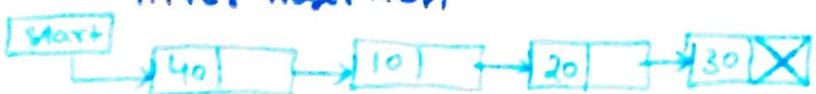
Step 2: Set PTR → INFO = Item

Step 3: Set PTR → Next = START

Step 4: Set START = PTR



After insertion



LINKED LIST

Insert A Node AT The End in Singly Linked List (41)

Algorithm ⇒

Insert_Last(START, ITEM)

Step 1: Check for overflow

    IF ptr = NULL then
        print overflow
        Exit
    Else
        PTR = (Node*) malloc (sizeof(Node));

Step 2: Set PTR → Info = Item ;

Step 3: Set PTR → Next = NULL ;

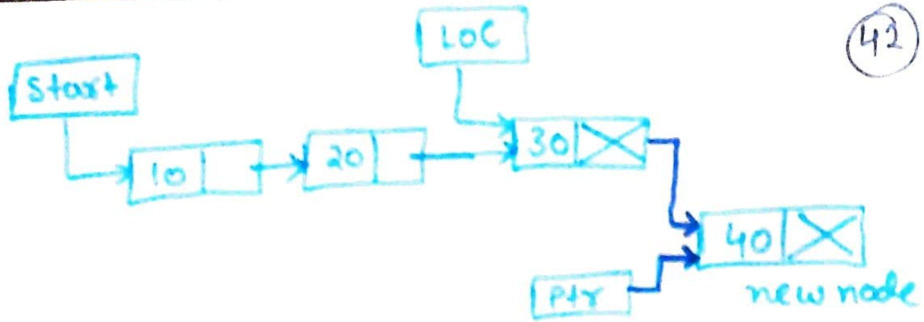
Step 4: If start = NULL and then
    Set START = ptr;
    Else,

Step 5: Set LOC = Start ;

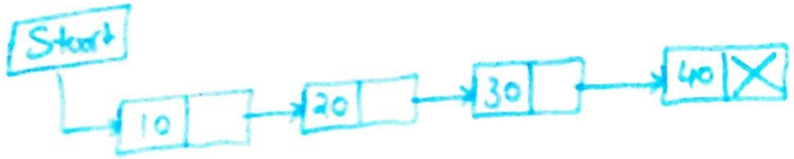
Step 6: Repeat Step 7 until LOC → Next != NULL

Step 7: Set LOC = LOC → Next ;

Step 8: Set LOC → Next = ptr ;



After Insertion



LINKED LIST

Inserting a Node at the Specified Position in Singly Linked List.

Algorithm ⇒

Insert_Location (START, Item, LOC)

Step1: Check for overflow
  If ptr == NULL then
    print overflow
  Exit
Else
ptr = (Node *) malloc (size of(Node))

Step2: set ptr → INfo = item

Step3: IF start = NULL then
    Set Start = ptr
    Set ptr → Next = NULL

Step4: Initialize the Counter I and pointers
  Set I = 0
  Set temp = Start

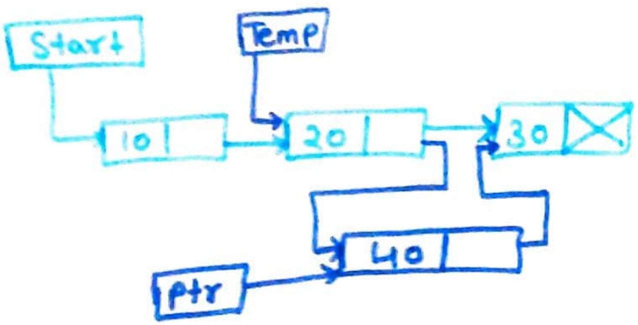
Step5: Repeat Steps 6 and 7 until $I < LOC$

Step6: Set temp = temp → Next

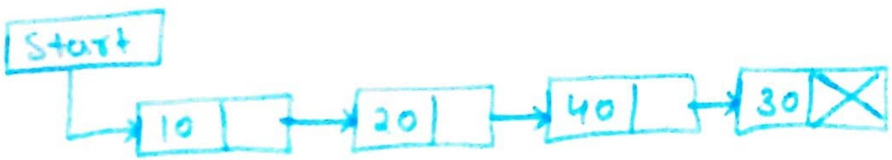
Step7: Set $I = I + 1$

Step8: Set ptr → Next = temp → Next

Step9: 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.

Deleting the first Node in Singly Linked List

Algorithms →

Deleted first (START)

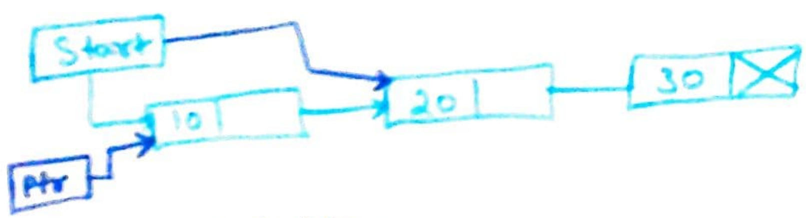
Step1: Check for under flow
If Start = NULL, then
print Linked List Empty
Exit

Step2: Set PTR = START

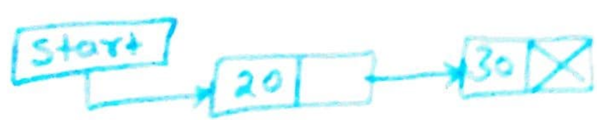
Step3: Set START = START → Next

Step4: print Element deleted is ptr → info

Step5: free (ptr).



After deletion



Deleting the Last node in Singly Linked List

Algorithm →

Deleting (START)

Step1: Check for underflow
If Start = NULL then
print Link List is Empty
Exit

Step2: if Start → Next = NULL then
Set Ptr = Start
Set Start = NULL
print element deleted is = PTR → Info
free (PTR)
End if

Step3: Set PTR = START

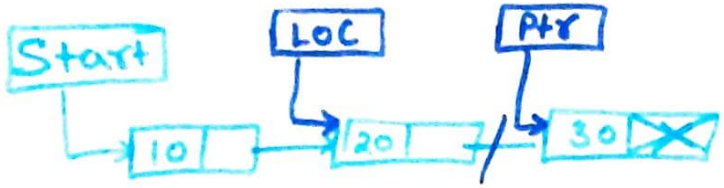
Step4: Repeat Step 5 and 6 untill
PTR → Next ! = NULL

Step5: Set LOC = PTR

Step6: Set PTR = PTR → Next

Step7: Set LOC → Next = NULL

Step8: free (PTR)



After deletion

Deleting the Node from Specified Position In Singly Linked List

Algorithm →

Delete-Location (START, LOC)

Step1: Check for underflow
if PTR = NULL then
print underflow
Exit

Step2: Initialize the Counter I and pointers
Set I = 0;
Set ptr = Start;

Step3: Repeat step 4 to 6 until I < LOC

Step4: Set temp = PTR

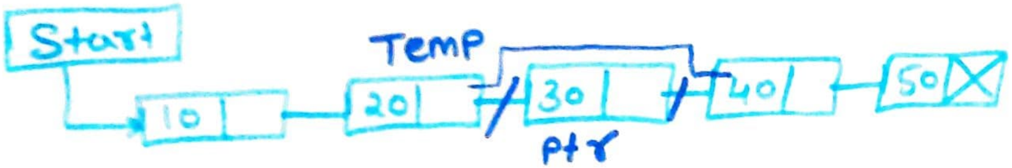
Step5: Set PTR = PTR → Next

Step6: Set I = I+1

Step 7 : print Element deleted is = ptr → info

Step 8 : set Temp → Next = ptr → Next

Step 9 : free (ptr)

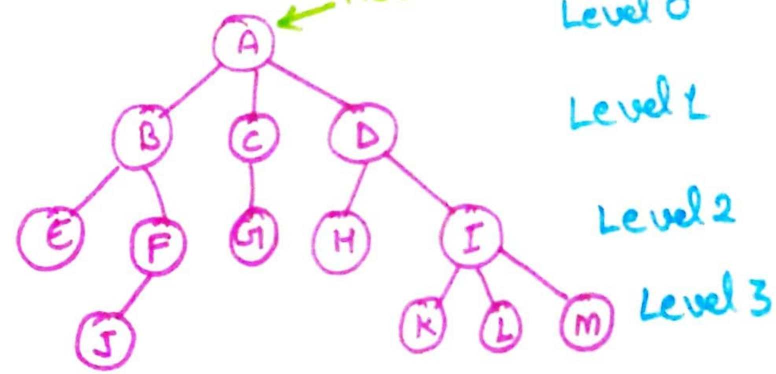


After deletion

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. In the above tree, A is root item.

2 › **Node** ⇒ Each data item in a tree is called a node. In the given

Tree there are 13 Node 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 ⇒ **None-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 nod 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 source node to the destination node. In the given tree the path between A and J is as.

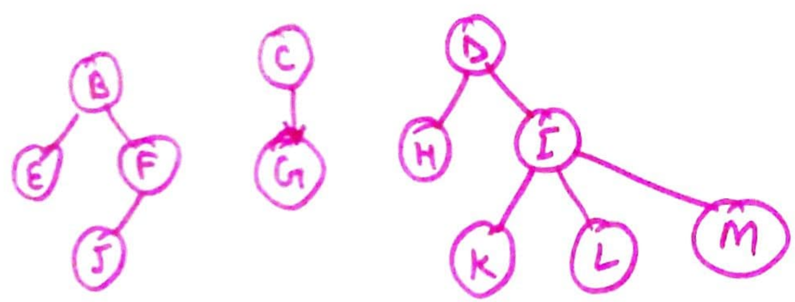
(A, B) (B, F) and (F, J)

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 a forest with three tree. such as.

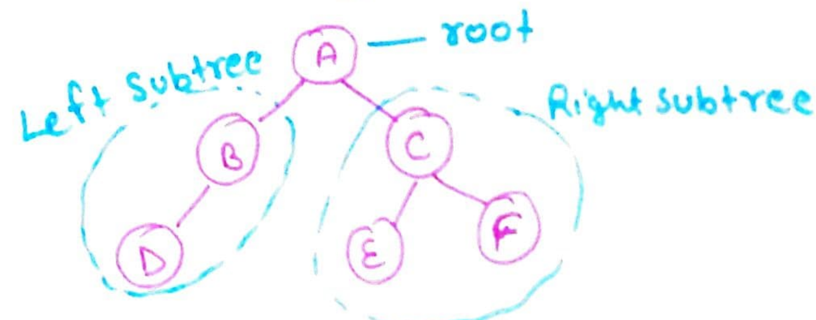
After removing root A. forest is.



BINARY TREES

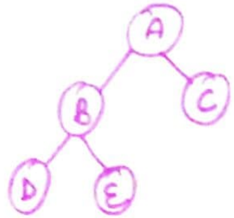
- Binary tree is a finite set of data item which is Either Empty of consists of a single item called root and two disjoint binary tree called the Left subtree and right subtree

- In Binary tree, Every node can have maximum of 2 Children which are Known as Left Child and Right child

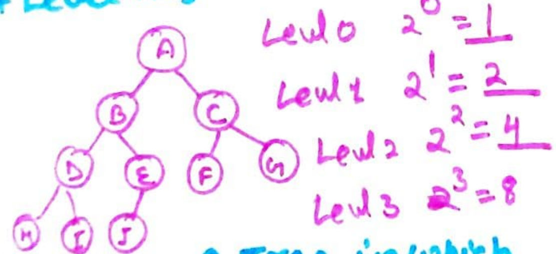


Types of Binary trees ⇒

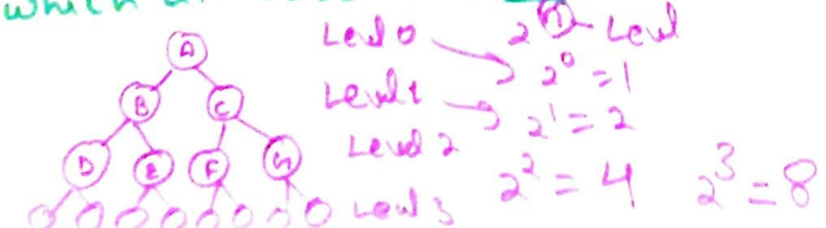
1) full Binary tree ⇒ A Binary Tree is full if Every node has 0 or 2 child.



2) Complete Binary tree ⇒ A Binary tree is Complete Binary Tree if all Levels are completely filled Except possibley the Last Level and the Last Level has all keys as Left as possible.



3) Perfect Binary Tree ⇒ A Tree in which all internal nodes has two children and all Leaves are at the Same Level. in which all Level has 2^n child



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- Preorder traversal (N L R)
2- Inorder traversal (L N R)
3- Postorder traversal (L R N)

1→ Preorder Traversal ⇒ In this Traversal method, the root node (N) is visited first, then the Left Subtree (L) and finally the right Subtree (R).

Algorithm ⇒

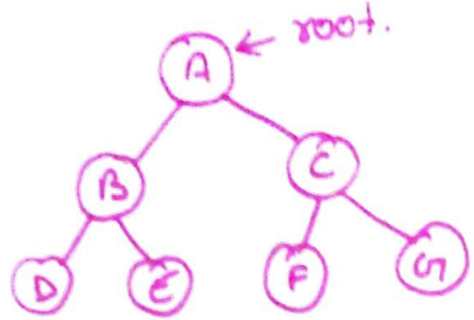
Until all nodes are traversed -

Step1: Visit root node.

Step2: Recursively traverse Left Subtree.

Step3: Recursively traverse Right Subtree.

Ex ⇒



Pre-order traversal is→

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

2→ Inorder Traversal ⇒ In this traversal method, the Left Subtree (L) is visited first, then the root (N) and later the right (R) Sub-tree.

Algorithm ⇒

untill all nodes are traversed –

Step1: Recursively traverse Left Subtree.

Step2: Visit root node.

Step3: Recursively traverse Right Subtree.

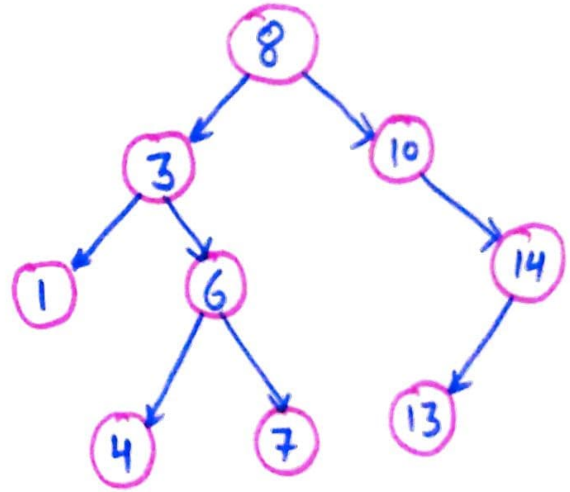
Binary Search tree (BST)

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

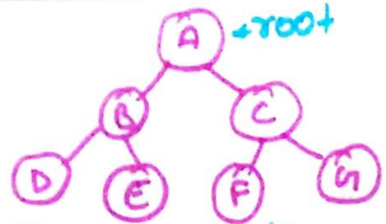
1→ The value of the key in the Left child or left Subtree is less than the value of root.

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 subtree (L), then the right subtree (R) and finally the root node (N).

Algorithm ⇒

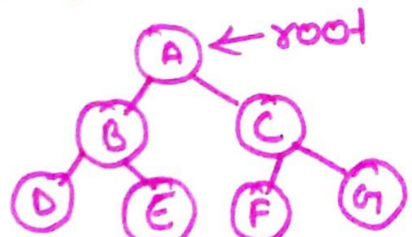
Until All nodes are traversed -

Step1: Recursively traverse Left subtree.

Step2: Recursively traverse right subtree.

Step3: Visit root node.

Ex ⇒



Post order Traversal is -

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

Difference between Stack and Queue

Stack	Queue
1→ It represents the Collection of Elements in Last in first Out (LIFO) order.	1→ It represents the Collection of Elements in First In First Out (FIFO) order.
2→ Objects are inserted and removed at the same end Called Top of Stack (TOS).	2→ Objects are inserted and removed from different Ends Called front and rear Ends.
3→ Insert operation is called push operation.	3→ Insert operation is Called Enqueue operation.
4→ Delete operation is called pop operation	4→ Delete operation is Called Dequeue operation.
5→ In Stack There is no wastage of memory Space.	5→ In Queue there is a wastage of memory Space.
6→ plate Counter at Marriage Reception is an Example of Stack.	6→ Students Standing in a line at fees Counter is an Example of Queue.

Difference between Singly and Doubly Linked List

Singly Linked List	doubly Linked List
1→ Singly Linked List has nodes with data fied and next Link field (forward link). &⇒ [Data | next]	1→ Doubly Linked List has nodes with data field and two pointer field. (Backword and forward Link). &⇒ [Previous | Data | Next]
2→ It allows traversal only in one way.	2→ It allows a two way traversal.
3→ It requires one List poiter variable (Start)	3→ It requires two List pointer variable (Start and Last).
4→ It occupies Less memory	4→ It occupies more memory.
5→ Complexity of Insertion and Deletion at known position is $O(n)$	5→ Complexity of Insertion and Deletion at known position is $O(1)$.

Difference between Linear and 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 ÷ E➜ Array, stack, queue etc.	1➜ In this data structure data is organized without any sequence. E➜ Tree, Graph etc.
2➜ In Linear data Structure Single Level is involved.	2➜ In non-Linear D.S 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➜ Applications of Linear D.S are mainly in Application Software development.	6➜ Applications of non-Linear D.S are in Artificial Intelligence and image processing.

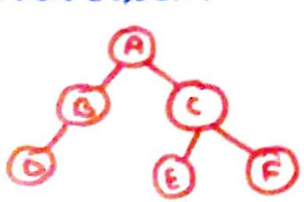
Difference between Array and Linked List

Array	Linked-List
1 → Size of an Array is fixed	1 → Size of a List is not fixed.
2 → Array is a Collection of Homogeneous (Similar) data type.	2 → Linked-List is a Collection of node (data & address)
3 → Memory is allocated from Stack.	3 → Memory is allocated from heap.
4 → Array work with Static data Structure.	4 → Linked-List work with Dynamic data Structure.
5 → Elements are Stored in Contiguous memory Locations.	5 → Elements Can be Stored anywhere in the memory.
6 → Array Elements are independent to Each other.	6 → Linked List Elements are depend to Each other.
7 → Array take more time. (Insertion & Deletion)	7 → Linked-List take Less time. (Insertion & Deletion)

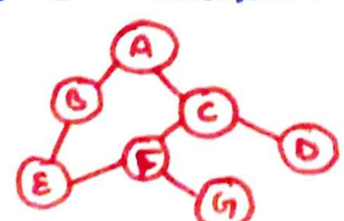
Difference between Tree and Graph

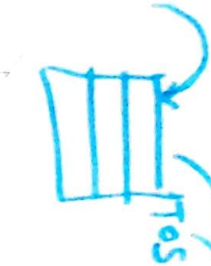
Tree	Graph
1→ Tree is a Collection of nodes and Edges. Ex→ T = {node, Edges}	1→ Graph is a collection of vertices/nodes and Edges. Ex→ G = {V, E}
2→ There is a unique node called root in tree.	2→ There is no unique node.
3→ There will not be any Cycle/Loops.	3→ There can be loops/Cycle.
4→ Represents data in The form of a tree structure, in a hierarchical manner	4→ Represents data similar to a network.
5→ In tree only one path between two nodes.	5→ In Graph one or more then one path between two nodes.
6→ In this Preorder, Inorder and Postorder Traversal.	6→ In this BFS and DFS traversal.

Ex→



Ex→





(DS) Data Structure Most important question for Exam.

Q1 ⇒ What do you mean by data structure?. Explain Different types of Data structure in detail.

Q2 ⇒ What do you mean by Space Complexity and time complexity of an algorithm?. write an algorithm/pseudo code for Binary Search and mention the best case and worst case time Complexity of Binary search.

Q3 ⇒ How array is implemented in memory and how the address of an element can be calculated in one and two dimensional array.

Q4 ⇒ What do you mean by stack?. Write an algorithm for stack push and pop operation.

Q5 ⇒ Write the Prefix and Postfix form of each of the following infix notation:-

a) A-B+(M$N)*(O+P)-Q/R^S*T+Z

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

Q6 ⇒ What is ment by Circular queue and priority queue. write a function to insert and delete an element from a circular queue.

Q7 ⇒ Define recursive function. What are the essential conditions to be satisfied by a recursive function.

Camlin

Q 8 ⇒ What do you mean by Linked List?. Explain Different types of Linked List. Describe the functional code for inserting and deleting a desired node in singly Linked List.

Q 9 ⇒ What is B-tree? Generate a B-tree of order 5 with the alphabets arrive in the sequence as:
a, g, b, k, d, h, m, j, e, s, i, r, x, c, l, n, t, u, p

Q 10 ⇒ Describe Sorting and types. Write an algorithm for insertion sort and quick sort.

Q 11 ⇒ The inorder and preorder traversal of a tree are given below:
Inorder: DBMINEAFCJGK
Preorder: ABDEIMNCFGJK
a) Construct the corresponding Binary tree.
b) Determine the postorder traversal of the tree.

Q 12 ⇒ Write Kruskal's and Prim's algorithm and Explain with Example.

Q 13 ⇒ Explain the following.
a) BFS and DFS
b) AVL tree
c) hash function
d) BST

Camlin