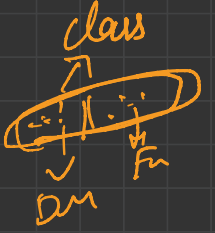
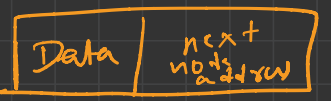
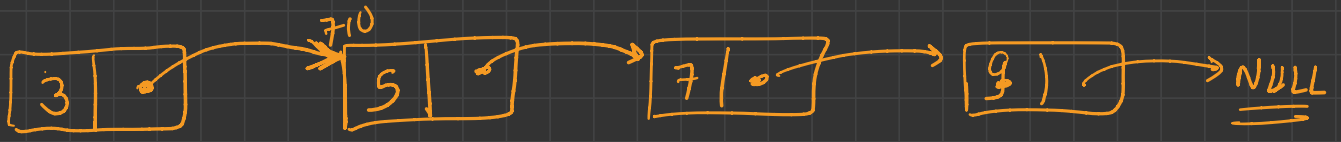
Linked List {Day 1}



→ what? → Linear Data Structure

↳ collection of Nodes





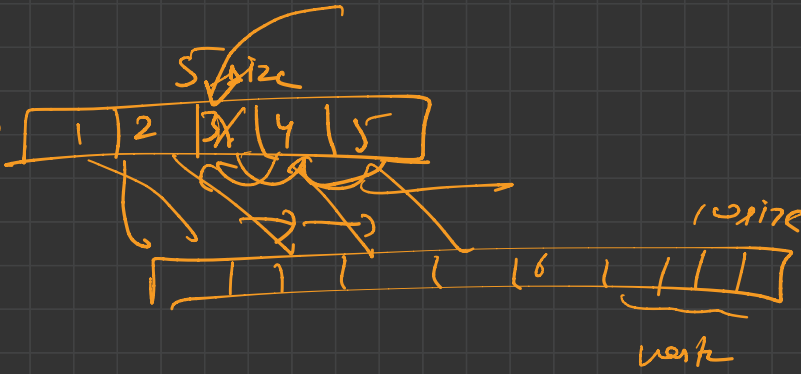
why - ?

arr [10] ⟶ run time - ?

↳ 100 size → ? ∠

Vector → Dynamic Array

vector <int> v(5);



→ new storage

→ values copy

→ Linked List

↳ Dynamic D.S (grow / shrink Run time)

↓

No memory wastage

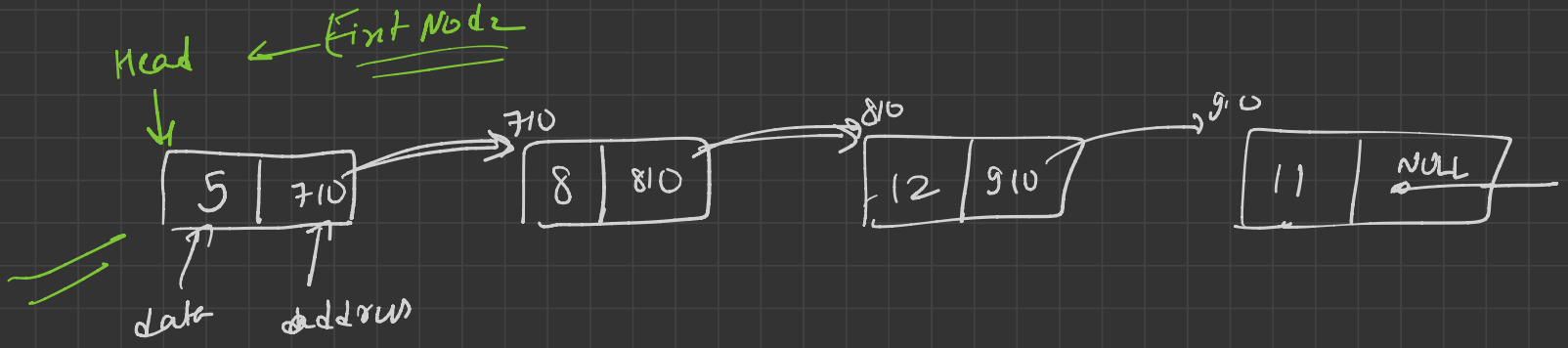
Disadvantage

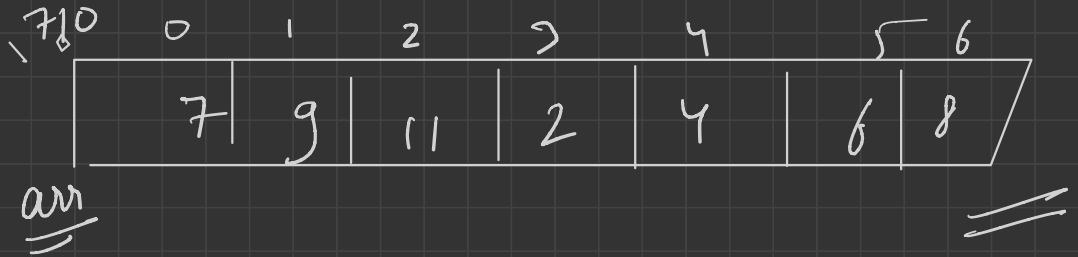
↳ Insertion / Deletion → Easy

↓

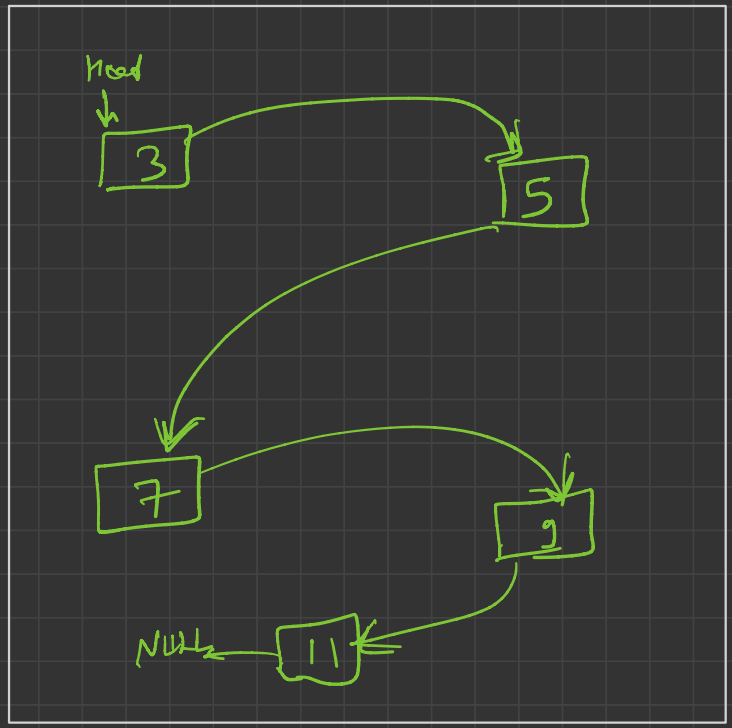
No shift needed

→ Linked List:-


Head
First Node
5
710
8
810
12
910
11
NULL
710
810
910
data
address


710
0
1
2
3
4
5
6
7
9
11
2
4
6
8
arr

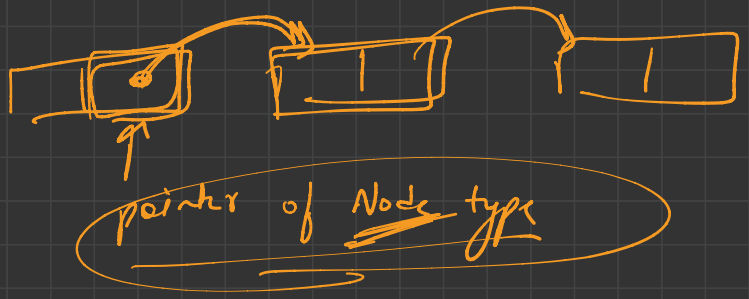
Heap



Types of Linked List:-

- Singly LL
- Doubly LL
- Circular LL
- Circular doubly LL

→ Singly Linked List

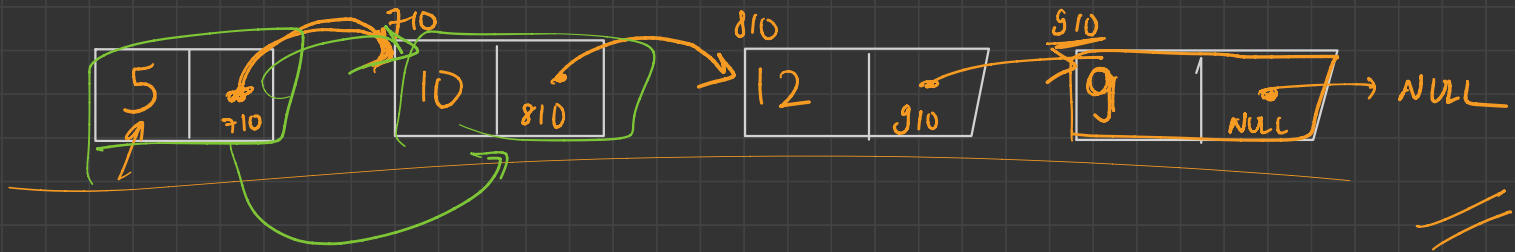


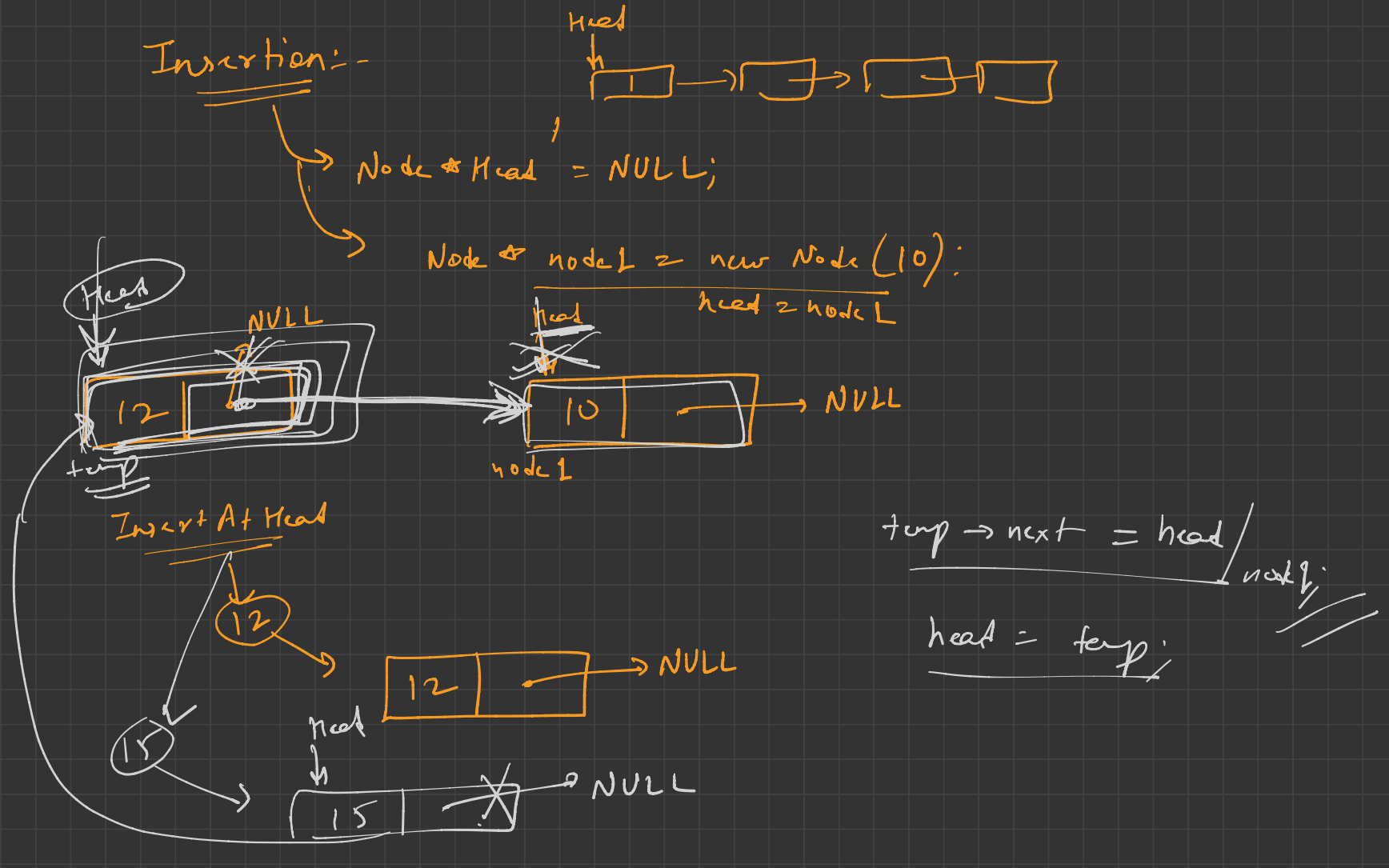
LL
↳ collection of Nodes

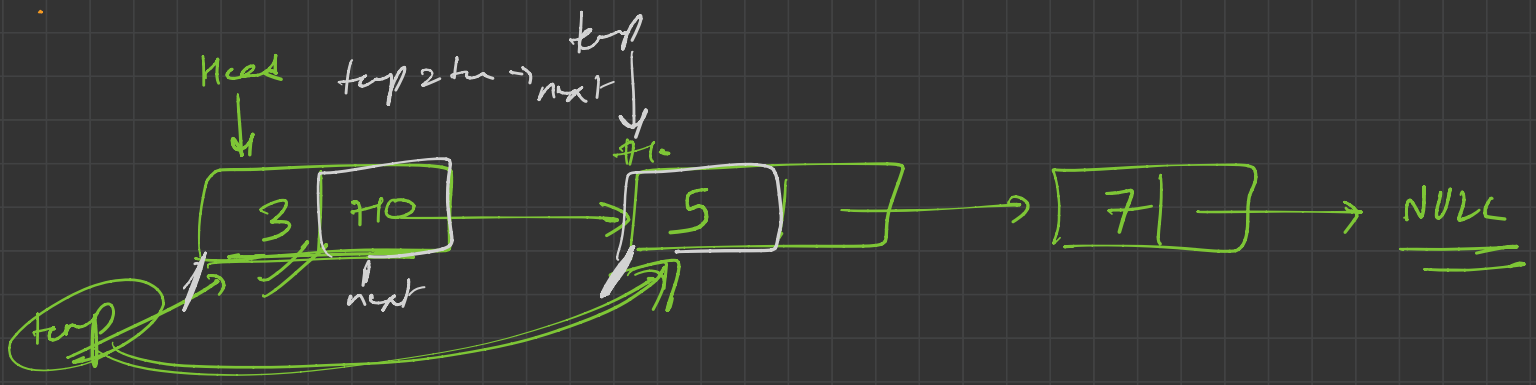
Node
↳ Data
↳ Address of Next Node
↳ Implement ?

```
class LinkedListNode {
public:
    int data;
    LinkedListNode * next;
};
```

Singly Linked List :-




Insertion:-
Head
1
Node * Head = NULL;
Node * node1 = new Node(10);
head = node1
Head
NULL
12
temp
head
10
NULL
node1
Insert At Head
12
NULL
15
head
12
NULL
15
NULL
temp -> next = head;
head = temp;

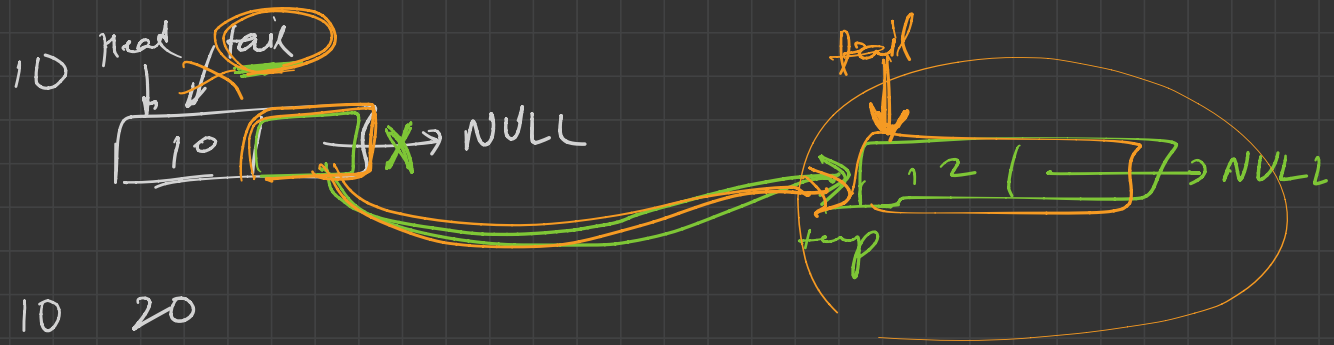


node * temp = head

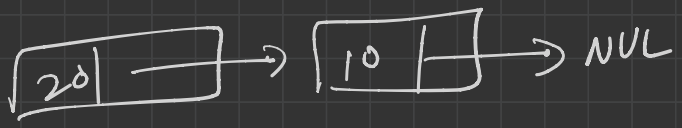
current node → print

temp ko aage badhado

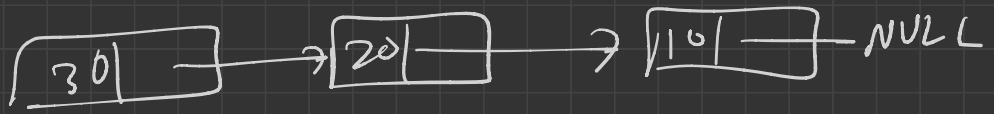
Jab tak temp NULL nhi hota



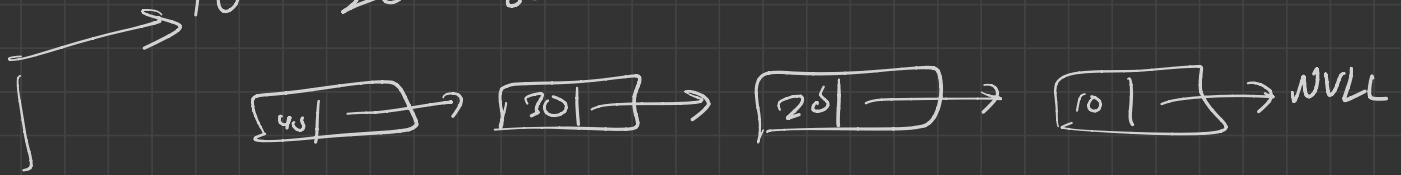
10 20

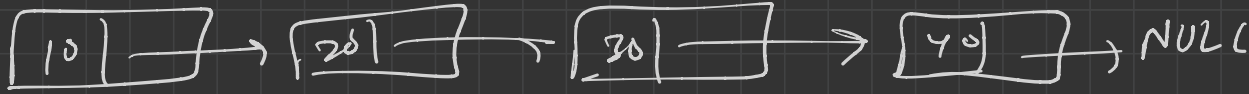


10 20 30



10 20 30 40





→ Insert At Tail -

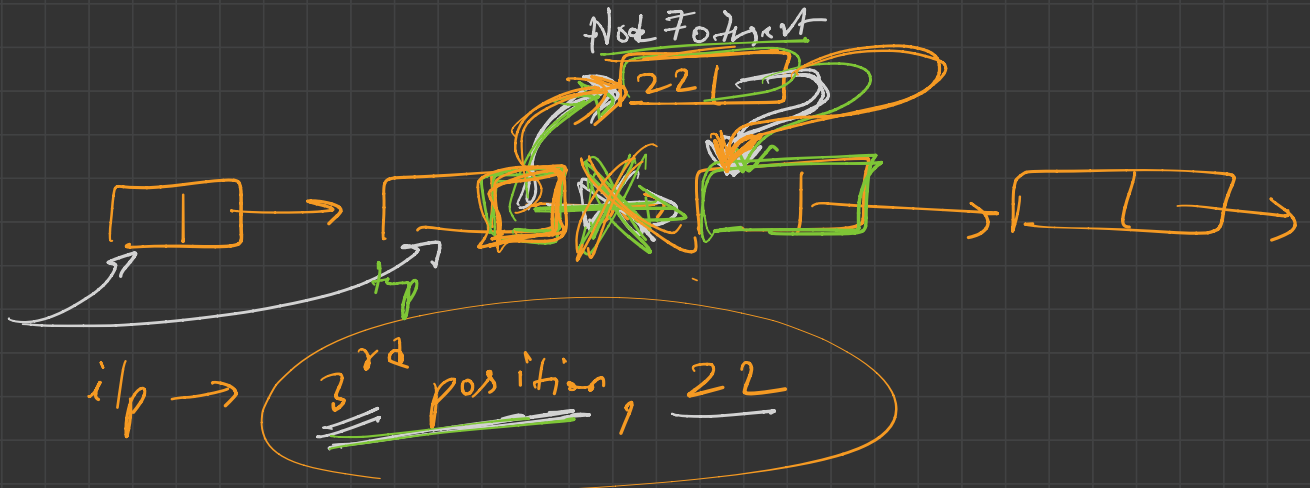
↳ ending node K aage New Node add karna

tail → pointer of Node type

↳ Last node darshayega

Insertion

- Insert at Start / Head
- Insert at End / Tail
- Insert in Middle

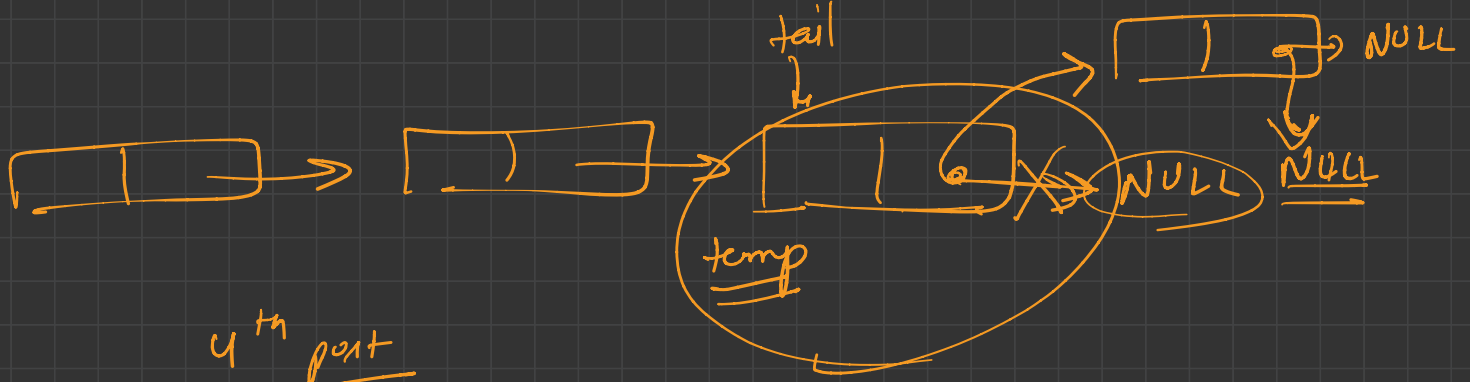


n position

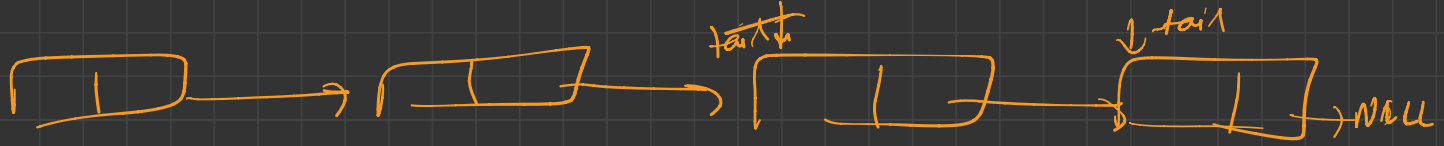
$(n-1)^{th}$ node → traverse

→ NodeToInsert → next = temp → next;

temp → next = NodeToInsert;



4th post

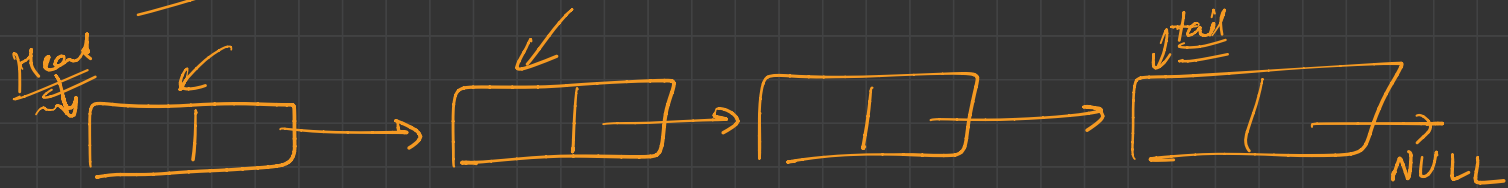


→ Insertion → At any position

→ Traversal

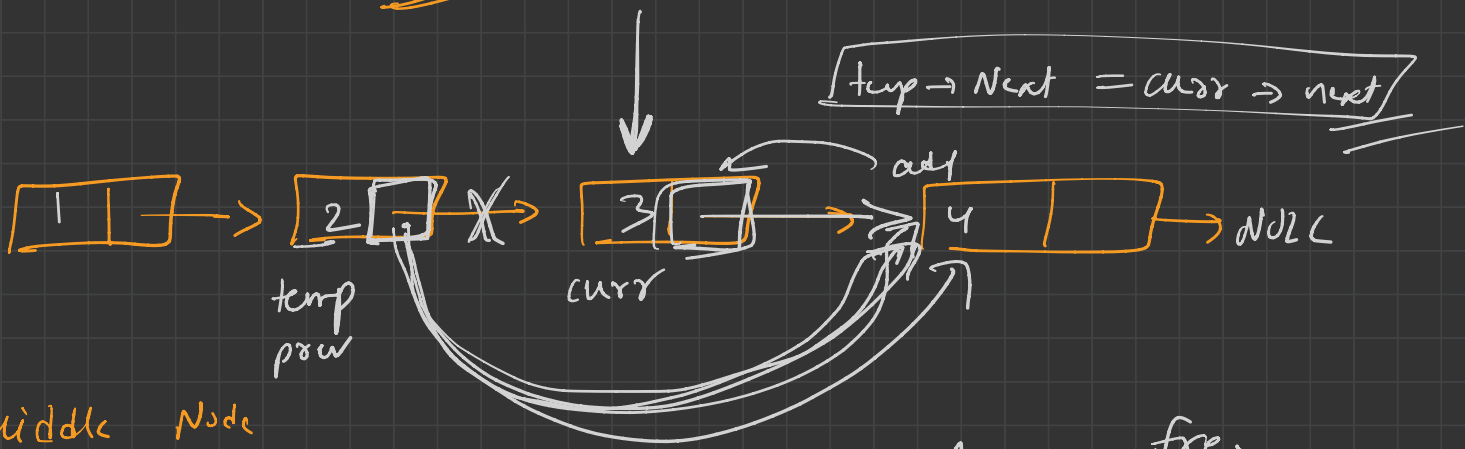
→ Deletion → At any position

→ Deletion:

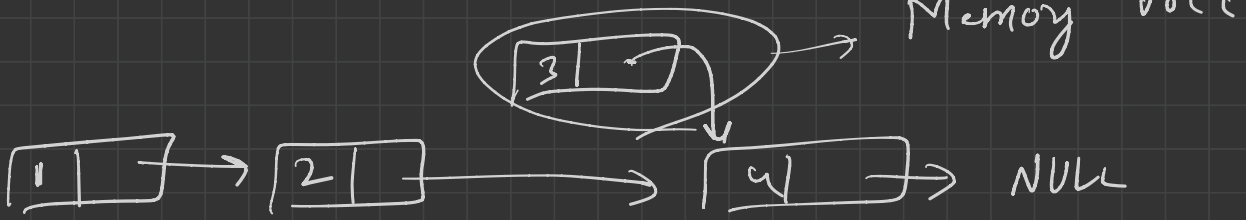


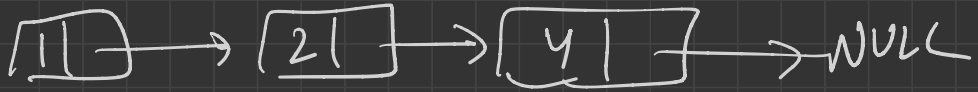
deleteNode (pos) → 1 → ien

deleteNode (value) → H/W

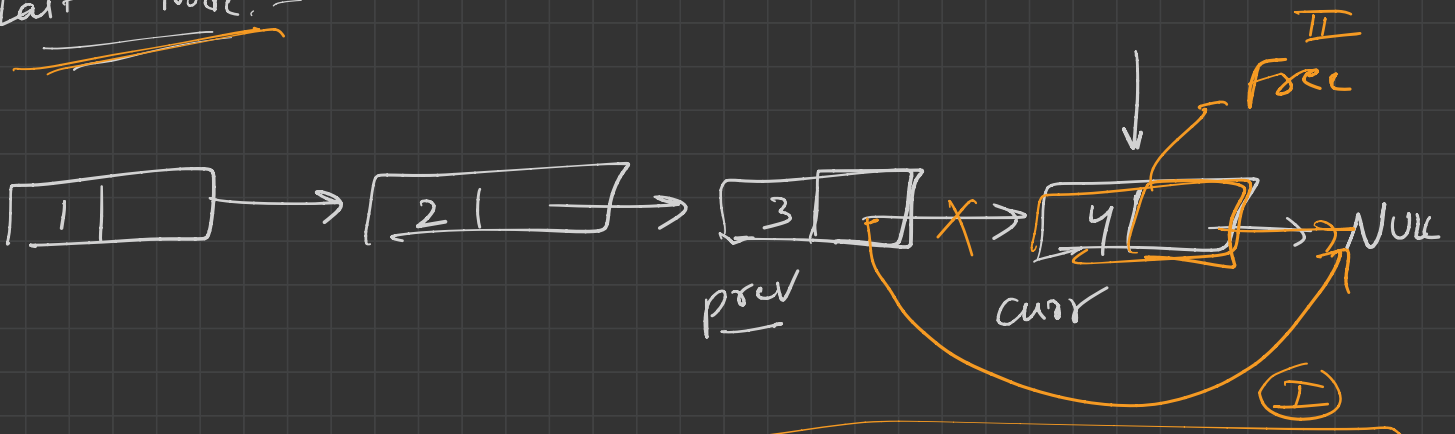


① Middle Node





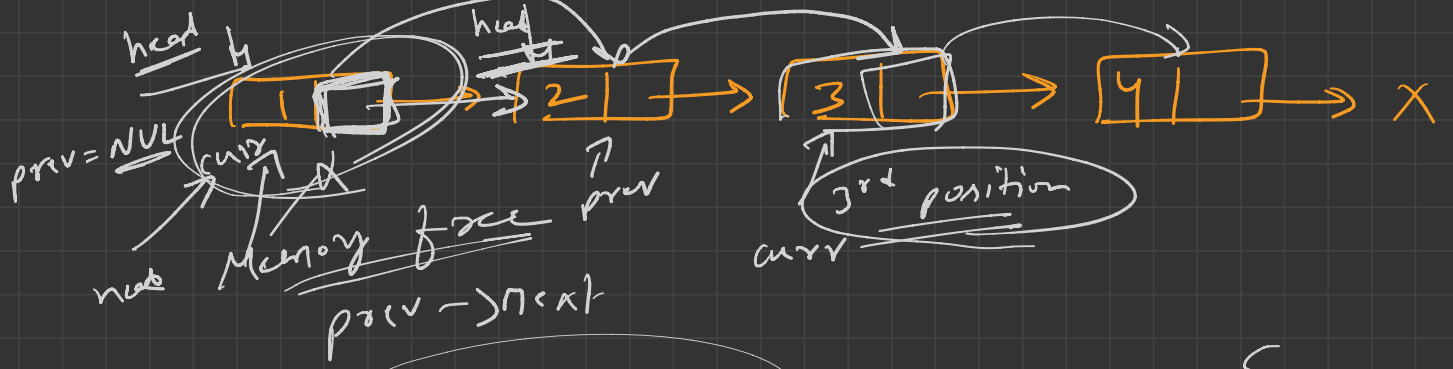
② Last Node :-



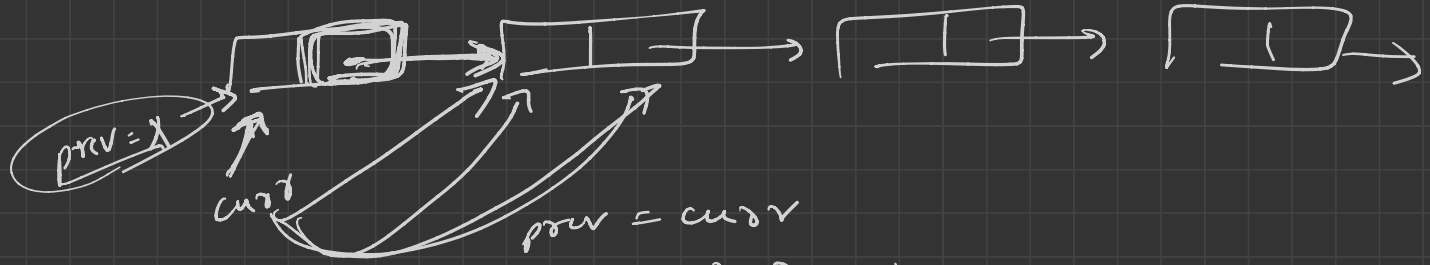
eg `prev → next = curr → Next`

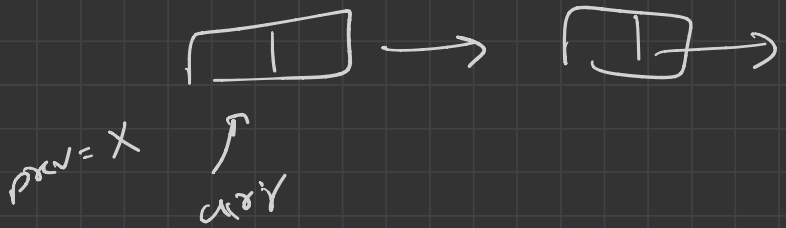
③ First Node → Delete

↳ algo is handle

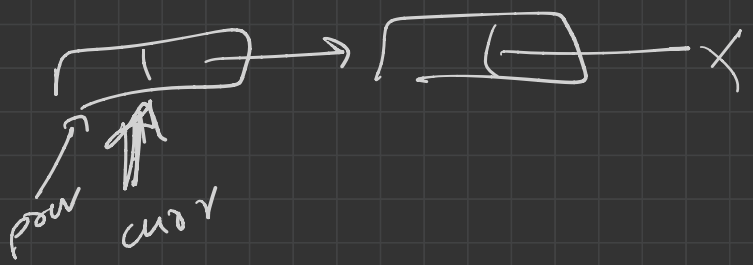


NULL -> next ✗ Error



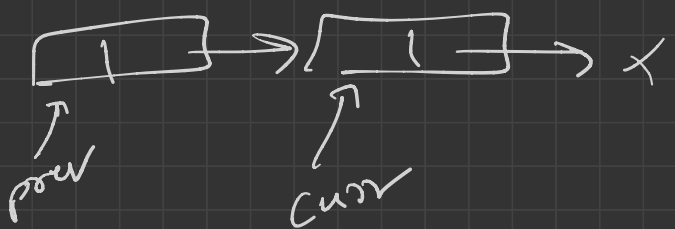


(I)



prev = curr;

(II)



curr = curr → next

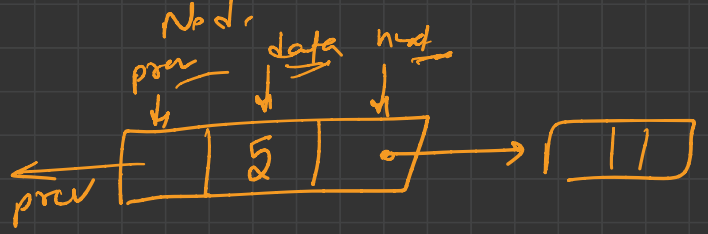
Last Node → Deletion

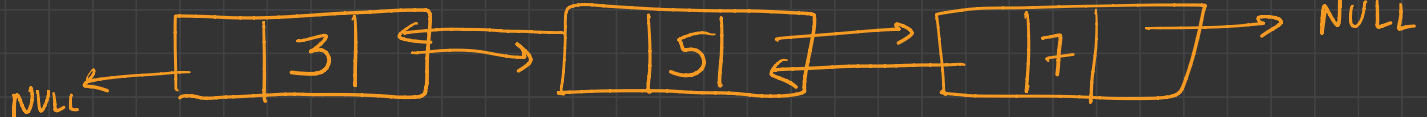
↓

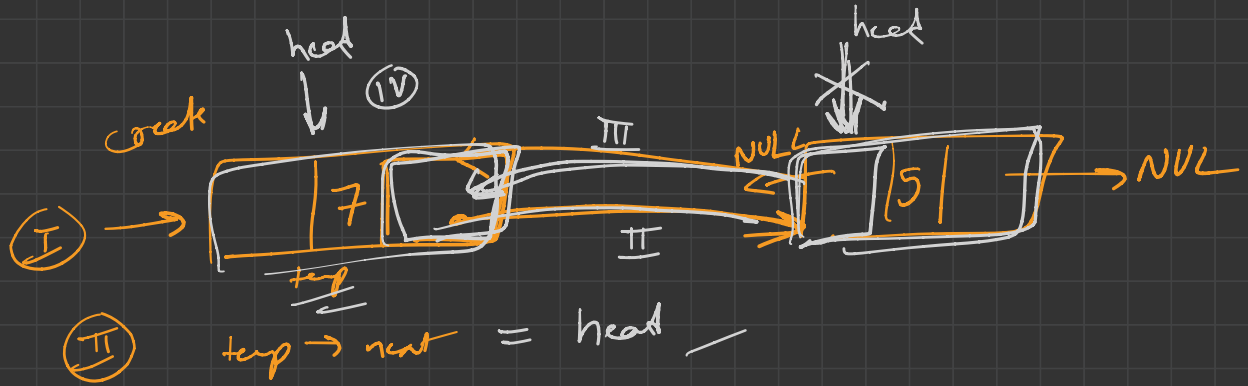
Tail handle Karo →

→ Singly Linked List :-

→ Doubly Linked List:-



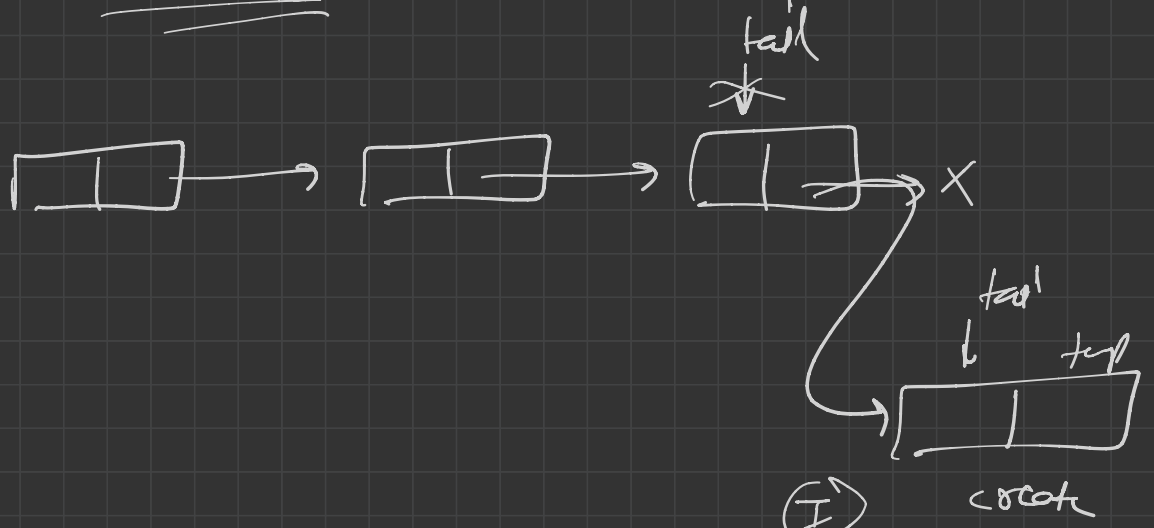




(III) head → prev = temp

(IV) head = temp

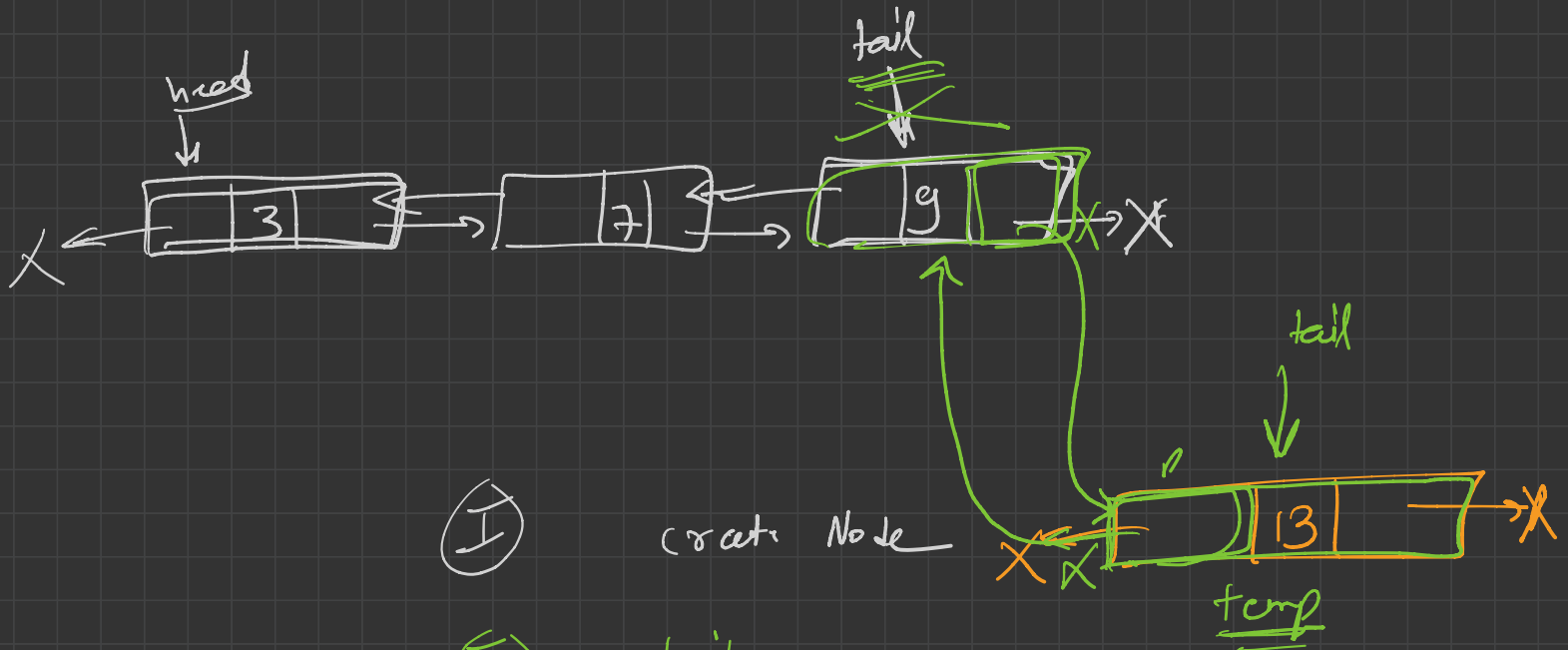
→ Insert at tail



(I) create

(II) tail → next = temp

(III) tail = temp

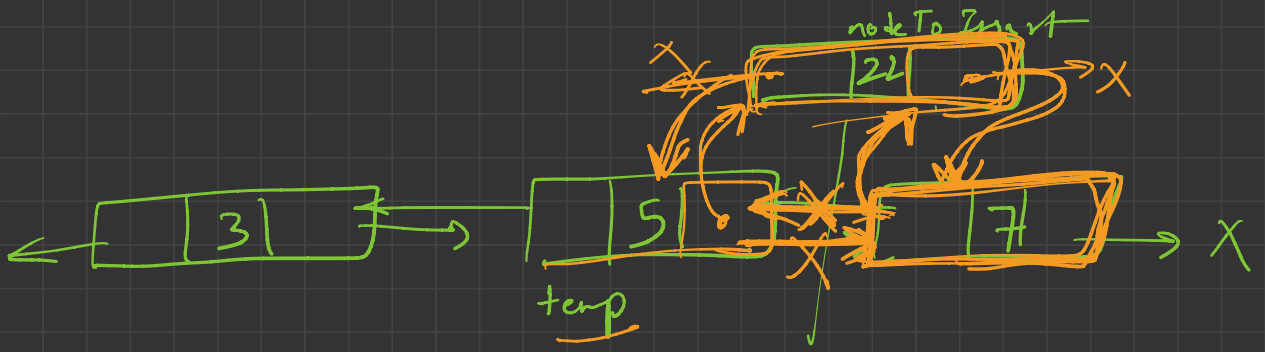


(I) create Node

(II) tail → next = temp

(III) temp → prev = tail

(IV) tail = temp



(I) Create Node

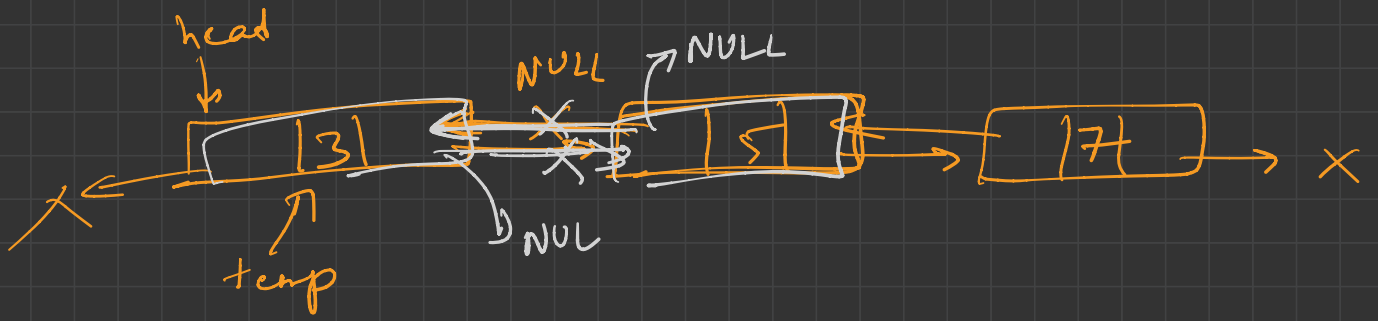
(II) nodeToInsert → next = temp → next

(III) temp → next → prev = nodeToInsert

(IV) temp → next = nodeToInsert

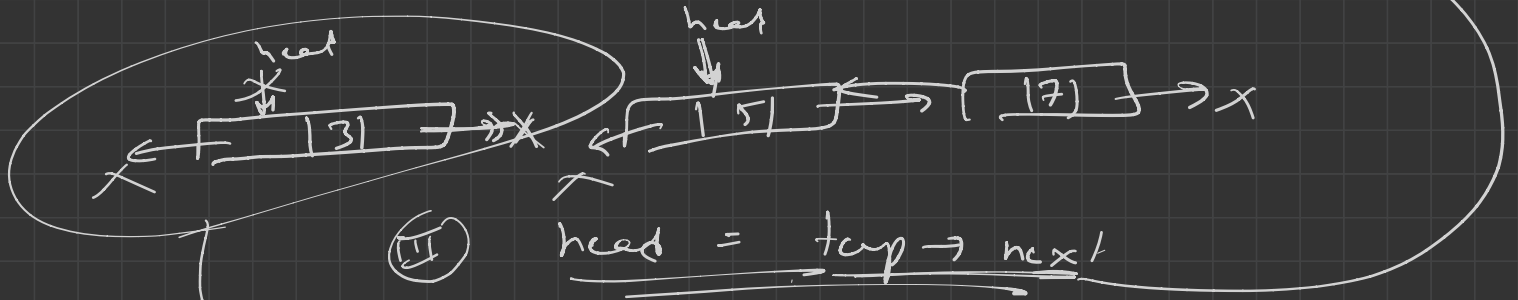
(V) nodeToInsert → prev = temp

DLL → Insertion → Start → Middle
↳ End
↳ Sake
→ Traverse
→ Deletion



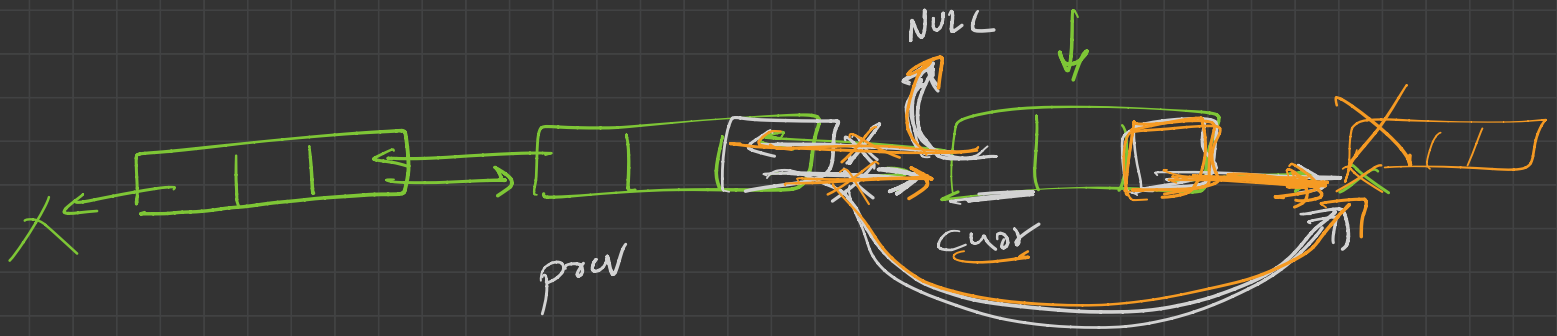
(I) temp → next → prev = NULL

(II) temp → next = NULL



(III) head = temp → next

(IV) Memory free

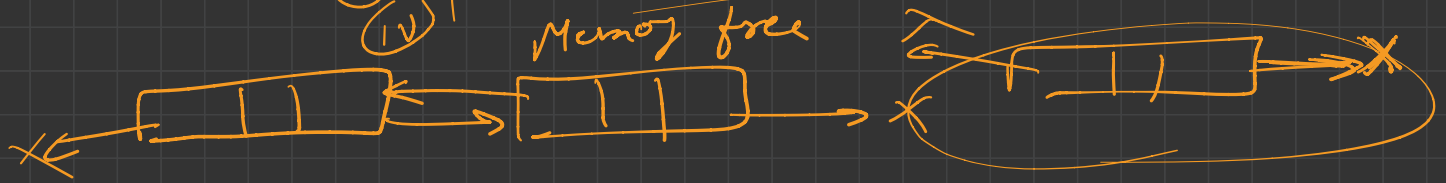


(2) curr → prev = NULL

(1) prev → next = curr → next

(1) curr → next = NULL

(1) Memory free



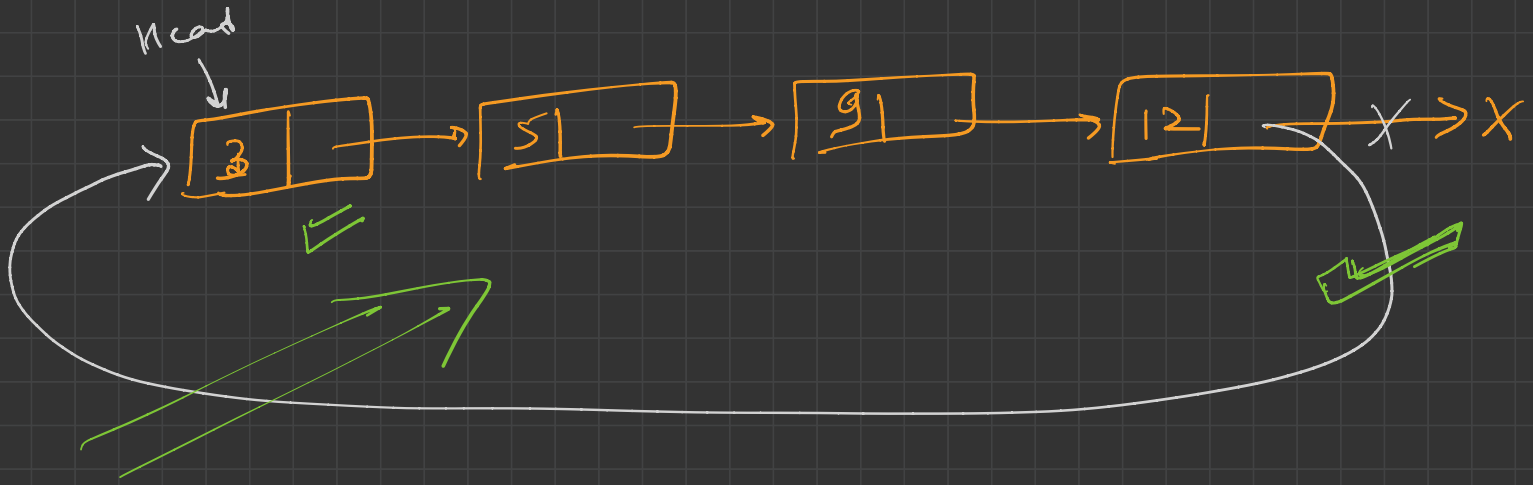
DLL → Insertion → any position

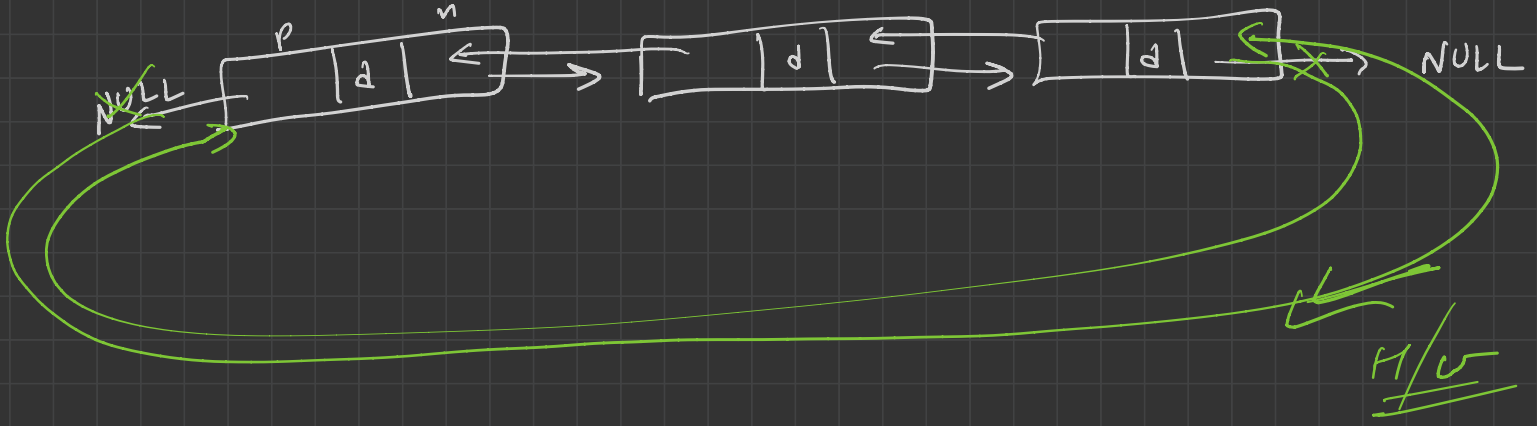
→ Traversal →

→ Deletion → any position

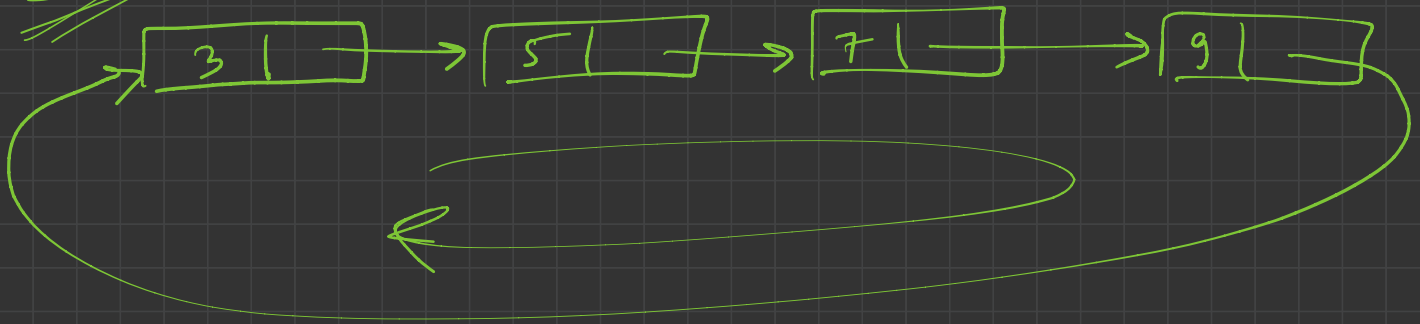
3rd → CLL

Circular Linked List:-


Head
3
5
9
12
X
X


NULL
p
n
d
d
d
NULL
FI/LI

CLL:




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

→ Insertion

→ Traversal

→ Deletion

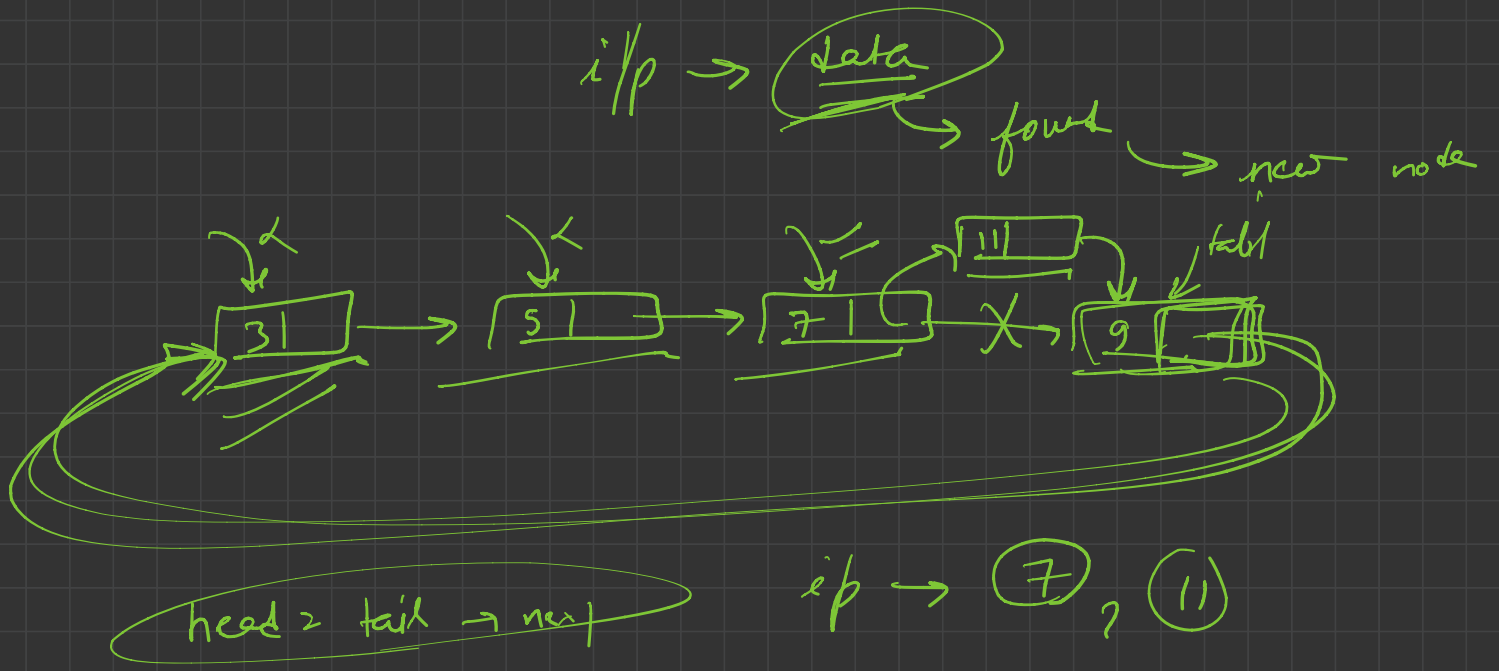
→ Insert AT Head

→ —— Tail

→ Insert at Position

Head ✗

Tail



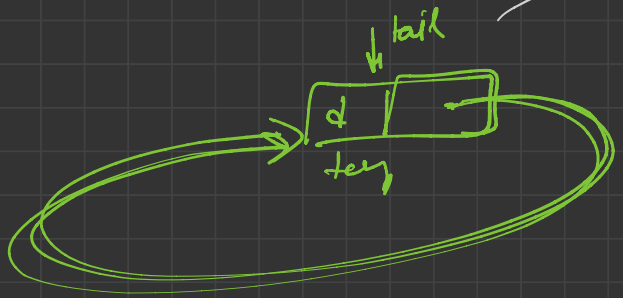
① Empty list

→ tail = NULL

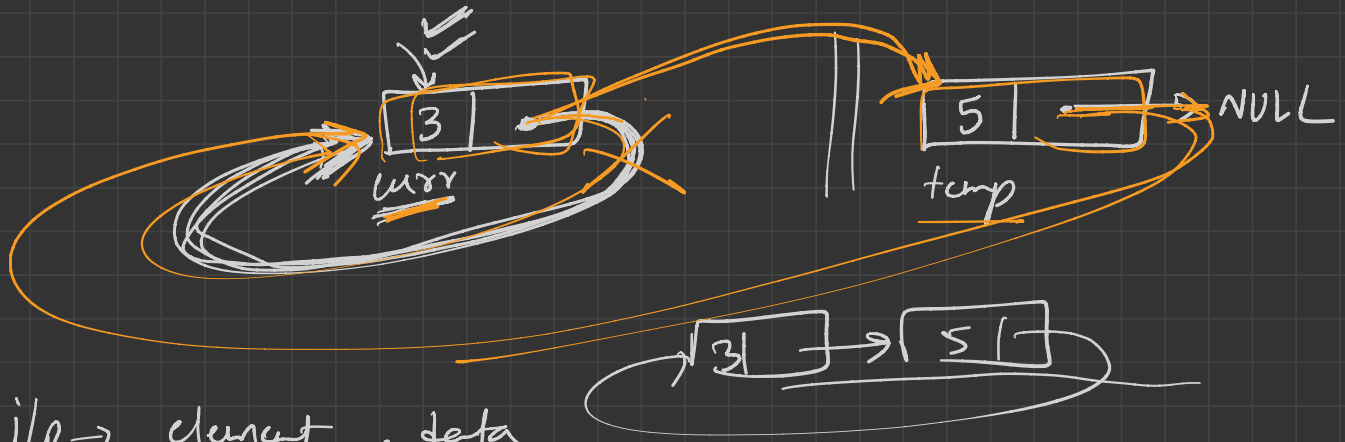
① → New Node

② tail = temp

③ temp → next = temp



② 1 node



i/p → element , data

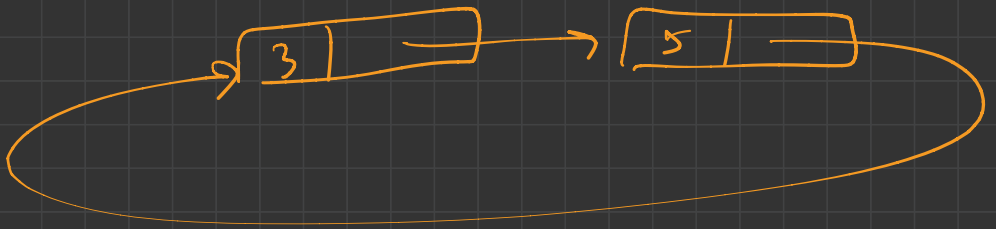
= 3

(I) node create

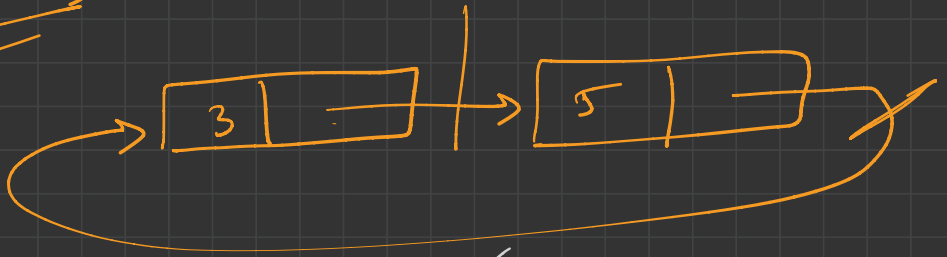
{
(I) forward = curr → next

(II) curr → next → temp

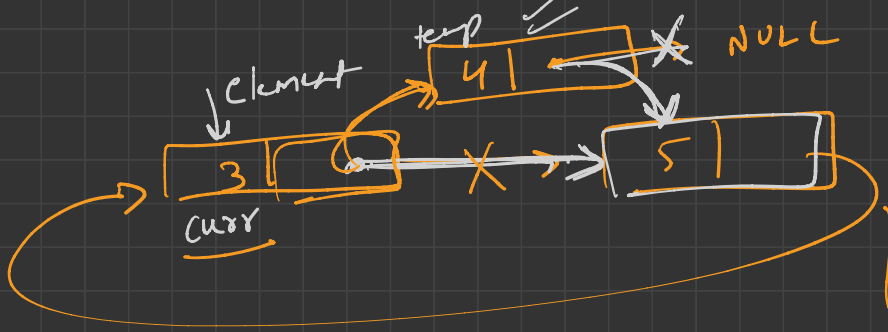
(III) temp → next = forward
}



(2) Node



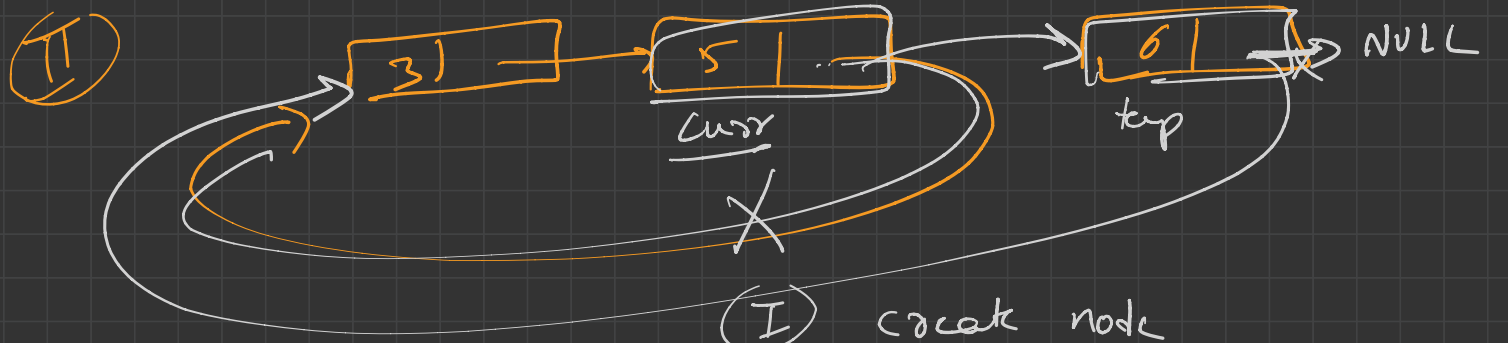
(1)



(I) Node create

(II) temp -> next = curr -> next

(III) curr -> next = temp

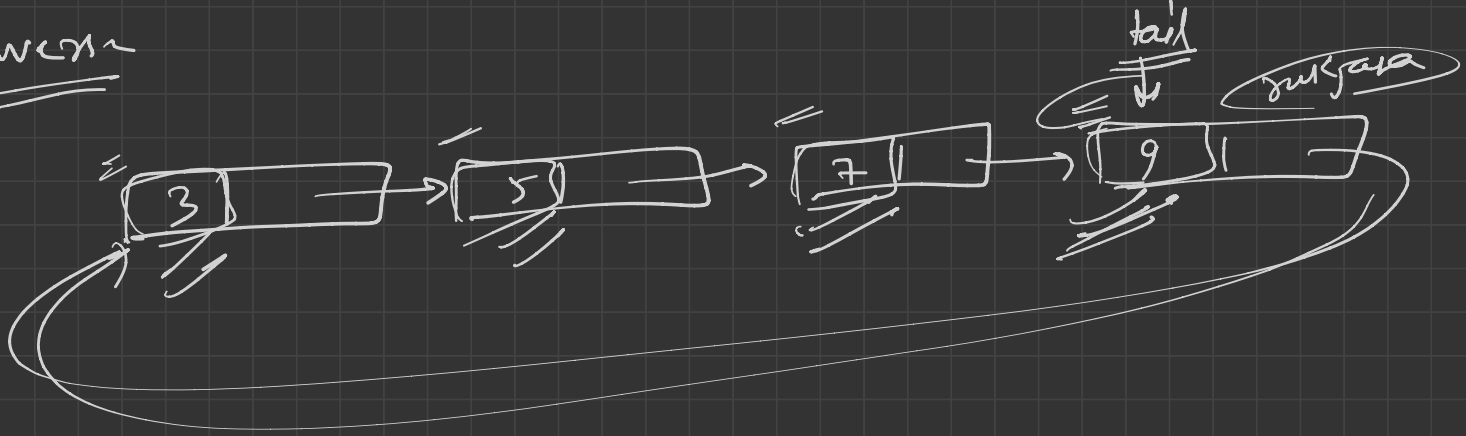


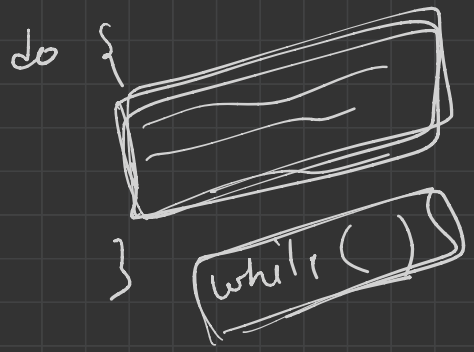
(I) create node

(II) temp → next = curr → next

(III) curr → next = temp

Traversal

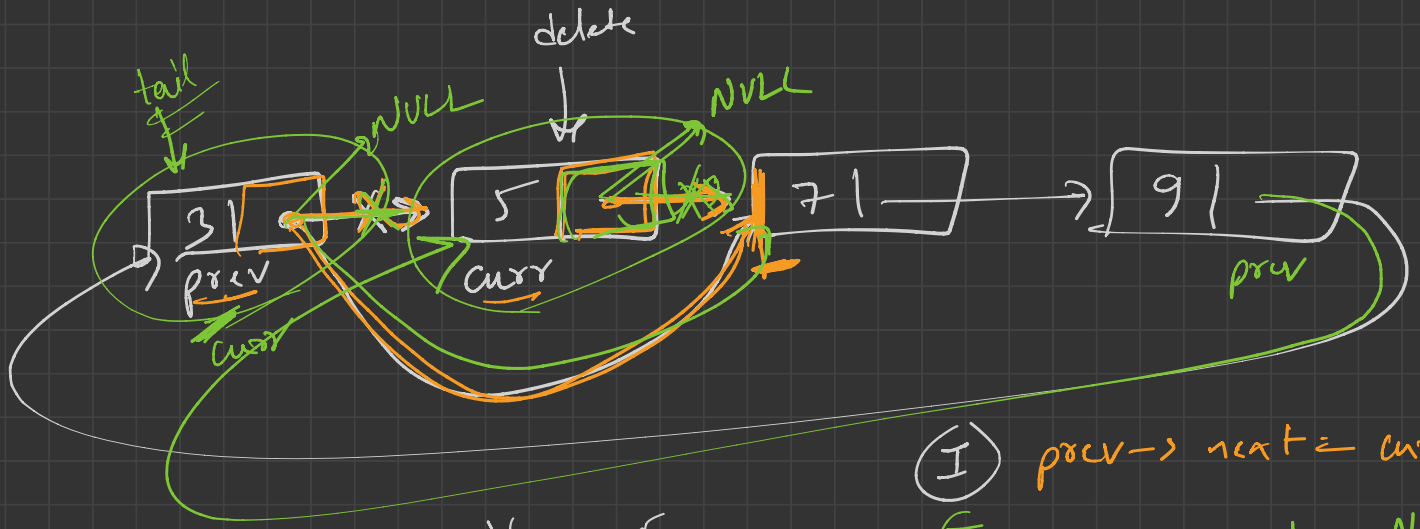




→ atleast 1 baar toh execute hota hai

DLL → Deletion → pos → ___

CLL → value → Node delete ___

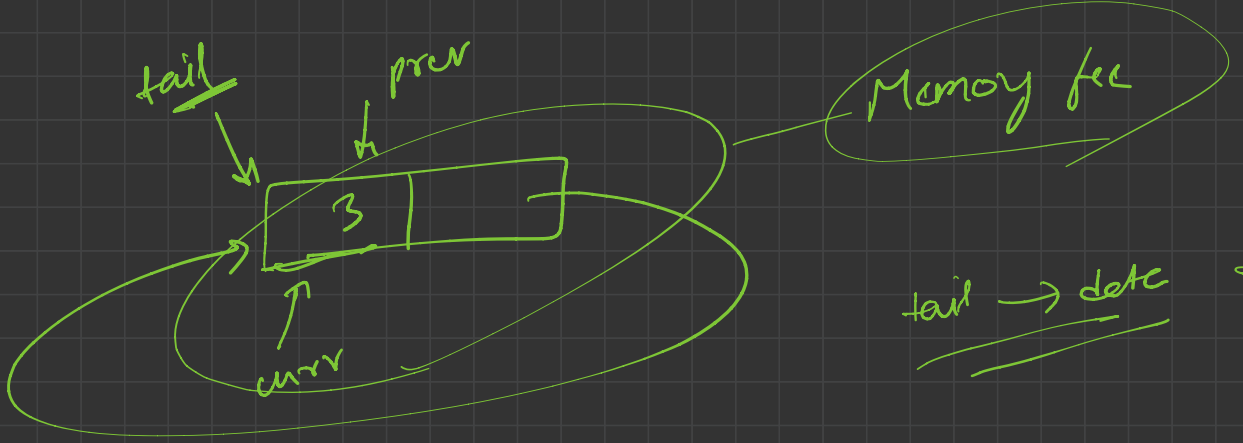


i/p → 5

(I) prev → next = curr → next

(II) curr → next = NULL

(III) Memory free → curr



tail → data

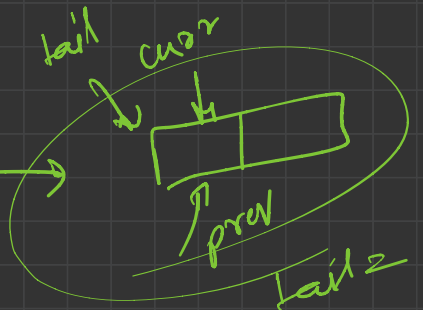
S.F

① Empty list

tail == NULL

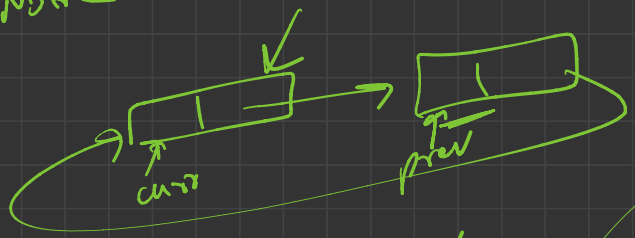
→ Deletion

② 1 Node



→ if (curr == prev)
{ tail = NULL
}

① 2 Node



curr != prev

```
if (tail == curr)
{ tail = prev;
}
```

→ CLL →

Insertion
Traversal
Deletion

SLL → I, T, D

DLL →

CLL →