

Data Structures

Notes

Data Structure

(1)

Data \Rightarrow Anything to give information is called data.

Ex \Rightarrow Student Name, Student Roll no.

Structure \Rightarrow Representation of data is called structure.

Ex \Rightarrow graph, Arrays, List.

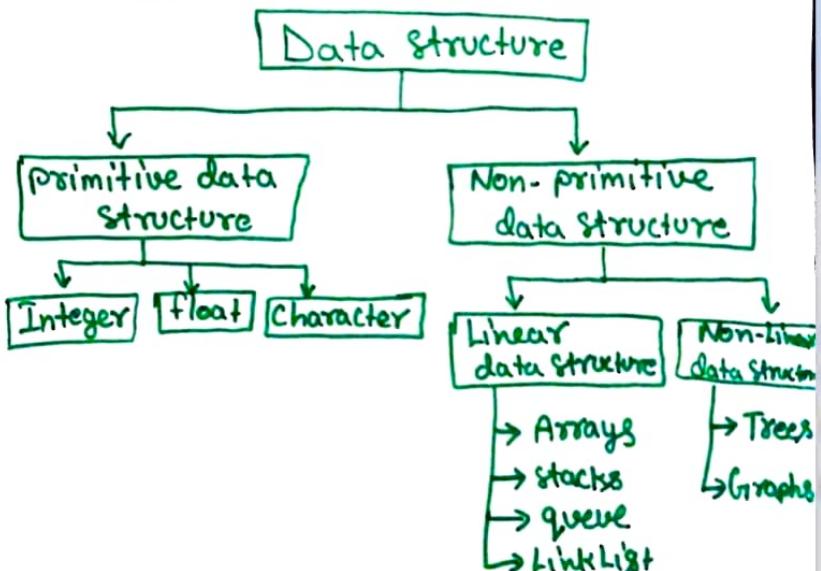
Data Structure \Rightarrow

- 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 \Rightarrow

There are mainly two types of data structure.



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

Ex \Rightarrow integer, float, character.

Non-primitive data structure \Rightarrow These are derived from the primitive data structure. It's a collection of same type or different type primitive data structure.

Ex \Rightarrow Arrays, stack, trees.

Data Structure operation \Rightarrow

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

- i) Insertion \Rightarrow Add a new data in the data structure.
- ii) Deleting \Rightarrow Remove a data from the data structure.
- iii) Sorting \Rightarrow Arrange data in increasing or decreasing order.
- iv) Searching \Rightarrow find the location of data in data structure.
- v) Merging \Rightarrow Combining the data of two different sorted files into a single sorted file.
- vi) Traversing \Rightarrow Accessing each data exactly one in the data structure so that each data item is traversed or visited.

Arrays

(5)

- 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 ➡ It's also known as One Dimensional (1D) Array.
- It's use Only one Subscript to define the Elements of Arrays.

Row [Col]
 ↓

Declaration \Rightarrow

(6)

data-type var-name [Expression],
size

Ex \Rightarrow int num[10];
char C[5];

Initializing One-Dimensional Array \Rightarrow

Data-type var-name [Expression] = {values};

Ex \Rightarrow 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 \Rightarrow multidimensional Arrays use more

than one Subscript to describe the
Arrays Elements. $\{ \} \{ \} \{ \}$ —

Two Dimensional Arrays \Rightarrow It's use two
Subscript, one Subscript
to represent row value and second
Subscript to represent column value.
It mainly use for matrix representation.

Declaration two-dimensional Arrays \Rightarrow

(7)

data-type var-name [rows][columns],

Ex \Rightarrow int num[3][3],

Initialization 2-D Arrays \Rightarrow

Data-type var-name [rows][columns] = {values},

Ex \Rightarrow int num[3][2] = {1, 2, 3, 4, 5, 6},
or
int num[3][] = {1, 2, 3, 4, 5, 6},

A diagram showing a 3x2 matrix. The matrix has 3 rows and 2 columns. The elements are labeled as 1, 2 in the first row, 3, 4 in the second row, and 5, 6 in the third row. The matrix is enclosed in brackets and labeled "3x2" below it.

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

$$num[0][0] = 1$$

$$num[0][1] = 2$$

$$num[1][0] = 3$$

$$num[1][1] = 4$$

$$num[2][0] = 5$$

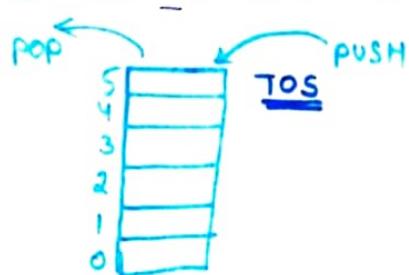
$$num[2][1] = 6$$

Write a program to read & write one dimensional Array.

```
(8)
# include <stdio.h>      Standard input output
# include <conio.h>      printf Scanf
Void main()
{
    int a[10], i;
    Clrscr(), getch()
    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) (9)

- 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.

(10)

There are two operation of stack.

1) PUSH operation \Rightarrow The process of adding a new element to the top of stack is called PUSH operation.

- Every new element is added to stack top is incremented by one.
- In case the array is full and no new element can be added it is called Stack full or Stack overflow Condition

2) POP operation \Rightarrow The process of deleting an element from the top of stack is called POP operation.

- After every POP operation the stack 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

(11)

→ Stack has two operation.

1) PUSH operation \Rightarrow

2) POP operation \Rightarrow

1) PUSH operation \Rightarrow The process of adding a new element of the top of stack is called PUSH operation

- Every PUSH operation Top is incremented by one.

$$\text{TOP} = \text{TOP} + 1$$

- 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). 12

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

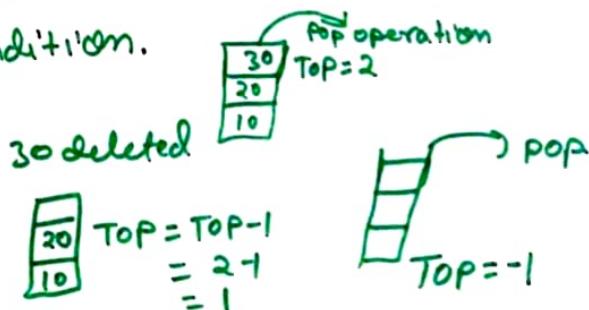
Step 6: print "stack overflow"

2 ⇒ POP Operation ⇒ 13

- 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.

$$\text{Top} = \text{Top} - 1$$

- 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) (14)

pop (Stack [maxSize], item)

Step1: Repeat Steps 2 to 4 until $\text{TOP} \geq 0$

Step2: Set item = Stack [TOP]

Step3: Set $\text{TOP} = \text{TOP} - 1$

Step4: print, No. deleted is, Item

Step5: Print stack under flows.

Stacks (prefix & postfix) (15)

Stack Notation \Rightarrow There are three stack Notation.

1) Infix Notation \Rightarrow where the operator is written in-between the operands.

Ex \Rightarrow A + B + operator
 A, B operands

2) Prefix Notation \Rightarrow In this operator is written before the operands.
It is also known as polish Notation.

Ex \Rightarrow + AB

3) Postfix Notation \Rightarrow In this operator is written After the operands.

It is also known as Suffix Notation.

Ex \Rightarrow AB+

Q \Rightarrow Convert the following Infix to prefix and Postfix for $(A+B)*C/D+E^F/G$

prefix \Rightarrow $(A+B)*C/D+E^F/G$
 $+AB*C/D+E^F/G$

Let $+AB = R_1$

$$R_1 * C / \Delta + E^N F / G$$

(16)

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

$$\text{Let } \Rightarrow \wedge E F = R_2$$

$$R_1 * C / \Delta + R_2 / G$$

$$R_1 * / C D + R_2 / G$$

$$\text{Let } \Rightarrow / C D = R_3$$

$$R_1 * R_3 + R_2 / G$$

$$R_1 * R_3 + / R_2 G$$

$$\text{Let } \Rightarrow / R_2 G = R_4$$

$$R_1 * R_3 + R_4$$

$$* R_1 R_3 + R_4$$

$$\text{Let } * R_1 R_3 = R_5$$

$$R_5 + R_4$$

$$+ R_5 R_4$$

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

$$+ * R_1 R_3 / R_2 G$$

$$+ * + A B / C D / \wedge E F G /$$

$$\text{postfix} \Rightarrow (A+B) * C / \Delta + E^N F / G$$

(17)

$$(\bar{A} \bar{B}+) * C / \Delta + E^N F / G$$

$$\text{Let } \bar{A} \bar{B}+ = R_1$$

$$R_1 * C / \Delta + E^N F / G$$

$$R_1 * C / \Delta + (\bar{E} \bar{F}^N) / G$$

$$\text{Let } \bar{E} \bar{F}^N = R_2$$

$$R_1 * C / \Delta + R_2 / G$$

$$R_1 * (\bar{C} \bar{D}) + R_2 / G$$

$$\text{Let } \bar{C} \bar{D} = R_3$$

$$R_1 * R_3 + R_2 / G$$

$$R_1 * R_3 + (\bar{R}_2 G) /$$

$$\text{Let } \bar{R}_2 G / = R_4$$

$$R_1 * \underline{\bar{R}_3} + R_4$$

$$(R_1 R_3 *) + R_4$$

$$\text{Let } R_1 R_3 * = R_5$$

$$R_5 + R_4$$

$$\underline{R_5 R_4 +}$$

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

(18)

$$\begin{array}{c} R_5 R_4 + \\ R_1 R_3 * R_4 + \\ AB + CD / * R_2 G I + \\ \hline AB + CD / * (EF ! G) I + \end{array}$$

Postfix Expression

prefix and postfix using tabular form (19)

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

to convert in prefix following operation:

position

- 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 brackets

$(A + B * C)$

Reverse string

$(C * B + A)$

Tabular form.

Symbol Scanned	Stack	Postfix Expression
((-
C	(C
*	(*	C
B	(* B	CB
+	(* B +	$\rightarrow CB*$
A	(* B + A	$CB*A$
)	-	$CB*A+$

So the postfix Expression $CB*A+$. Now (20)
reverse this Expression to get the prefix
so prefix is

$+ A * BC$ prefix

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

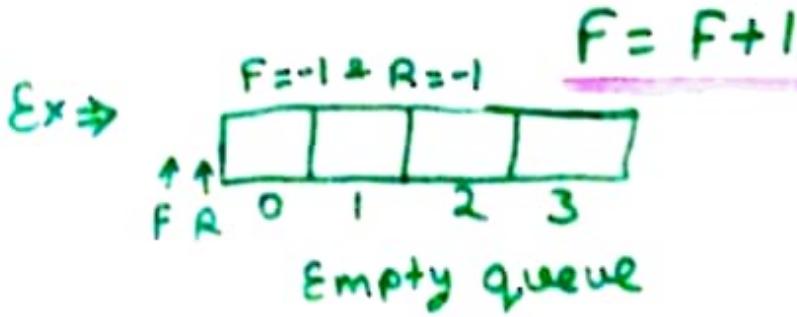
Symbol Scanned	Stack	Postfix Expression
((-
A	(A
+	(+	A
B	(+ B	AB
*	(+ B *	AB
C	(+ B * C	ABC
)	X	<u>ABC#+</u>

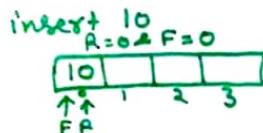
postfix Expression = $ABC*+$

Queues

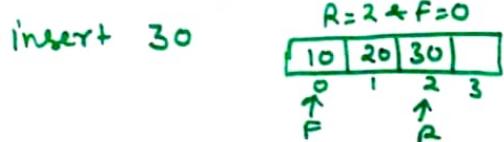
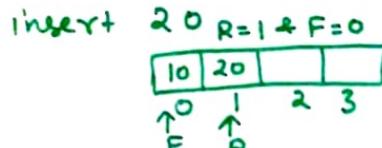
(21)

- 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

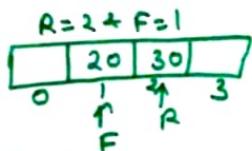




(22)



deleted Element. first delete 10



deleted Second Element.



operation on Queue

(23)

1) To insert an Element in a Queue \Rightarrow

Algo \Rightarrow

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 \Rightarrow

QDELETE (queue[maxsize], item)

(24)

Step 1: Repeat step 2 to 4 until $front \geq 0$

Step 2: Set item = Queue[front]

Step 3: If front == Rear

 Set front = -1

 Set Rear = -1

Else

 front = front + 1

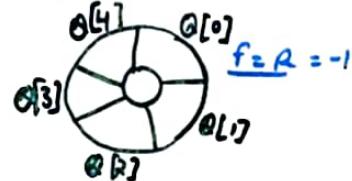
Step 4: print, No. Deleted is, item

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

CIRCULAR QUEUE

(25)

- # 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
$$\underline{\text{Rear} = \text{Rear} + 1}$$
- 4) Each time an Element is deleted from the queue the value of front is incremented by one.
$$\underline{\text{front} = \text{front} + 1}$$

Insert an element in Circular Queue \Rightarrow 26
Algo \Rightarrow QINSERT (QUEUE [MAXSIZE], Item)

Step 1 \Rightarrow 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 \Rightarrow Exit

Queue (Data Structure) 27

operation on Queue

Ex \Rightarrow

--	--	--

maxsize = 3

1) front = -1 Rear = -1 Empty queue

2) 3 to 5 Step Repeat

$R < maxsize - 1$

$-1 < 3 - 1$

$-1 < 2$ true

$\begin{matrix} \swarrow & \searrow \\ 3 & 4 & 5 \end{matrix}$

3) Read item

Read 10

4) $f = -1$

$-1 == -1$ true

$f = 0$

$R = 0$

5) Set $q[0] = \text{item}$
 $q[0] = 10$

$\begin{matrix} 10 & & \\ q[0] & q[1] & q[2] \end{matrix}$

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

Rear < maxsize-1

1 < 3-1

1 < 2 true

Read 30

if f == -1
 0 == -1 false

Else

R = R+1 28

R = 1+1 = 2

5) set q[2] = 30

[10 | 20 | 30]
 q[0] q[1] q[2]
 f=0 R=2

yes Rear < maxsize-1

2 < 3-1

2 < 2 false

6) queue is overflow

DELETE an Element in Circular queue \Rightarrow 29

Algo \Rightarrow QDELETE (Queue[maxsize], I+em)

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]

\rightarrow item deleted.

2. Exit.

QUEUE (Data Structure)

(30)

Delete operation on queue

So \Rightarrow

10	20	30
$q_f[0]$	$q_f[1]$	$q_f[2]$

$F=0$ $R=2$

 $\text{maxSize} = 3$ Case 1 \rightarrow

$f >= 0$

 $O >= 0 \text{ true}$ 2) set item = $q_f[0]$

item = 10

3) $f = R$ $O = 2 \text{ false}$

Else

$f = f + 1$

$f = O + 1 = 1$

4) item is deleted

10 is deleted

	20	30
$q_f[0]$	$q_f[1]$	$q_f[2]$

$F=1$ $R=2$

	20	30
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

	30
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

(31)		

$f = -1$
 $R = -1$

(Case 4) 1) $f >= 0$
 $-1 >= 0$ false

Step 5: queue is Empty
queue is Underflow.

Linked Lists

(32)

- 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.



Ex :-

10	next
info	pointer

Node



Advantages of Linked Lists

(33)

- 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 no longer needed.
- 3) **Insertion and deletions are easier and efficient:** It provides 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.**

Operations ON Linked List:

(34)

The basic operations to be performed on the linked lists are:-

- 1) **Creation:** This operation is 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. (35)

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 (36)

- 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.

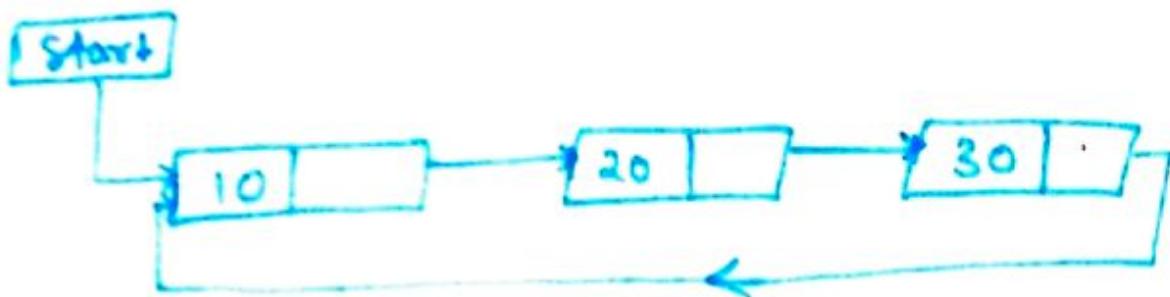


2) Doubly-Linked List ⇒ It's one in which all nodes are linked together by multiple links which help in accessing both the successor node (Next node) and predecessor node (Previous node) within the list. This helps to traverse the list in the forward direction and backward direction.

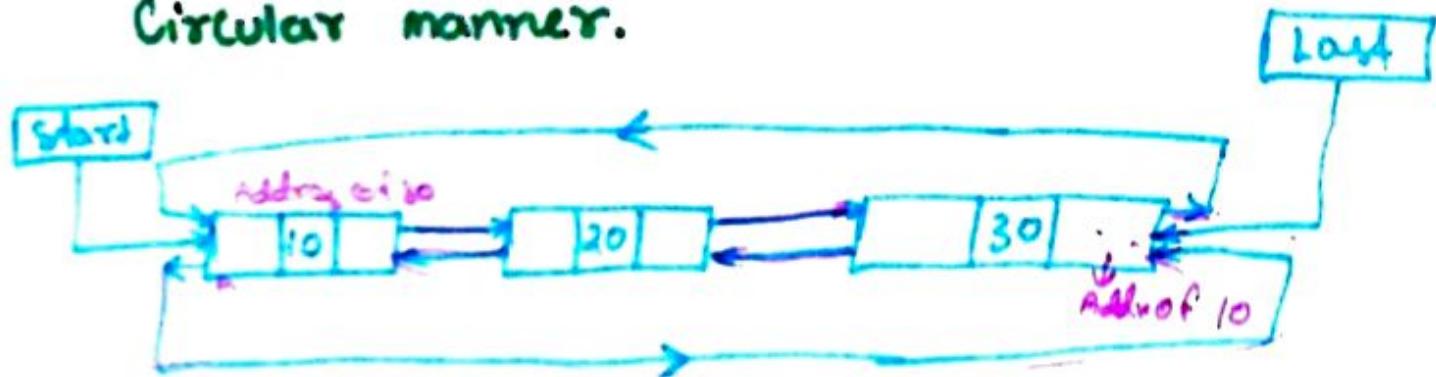


3 Circular Linked List \Rightarrow It's one which has no beginning and no end. A singly linked list can be made a circular linked list by simply setting the address of the very first node in the link field of the last node.

(37)



4 Circular doubly linked list \Rightarrow It's one which has both the successor pointer and predecessor pointer in a circular manner.



Inserting Nodes in Linked List

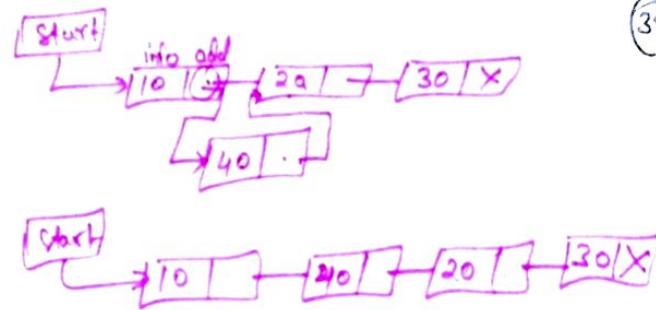
(38)

1) Inserting at the beginning
of the List.

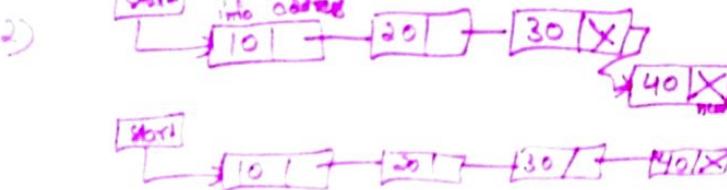
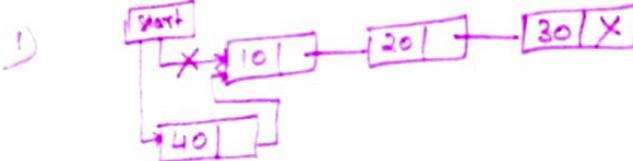
2) Inserting at the End of the List

3) Inserting at the Specified position
Within the List.

3)



(39)



LINKED LIST

Inserting A Node AT the Beginning in Linked List

(40)

Algorithm →

INSERT-FIRST(START, ITEM)

Step1: [check for overflow]

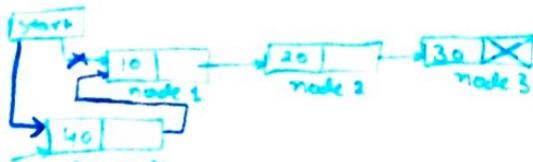
IF PTR = NULL then
print overflow
Exit

Else
PTR = (Node*) malloc(sizeof(Node))
// Create new node from memory and
assign its address to PTR.

Step2: Set PTR → INFO = Item

Step3: Set PTR → Next = START

Step4: Set START = PTR



After insertion



LINKED LIST

Insert A Node AT THE End in singly Linked List

(41)

Algorithm →

Insert-Last(START, ITEM)

Step1: Check for overflow

IF PTR = NULL then
print overflow
Exit

Else
PTR = (Node*) malloc (sizeof(Node));

Step2: Set PTR → INFO = Item ;

Step3: Set PTR → Next = NULL ;

Step4: IF start = NULL and then

Set START = PTR;

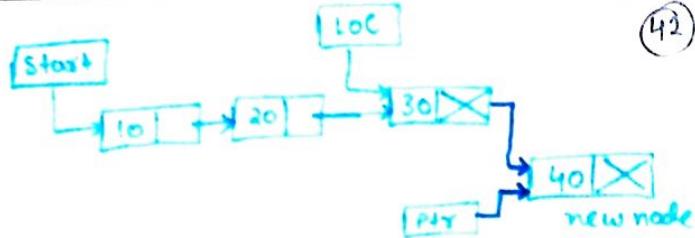
Else,

Step5: Set LOC = Start ;

Step6: Repeat Step 7 until LOC → Next != NULL

Step7: Set LOC = LOC → Next ,

Step8: Set LOC → Next = PTR ,



After Insertion



(43)

LINKED LIST

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

Algorithm →

Insert_Location (START, Item, LOC)

Step1: Check for overflow

If $ptr == \text{NULL}$ then
print overflow

Exit

Else
 $ptr = (\text{Node} *) \text{malloc}(\text{size of(Node)})$

Step2: Set $ptr \rightarrow \text{Info} = \text{item}$

Step3: If $\text{start} == \text{NULL}$ then

Set $\text{start} = ptr$

Set $ptr \rightarrow \text{Next} = \text{NULL}$

Step4: Initialize the Counter I and Pointers

Set $I = 0$

Set $\text{temp} = \text{start}$

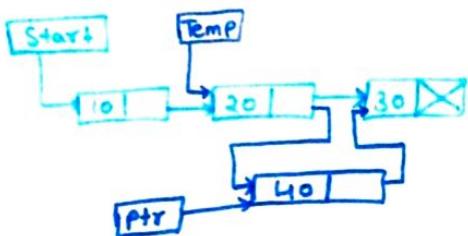
Step 5: Repeat Steps 6 and 7 until $I < LOC$ (44)

Step 6: set $temp = temp \rightarrow Next$

Step 7: set $I = I + 1$

Step 8: Set $ptr \rightarrow Next = temp \rightarrow Next$

Step 9: set $temp \rightarrow Next = ptr$.



After Insertion



Deleting Node in Linked List (45)

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 (46)

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



LINKED LIST DELETING NODES (47)

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

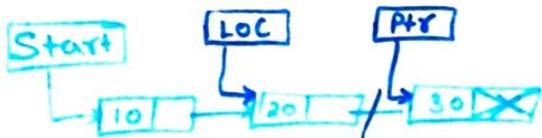
Step4: Repeat Step 5 and 6 until
PTR → Next ! = NULL

Step5: Set LOC = PTR

Step6: Set PTR = PTR → Next

(48)

- Step 7: set LOC \rightarrow Next = NULL
 Step 8: free (PTR)



After deletion



(48)

LINKED LIST DELETING NODES

(49)

Deleting the Node from Specified Position

In Singly Linked List

Algorithm \Rightarrow

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

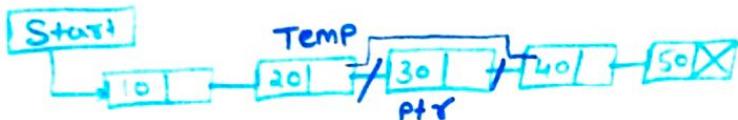
Step 5: Set PTR = PTR \rightarrow Next

Step 6: Set $I = I + 1$

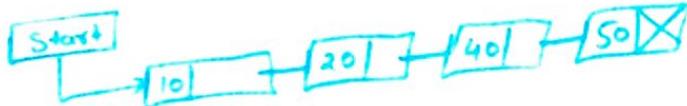
Step 7 : Print Element deleted i.e. $\text{ptr} \rightarrow \text{info}$

Step 8 : Set Temp \rightarrow Next = $\text{ptr} \rightarrow \text{Next}$ (50)

Step 9 : free(ptr)



After deletion

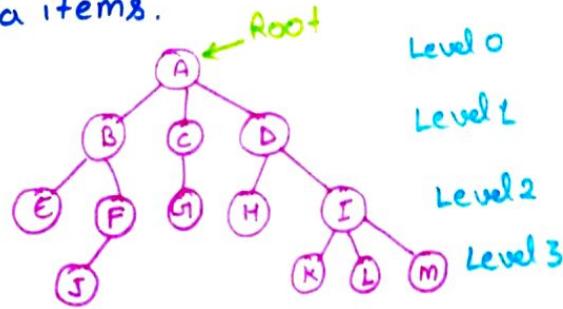


Trees in Data Structure

(5)

Tree \Rightarrow 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 \Rightarrow Tree has different terminology such as:

- 1 \Rightarrow Root \Rightarrow 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 \Rightarrow Node \Rightarrow Each data item in a tree is called a node. In the given

Tree there are 13 Node such as- (52)
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.

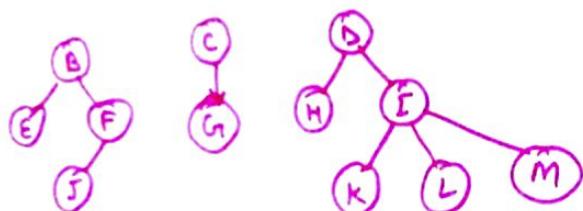
10 ⇒ Path ⇒ It is a sequence of consecutive edges from the source node to the destination node. In the given tree the path between A and J is as.

(A, B) (B, F) and (F, J)
 $A \rightarrow B \rightarrow F \rightarrow J$

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

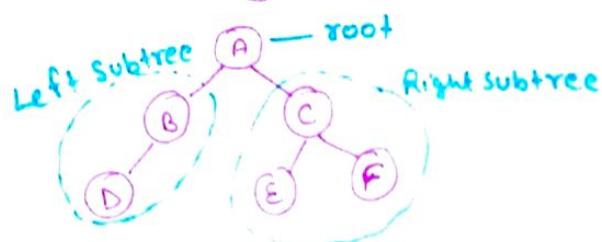
12) forest \Rightarrow It is a set of disjoint trees. In a given tree if you remove its root node then it becomes a forest. In the given tree, there is forest with three tree. Such as.

After removing root A. forest is.



BINARY TREES

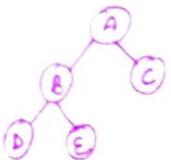
- Binary tree is a finite set of data item which is either empty or consists of a single item called root and two disjoint binary tree called the Left subtree and right subtree
- In Binary tree, Every node can have maximum of 2 children which are known as Left child and Right child



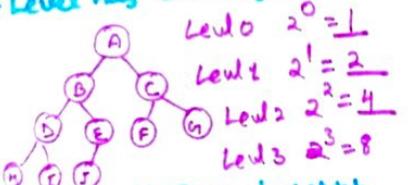
Types of Binary trees \Rightarrow

(56)

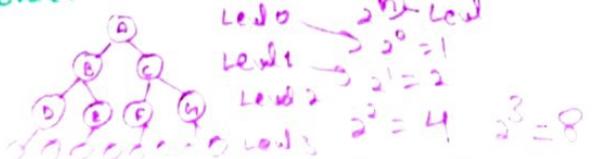
- 1) full Binary tree \Rightarrow A binary Tree is full if every node has 0 or 2 children.



- 2) Complete Binary tree \Rightarrow 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.



- 3) Perfect Binary Tree \Rightarrow A Tree in which all internal nodes have 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 - Pre order traversal (NLR)
- 2 - In order traversal (LNR)
- 3 - Post order traversal (LRN)

1 \rightarrow Pre order Traversal \Rightarrow In this Traversal method, the root (N) is visited first, then the left subtree (L) and finally the right subtree (R).

Algorithm \Rightarrow

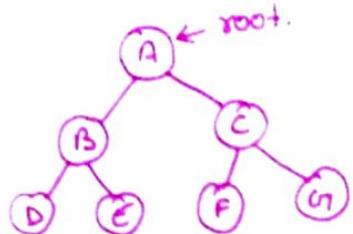
Until all nodes are traversed -

Step 1: Visit root node.

Step 2: Recursively traverse Left Subtree.

Step 3: Recursively traverse Right Subtree

Ex →



(58)

Pre-order traversal is →
A, B, D, E, C, F, G.

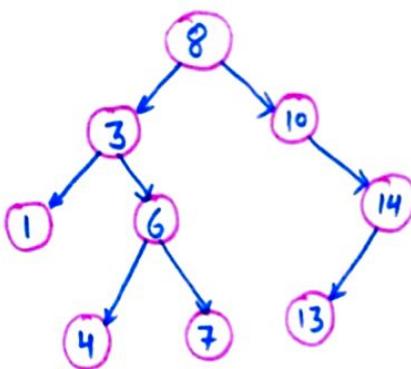
2 → Inorder Traversal ⇒ In this traversal method, the Left Subtree is visited first, then the root and later the right Subtree.

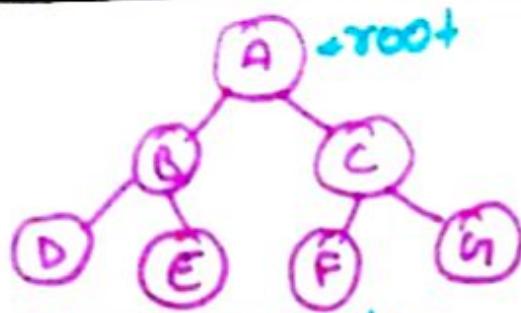
Algorithm ⇒

until 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).





Inorder Traversal is -
D, B, E, A, F, C, G.

\Rightarrow Post-order Traversal \Rightarrow In this method the root node is visited last, hence the name first we traverse Left subtree, then the right subtree and finally the root node.

Algorithm \Rightarrow

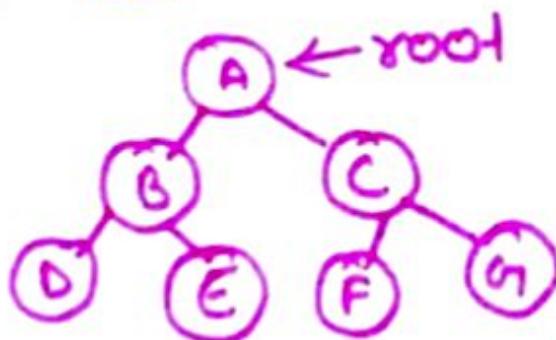
until All nodes are traversed -

Step1: Recursively traverse Left subtree.

Step2: Recursively traverse right subtree.

Step3: Visit root node.

\Rightarrow



Post order Traversal is -

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

Difference between Stack and Queue

(61)

Stack

- 1 ➔ It represents the collection of Elements in Last in first 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 First In First Out (FIFO) order.
- 2 ➔ Objects 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.

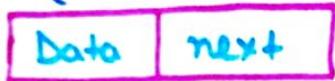
Difference between Singly and Doubly Linked List

(62)

Singly Linked List

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

⇒



- 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).

⇒



- 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 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 :- 5 ➤ Array, Stack, Queue etc.	1 ➤ In this data structure data is organized without any sequence. 5 ➤ 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 an efficient way.	5 ➤ memory is utilized 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

(63)

Array

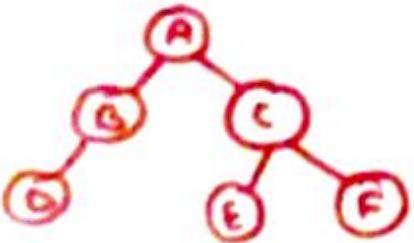
- 1 ➔ Size of an Array is fixed.
- 2 ➔ Array is a Collection of Homogeneous (Similar)datatype.
- 3 ➔ Memory is allocated from stack.
- 4 ➔ Array work with Static data structure.
- 5 ➔ Elements are Stored in Contiguous memory Locations.
- 6 ➔ Array Elements are independent to Each other.
- 7 ➔ Array take 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

(64)

Tree	Graph
1 → Tree is a collection of nodes and edges. Ex → $T = \{ \text{node}, \text{edges} \}$	1 → Graph is a collection of vertices/nodes and edges. Ex → $G = \{ V, E \}$
2 → There is a unique node called <u>root</u> 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 than one path between two nodes.
6 → In this Preorder, Inorder and Postorder Traversal.	6 → In this BFS and DFS traversal.
Ex → 	Ex → 

Thank
you!