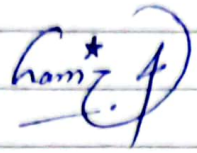
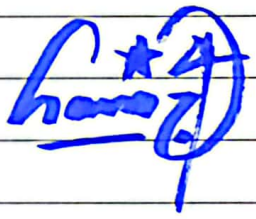
Date_____20__
M T W T F S



"Muhammad Hamza
Software Engineer"

"Data Structure
&
Algorithms"



Date _____ 20__
M T W T F S

"DSA"

DSA :-

A data Structure is a way which Organized and store data which Compiler can used more effiently.

Abstract Data type :-

ADT like a blue print that will store some data.

In this data we can add, remove and search for data.

→ It also Organized the data in a certain way.

→ It's not tell about implement.

List :-

List is ADT which store the same type.

Just like list of name, number etc.

List

Static list

- it have fixed sized.
- It take fixed memory if we used or not.
- It used when we have given a specific number of list.

Dynamic list

- It have a variable size and it can be changed it run time.
- If we need a long list we can expand it on run time.
- we used a dynamic list if we don't know how much length of list as?

- Remove
- add
- Search
- find inden

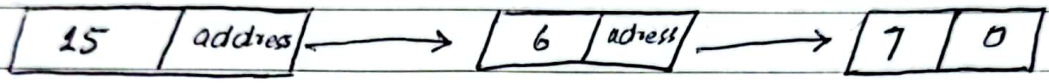
→ List is not effient in memory.

Date_____20__
M T W T F S

LinkedList :-

It is a linear data structure where each element is a separate object known as a node.

1. The first node is called Head Nodes.
2. Last node is Tail.




```
struct {
    int data;
    Node* next;
};
```

Operation in linked list :-

1. Insertion :- Insert new nodes at specific position.
2. Delete :- Delete a node at specific position.
3. Traversal :- Iterating through the list visiting each node sequence.

Date _____ 20___
M T W T F S

Link List vs Array :-

Both have good and efficient data structure used according to the requirements.

1. Const time :-

→ Cost of accessing in element array is best and good choice to use because it take less time according to link list.

Const time O(1) ↳ Array.

Const time O(n) ↳ linked list

2. Memory requirements :-

→ In case of array it have a fixed size commonly it have used and unused bit of memory. In case of linked list it have no fixed size but the all nodes is used. Extra memory for pointers variables.

→ We can extend the size of linked list any time so the array have not. So linked list in this case is better choice.

Date______20__
M T W T F S

Stack :-

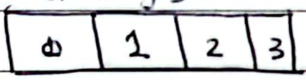
A Collection of Some data type in which the removal and insertion of items Can take place from one end is called top.

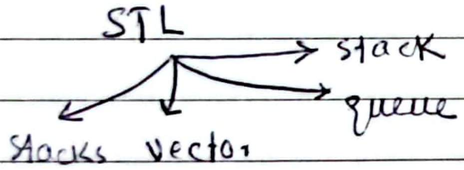
LIFO
(Last in first out)

→ BrackTracking means Recursion

vector :-

it is array like. but Size is not fixed like arrays. (STL)
std Templete library.





Syntax :-

vector <int> val
Keyword <datatype> name

by default it size is 0.
we also include library #include<vector

Date 20
M T W T F S

Stack :-

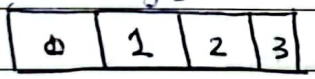
A collection of some data type in which the removal and insertion of items can take place from one end is called top.

LIFO
(Last in first out)

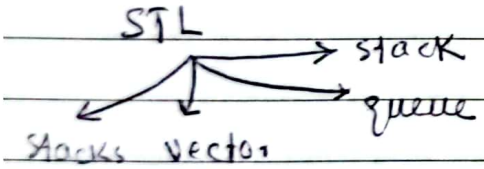
→ BrackTracking means Recursion

Vector :-

it is array like. but size is not fixed like arrays.



(STL)
std Template library.



Syntax :-

vector < int > val
Keyword < datatype > name

by default it size is 0.
we also include library #include<vecto

Date 20
M T W T F S

Initialization of vector :-

① `Vector <int> vec = { 1, 2, 3 };`
② `Vector <int> vec (3, 0)` → value at index
↳ size of vector

for each loop :-

`for (int i : vector)`

dataType of itterater must be same to vector.

Vector function :-

- Size — `vector.size();`
- push_back — adding value
- pop_back — Remove value from last index
- front
- back
- at — accessing a specific index value. `(vec.at(i);`

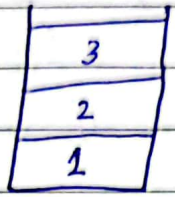
Static vs Dynamic Allocation :-

Compile time	Run time
Array	vector
fixed size	Resize

Date_____20__
M T W T F S

Stack :-

LIFO
(Last in first out)



For insert we use push operation
Remove we use pop Operation
peak means Top element
empty. check that stack is empty or not.
Size,

STL :-

```
stack <data type> S;
push  →  S.push(2);
pop   →  S.pop();
Top   →  S.Top();   // give tops value.
```

for stack we include stack

Implement Stack Class :-

check stack if stack is no avaliable space
it means stack is overflow.

check element is present

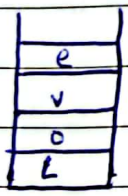
F underflow

Date 20
M T W T F S

if stack is empty it's means :-

Top = -1 → stack empty.
return stack[top];
if they return -1 stack is empty.

Problem :- Reversing a String using Stacks.



"Love"

↳ if i pop this they will print in reverse.

ans = "evoL"

Kadanes Algorithm :-

They can only remove +ive value from array sum.

Maximum Sub Array Sum :-

1	2	3	4	5

we can find the Sub-Array by ussing the formula :-

$$\text{Sub-Array} = \frac{n * (n+1)}{2}$$

n = no of elements in ~~Sub Array~~.

Date 20
M T W T F S

1	2	3	4	5

Sub-Array mean how many possiable value is made

1, 2, 3, 4, 5
12, 23, 34, 45,
123, 234, 345
1234, 2345
12345

Sub-A = $\frac{5*(5+1)}{2} = \frac{5*6}{2} = \frac{30}{2} = 15$

15 Sub Array is possible

To print the Sub-Array using code:-

Code :-

```
int main() {
    int n = 5
    int arr[5] = { 1, 2, 3, 4, 5}
for(int st=0; st<n; st++){          //using the nested
                                    // loops
  for(int end= st; end<n; end++){
    for(int i= st ; i<end; i++) {
        cout << arr[i];
    }
    cout << " ";
}
cout << endl;
}
```

Date_____20___
M T W T F S

↳ Time and Space Complexity :-

Time Complexity :-

It is amount of time which algorithms take to run is called —.

→ as a function of length of the input.

↳ Why ?

→ For making better program.

Comparision of Algorithms.

Big O notation ↓ upper bound

Theta Θ ↓ for average case

Omega Ω ↓ minimum or lower bound.

Const time → O(1) if we have fixed value

linear time → O(n) if we have n values

logrithm time → O(log n)

Quadritic time → O(n²) use if we have nested loop (2 loops) use the condition upto n.

Rules :-

If we find the time Complexity if nested loop we multiply it.

external loop added.

Date _____ 20__
M T W T F S

Example 8-

①
```
for (int i = 0; i < n; i++) {
    for (int j = i; j < n; j++) {
    }
};
```

Time Complexity is

$$O(n \times n) = O(n^2)$$

②
```
for (int i = 0; i < n; i++)
{
}
for (int j = 0; j < n; j++)
{
};
```

Time Complexity is

$$O(n+n)$$ Ans

★=★

Qw

Date 20
M T W T F S

2→ Space Complexity:- How many memory can take the algorithm.

int a=0; int b=0; O(1) means Const Space c.
int arr[5] = {1,2,3,4,5} O(1) because fixed size is Const Space Complexity.

Uses of Stacks:-

① Useful for processing nested Structures. checking that mathematically brackets expresion is balance or not.

② To implement function calls and Returns:- when a program runs it open call functions. for this System use tracks of which function is running and what happen when a function is finished. Stack work like show direction to a function.

③ Stacks helps to Order the mathematical expresion. and calculate the results. just like solving the postfix, infix and prefix.

Date 20
M T W T F S

↳ When function is called :-

In Stacks when function is called it excutes their block of code and then after return to their function call back place that returning place is called return adress.

Code :-

```
int main() {
    int n = 5;
    cout << " Factorial of " << n << " is : " << factorial(n) << endl;
};
int factorial(int num) {
    if (n == 0) {
        return 1;
    }
    else
    {
        return n * (factorial(n-1));
    };
```

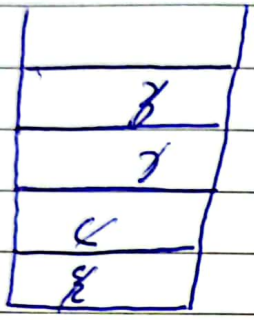
Date 20
M T W T F S

Symbol balancing:-

To check opening and closing brackets or correct or not. It cause compiler error.

Example:-

{a+b(c)}



Stack is empty and balanced.

valid expresion.

STACK ADT Operations:-

1. Initialize it an empty State.
2. IsEmpty check stack is empty or full.
3. IsFull check that Stack is currently full.
4. push Addnew item to top of Stack.
5. pop Remove item from the top.

Dynamic Memory Allocation:-

→ Memory allocated during Compile time is called Static memory.

Memory allocated is fixed in Static memory.

Size is fixed

Date _____ 20__
M T W T F S

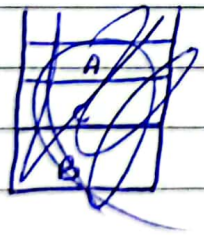
↳ Memory allocating during time of execution is called dynamics memory allocation.
- It's prevent from extra memory.

→ When we implement the dynamics we used Keywords New.

→ new is Keyword and create space.

placement of Operators:-

① Infix A+B
② postfix AB+ → imp because computer understand it.
③ prefix +AB



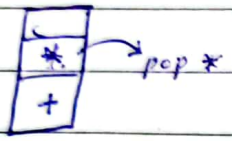
~~A+(B+C)~~

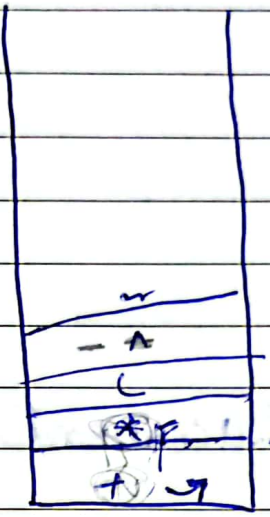
Example :-

$$a+b*(c\hat{}d-e)\hat{}(f+g*h)-i$$

Sol

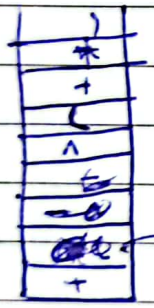
ab*





Output :-

ab*cd^e-fg^*+i-



output : abcd^e

Date 20
M T W T F S

Recursions-

function calling itself is called Recursion.

↳ if base case is not given the function can call infinite time again and again.

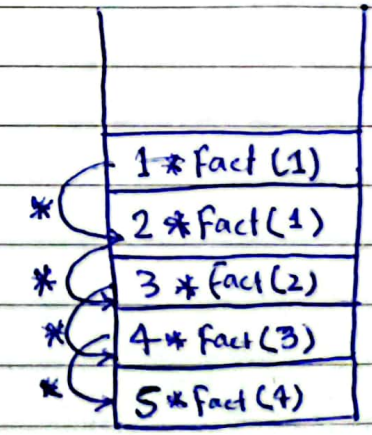
Mostly used for :-

1. chess games
2. Tower of Hanoi
3. Mathematically formulas
4. Scientific formulas-

↳ factorial :-

Code :-

```
long factorial (int n) {
    if (n == 0)        // base condition.
    { return 1;
    }
    else
    { return n * factorial(n-1);
    }
};
```



Factorial Stacks imp.

5! 20 x 3 x 2 x 1
60 x 2 x 1
120 Ans

3 x 2 x 1
6 x 1
6

Date _____ 20__
M T W T F S

using in Real Life :-

Stack :-

1. used for undo / Redo functionality
2. Browser history

Recursive :-

It is mostly used for games.
e.g:- Tic-Toe, chess, sudoku etc.

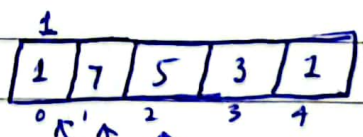
↳ Sum of digits :-
using Recursion

Input = 1234 123 + 123
output = 10

Queue :-

Queue :-
It's means line first in first output.
(FIFO)

push → rear // insertion to rear
pop → front



Queue :-
push
pop
size
isEmpty
} operations

push (1)
"
"

Date _____ 20__
M T W T F S

↳ Queue is mostly used in networking.

STL :-

```
queue<int> q;
q.push(11);
q.pop();
q.sizeo;
q.empty(); Return T/F.
q.front(); show us the front element.
```

Queue is implemented by two types :-

i) Through Array
ii) Through linked list.

class Queue :-

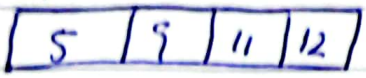
Data member required is
array // due to declaration
Size // array size must be needed
front // To insert new element

↳ Circular Queue :-

A type of queue in which the last position is connected back to the first forming a circle. When the queue is full and an element is removed new element can be added to the empty spot but the queue follows (FIFO) Rules.

Size = 4;

Step 2:- front

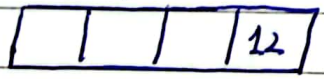


pop();

front

pop();

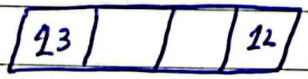
pop();



Wrapped around Concept is based on modulus. % ✓.

Now when I push the value the new value will take place in 1st position.

push(13);



But front is still 22.

2→ So this is called Circular queue.

→ push →

→ Input Restricted Queue :- insert and push is allowed from one side to rear. but pop will available from both side.

push_back();
pop_front();
pop_back();

Date_____20__
M T W T F S

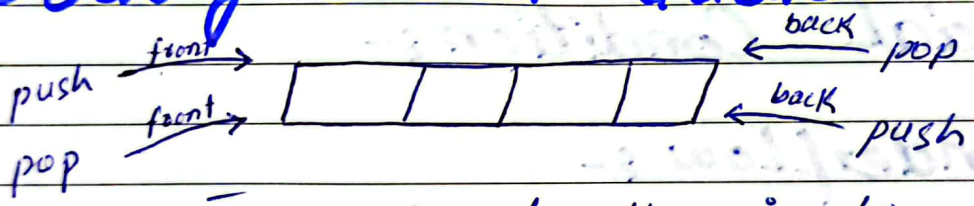
2. Output Restricted Queue :-

Same as input restricted queue but opposite.

insert or push is allowed from both side and pop is allowed from one side.

```
push_front();
push_back();
pop_front();
```

3. Doubly Ended Queue :-



They allowed the inserting and removing from both side (front and back).

→ using in cpu scheduling etc.

STL :-

```
push_front();
push_back();
pop_front();
pop_back();
```

Date_____20___
M T W T F S

Problem :-

Reverse the number using queue.

Sol :-

I use the stack because when I push the element from queue to stack and them after pop from stack its order will reverse.

1. push the queue element to stack.
2. pop from the stack to queue.
3. Order will reverse
4. So print it.

→ Special Condition :-

i :- underflow :-

if both front and rear are -1

ii :- overflow :-

If the rear of the list is one spot behind the front of the list.

front == (rear + 1) % maxQue

$\{ \begin{matrix} i = 0 \\ n = n-1 \end{matrix} \}$

Date 20
M T W T F S

Codes:-

Insertion in a Queue

// is Queue Full
If (Front == 1 AND Rear == N) OR (Front == Rear + 1)
check for linear — check for circular
write overflow return.
// Find new value of rear
if (Front == Null AND Rear == 1)
Then Front = 1 and Rear = 1;
else if (Rear == N) Then
Rear = 1;

Else
Rear = Rear + 1;

Q[Rear] = item;

Transformers:-
insert
Remove
Initilize

IsFull
IsEmpty

$2^2 = 2 \times 1 = 2$
$2 \times 2 = 4$

Date 20
M T W T F S

2 → need more practice in prefix and postfix.

→ Priority Queue :-

Sometime it is not enough just do FIFO Ordering

- may want to give some items a high priority than other items.

Two Major way of Implementation :-

1 :- inserted item in Sorted Ordered
→ always removed the head.

2 :- insert in unordered ordered and Search list on remove
→ always add to the tail.

either way time is Same.

(7 to 8 seconds in 512 GB RAM)

→ Lab 04 :-

```
front == rear;          // Empty
front == rear == -1     // Empty
front == -1 && rear == -1; // Empty
```

12 , 19, 2, -5 ;

-5

Date _____ 20___
M T W T F S

(+ +)

↳ Check for balanced parentheses :-

∴ if the opening and closing are match then it is called balanced expresion.

{(A+B) + (C+D)} balanced

{(x+y) * (z) Not balanced due. { is present and closing is not.

count of opening = count of closing but also it's position.

↳ Infix, postfix, prefix :-

2+3 } → infix
A+B }

➤ PRECEDENCE :-

1. parentheses (), {}, [ ]
2. Exponent (Right to left)
3. Multiplication and division } ∴ left to Right
4. Addition and Subtraction }

Exponent Rules :↓

$2^3^2 = 2^9 = 512$

Date_____20__
M T W T F S

PREFIX :-

+ AB

+ 23

< operator > < operand > < operand >

POSTFIX :-

AB+

23+

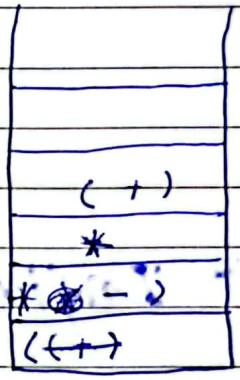
→ Infix to postfix :-

① ((A+B) * C - D) * E

AB+C*D-E*

② A * (B + C)

A BC+ *



→ TIME COMPLEXITY OF QUEUE :-

1 EnQueue
2 DeQueue
3 Front().
4 IsEmpty()

} const time or O(1)

Date_____20__
M T W T F S

↳ Pointers:-

Memory Adress:-

int a = 10;

They occupy the memory.

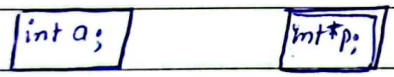
↳ from this memory we can find which number is stored. The address is store in the form of hexadecimal.

```
int a = 10;
cout << &a << endl; // They show address
```

Defs-

Store variable that store address of other variables.

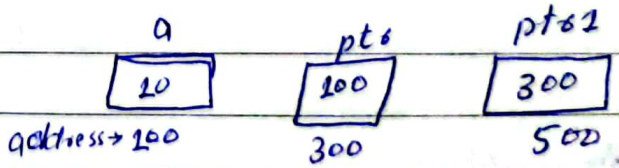
* To make pointer.



address = 002f address = 005fc

↳ The adress of a store p but the address of p is different.

→ pointer to pointer:-




```
int a = 10;
int* ptr = &a;
int** ptr1 = &ptr; // They store the address
                   // of the other pointers.
```

↳ The address of a is value of ptr and the ptr address is value of ptr1.

Date_____20__
M T W T F S

NULL POINTER :-

A pointer that doesn't point to any location.

```
int* ptr = NULL; //
```

↳ mostly used in linkedlist and trees.

↳ NULL pointer can't stored Garbeg values

↳ Lab :-

1) LinkedList is a stack.
2) LinkedList is Queue.
3) LinkedList is a priority Queue.
4) LinkedList is a circular Queue.

↳ Six Function :-

1) insert at start.
2) Delete at start.
3) insert at middle.
4) Delete at middle.
5) insert at end.
6) Delete at end.

1) :- Make a program Menu base using the above six function?

Date 20
M T W T F S

LINKLIST :-

The collection of nodes is called LinkedList.

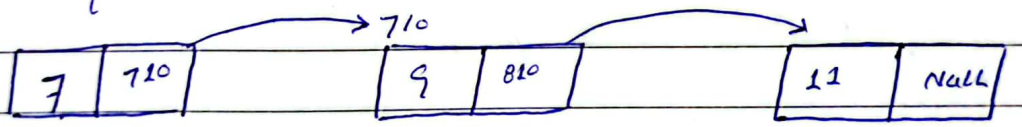
★ Node Contain Two part
i) Data
ii) pointer

2→ LinkedList is dynamic because they can Grow or shrink at Run time.

2→ No memory wastage

SINGLY LINKEDLIST :-

which one node is store the address of other nodes and Tail point to NULL address.



Insertion :-

```
Node* Head = NULL;
Node* Node = new node (10);
```

Insert At Head :-

Date 20
M T W T F S

```
#include <iostream>
using namespace std;
class Node {
 public :
      int data;
      Node* next;
    // constructor
    Node (int data) {
       this -> data = data;
       this -> next = NULL;
       }

       };

void InsertAtHead(Node* &head, int d) {
       // new node create
        Node* temp = new node(d);
         temp -> next = head;
          head = temp ;
       }
int main( ) {

      Node* node1 = new node(10);
       cout << node1 -> data << endl;
       cout << node1 -> next << endl;

       }
```

Date_____20__
M T W T F S

Insert At Tail-

```
Node* temp = NewNode(d);
temp -> next = NULLptr;
if (Head == NULLptr)
{
    Head = NewNode;
}
else
{
    Node* Temp = head;
while (temp -> next != nullptr)
{   temp = temp -> next
}
temp -> next = newNode;
};
```

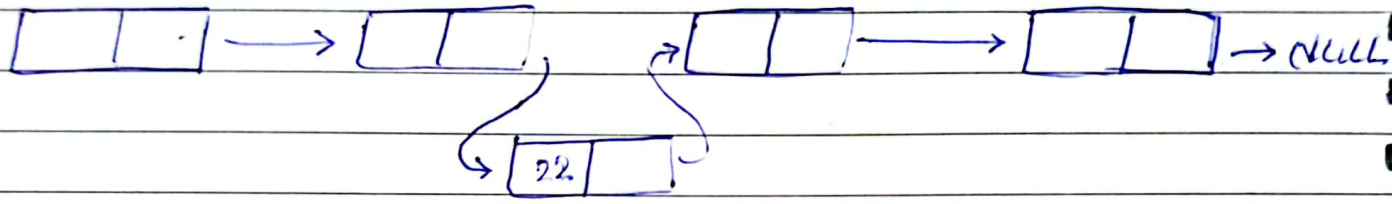
Date 20
M T W T F S

Delete :-

In next page.

Insert At Middle :-

insert 22 in 3rd position




```
void InsertAtposition(Node* &tail, Node* &head, int position,
                                                int d) {
// insert At Start
    if (position == 1) {
        InsertAtStart(head, d);
        return;
    }
    Node* Temp = head;
    int cnt = 1;
    while (cnt < position - 1) {
        temp = temp->next;
        cnt++;
    }
// check if last poseton
```

Date 20
M T W T F S

```
if (temp -> next = NULL) {
        insertAtTail (tail, d);
        return;
}
Node* NodeToInsert = newNode(d);
NodeToInsert -> next = Temp -> next;
Temp -> next = NodeToInsert -> next;
}
```

→ **Traverse:-**

Traverse meaning to visiting every element of it.

```
void print (Node* &head) {
        Node* temp = head;
        while (temp != NULL) {
                cout << temp -> data << " " << endl;
                temp = temp -> next;
        }
        cout << endl;
}
```

Date_____20___
M T W T F S

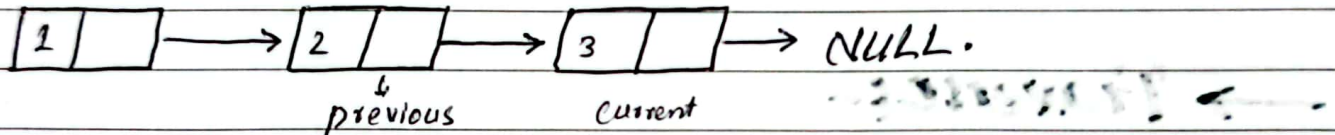
Delete :-

To delete a a element from linked List.
There are three possibilities:-

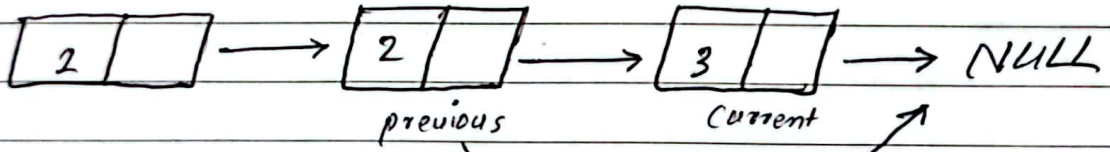
1) Delete from Tail :-
2) Delete from Middle / Any position.
3) Delete from first position.

1:-

Delete from End / Tail :-



So if i want to delete the last So the previous store the current address and then directly point to null.



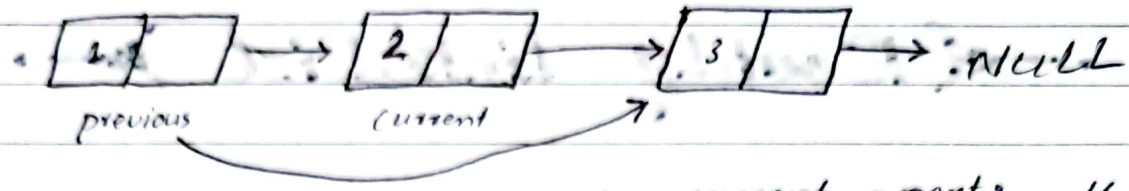
previous → next = current → next; // code.

2:-

Delete from Middle :-

If delete 2 its current.

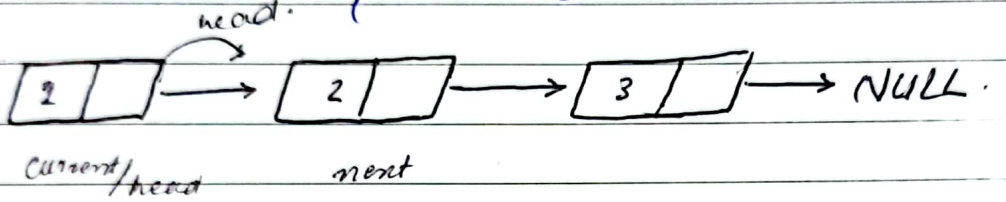
Date_____20__
M T W T F S



previous → next = current → next; // code.

3:-

Delete from Start:-



~~head~~ head = head → next; // code

Date_____20___
M T W T F S

Doubly LinkedList

They allowing the inserting and deletion from both side.

Code Doubly {

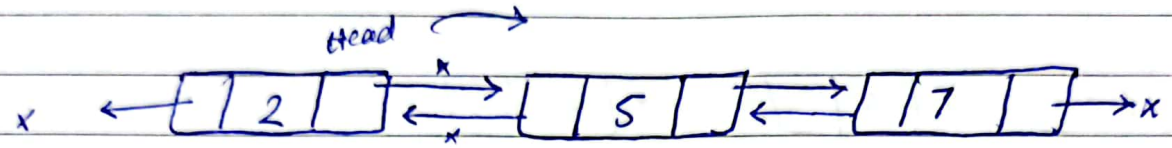
```
public :-
    int d;
    Node* previous;
    Node* next;

Doubly (el) {
This → previous = Null;
This → next = Null;
  This → d = value;
}

void InsertAtStart (Node* &head, value) {
Node* temp = head;
  temp → next = head;
    head → previous = temp;
      head = temp;
}
```

Date_____20___
M T W T F S

2→ Delete from doubly LinkedList:-



if I want to Delete 2
These steps are be followed

1. temp → next → prev = NULL;
2. temp → next = NULL;
3. head = temp → next;
4. Memory free //deconstructor call;

Destractor To Free Memory:-

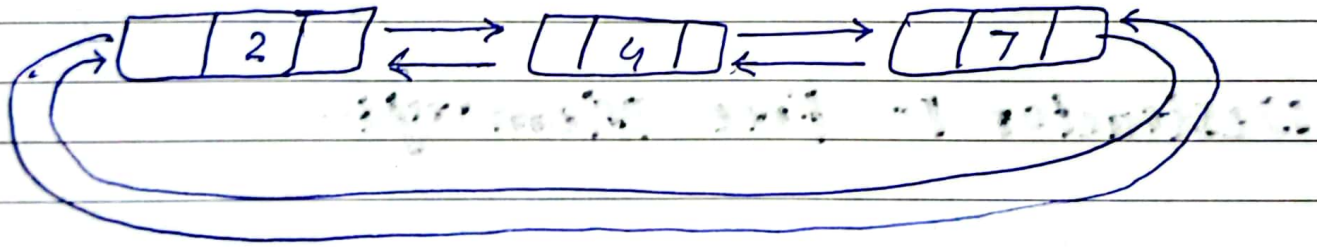
```
~Node() {
    int val = This → data;
    if (next != NULL) {
        delete next;
        next = NULL;
    }
    cout << "Memory is free" << val << endl;
}
```

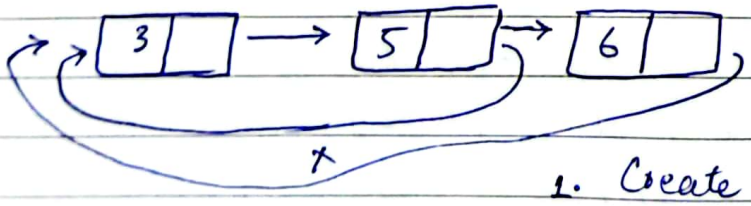
Date_____20__
M T W T F S

Singular Circular LinkedList :-

In circular linkedlist the Tail next is point to head.

Doubly Circular LinkedList :-





1. Create Node
2. temp → next = current → next;
3. current → next = temp;

Date _____ 20__
M T W T F S

2→ In insertion they have the following case :-

1. If Tail is Empty :-

```
if (tail == nULL) {          // if Tail is
                             // Empty
    Node* temp = new Node(d);
    tail = temp;
    temp → next = temp;
}
```

90
40
53
30
30

2. If list is non empty

```
Node {
    Node* curr = tail;
    curr → next = head;
    tail → next = curr;
    tail = curr;
};
```

Date_____20__
M T W T F S

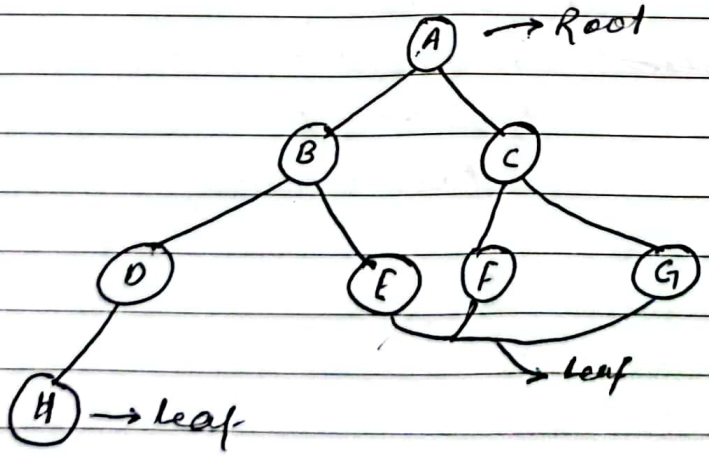
Tree

→ It's a non-linear data structure.
→ It's show a hirearchicals Relation.

Definition :-
It is defined is finite set of nodes such that there is node called roots node.

Node :-
Which store the data and linked to other.

Root :- A node without parent.



Leaf Node :- A node with no childern.

Date_____20__
M T W T F S

Binary tree :-

It's a tree where each node has two childern left and Right

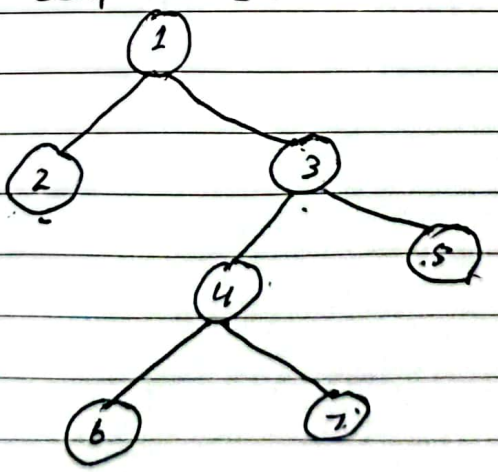
Types of Binary Tree :-

(1) StrickLy Binary tree :-

In which every node contain 2 or 0 node is called strickly Binary tree.

→ $2 \times n - 1$ To find total node

n = non-leaf node.



$2 \times 2 = 4$

in the above Tree every node have 0 or 2 node.

$2 \times n - 1 = 2 \times 4 - 1 = 8 - 1 = 7$

7 is total node.