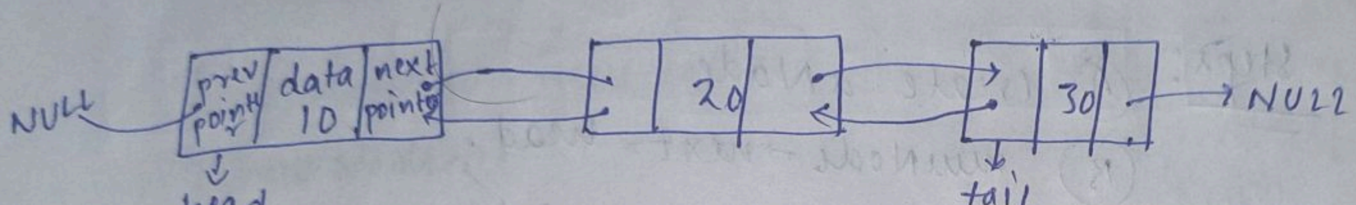
* Doubly Linked List:



```
class Node {
        int data;
        Node* prev;
        Node* next;

}
```

→ Node create of doubly Linked List.

* Print and find the length of all Nodes:

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

        }
    }
```

T.C = O(n) as while loop run till n time

S.C = O(1)

Node* temp = head

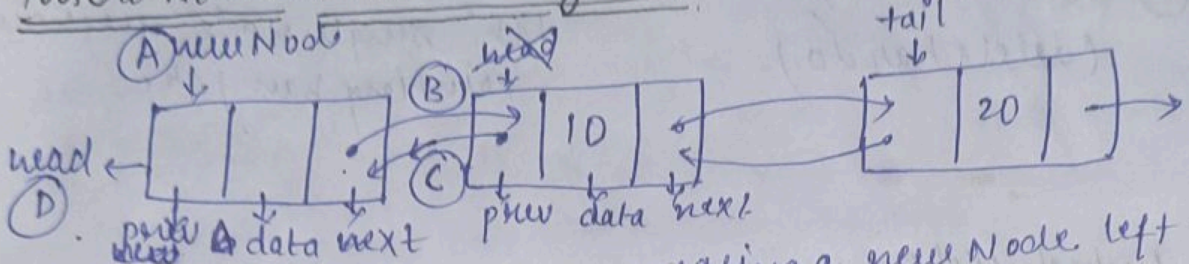
find the length

```
     int findlength(Node* head){                    T.C = O(n)
O(1) ←[ int len = 0;                                S.C = O(1).
SC     Node* temp = head;
       while(temp != NULL){ ]→ T.C = O(n)
            temp = temp→next;
            len++;
       }
       return len;
}
```

* Node:

LL is Hindi Magical Line.

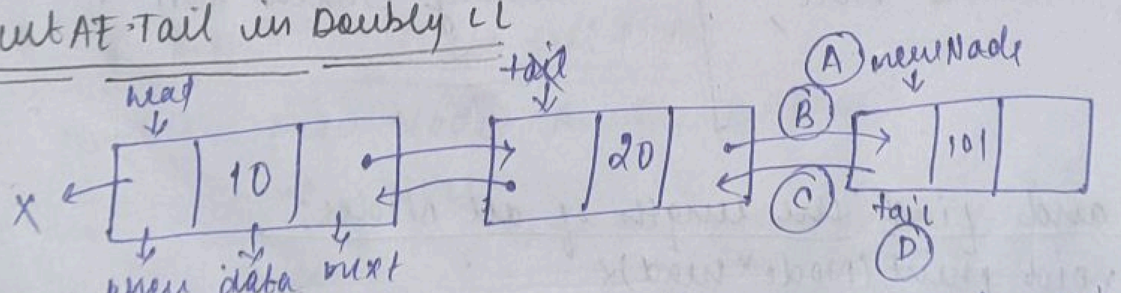
* Insert At head in Doubly LL



(insert at head means creating a new Node left to the curr head node

Steps: Ⓐ Create a Node
Ⓑ newNode→next = head;
Ⓒ head→prev = newNode;
Ⓓ head = newNode (updata head with newNode as newNode now become the head)

* Insert At Tail in Doubly LL



(insert at tail means creating a newnode right to the curr tail node)

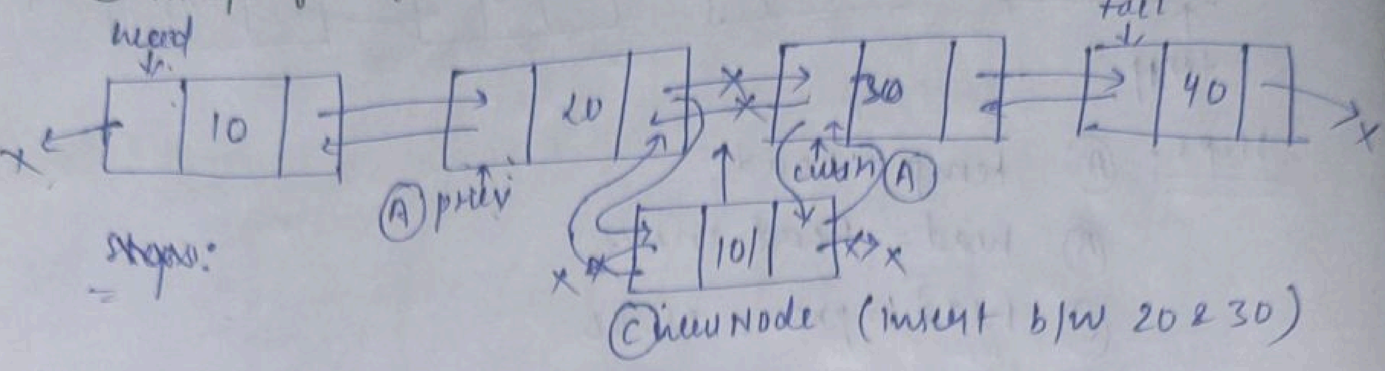
Steps: Ⓐ create a Node
Ⓑ tail→next = newNode

Ⓒ newNode → prev = tail

Ⓓ tail = newNode.

* Insert at position in Double LL:

① At specific position. (i.e 1, 2, 3, 4 - -)



~~Steps:~~

Steps:

Ⓐ find prev & current.

Ⓑ create a newNode

Ⓒ prev → next = newNode

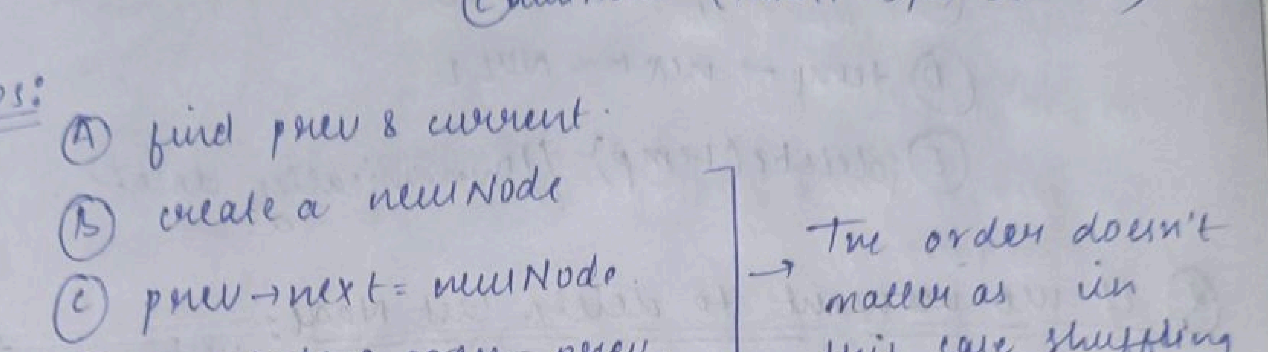
Ⓓ newNode → prev = prev,

Ⓔ curr → prev = newNode

Ⓕ newNode → next = curr.

→ The order doesn't matter as in this case shuffling the order do not results in lost of any ~~part~~ track of LL.

* Insert at position in Double LL (with only prev pointer)



Steps Ⓐ find prevNode

Ⓑ create New-node

Ⓒ $\text{prev}\overset{\text{node}}{\rightarrow}$ next → prev = NewNode,

Ⓓ newNode → next = prevNode → next,

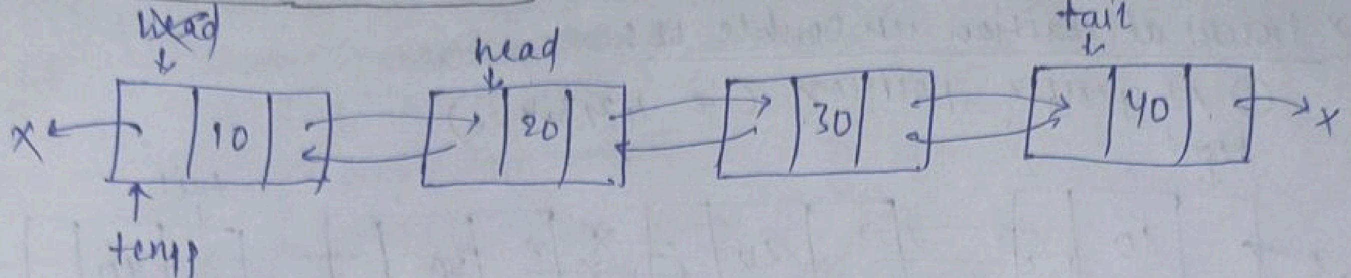
Ⓔ prevNode → next = newNode

Ⓕ newNode → prev = prevNode.

→ Order matter if we write step Ⓒ and Ⓓ at last ~~of~~ the track of right will lost as we don't have curr node

Deletion of Node in DLL:

① If we want to delete first Node:



Steps: Ⓐ temp = head

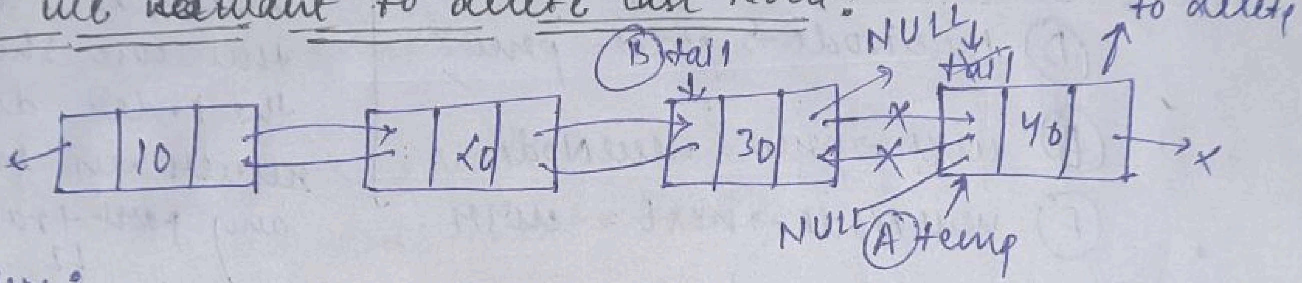
Ⓑ head = head → next

Ⓒ head → prev = NULL

Ⓓ temp → next = NULL

Ⓔ delete(temp) //Dynamically delete.

② If we ~~want~~ want to delete last Node:



Steps:

Ⓐ create temp = tail

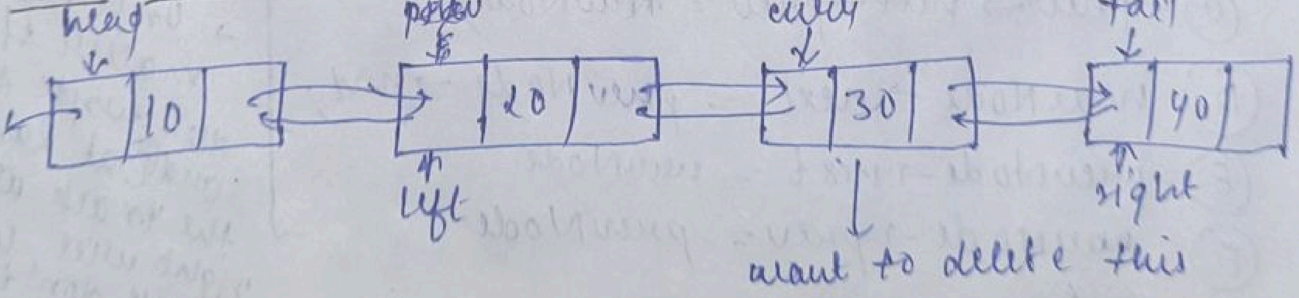
Ⓑ tail = tail → prev.

Ⓒ temp → prev = NULL

Ⓓ tail → next = NULL

Ⓔ delete (temp)

* If we want to delete from middle from any position:



Steps: (A) find left, curr, right Node.

(B) left → next = right.

(C) right → prev = left

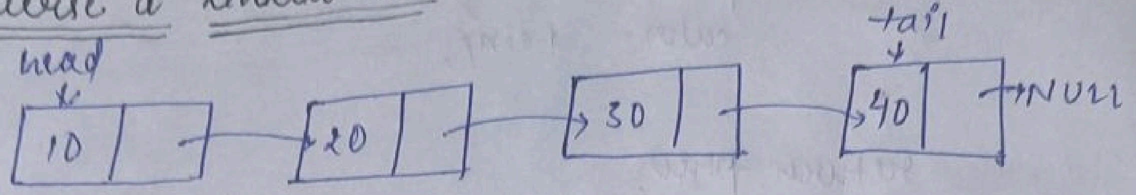
(D) curr → prev = NULL

(E) curr → next = NULL

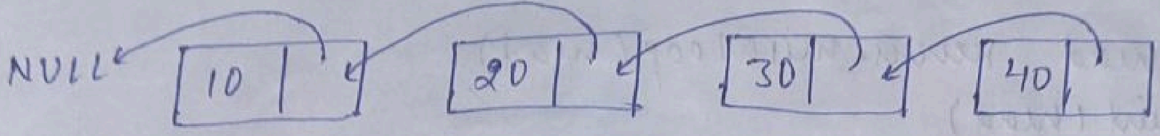
(F) delete curr;

Ques: 1 Reverse a Linked list:

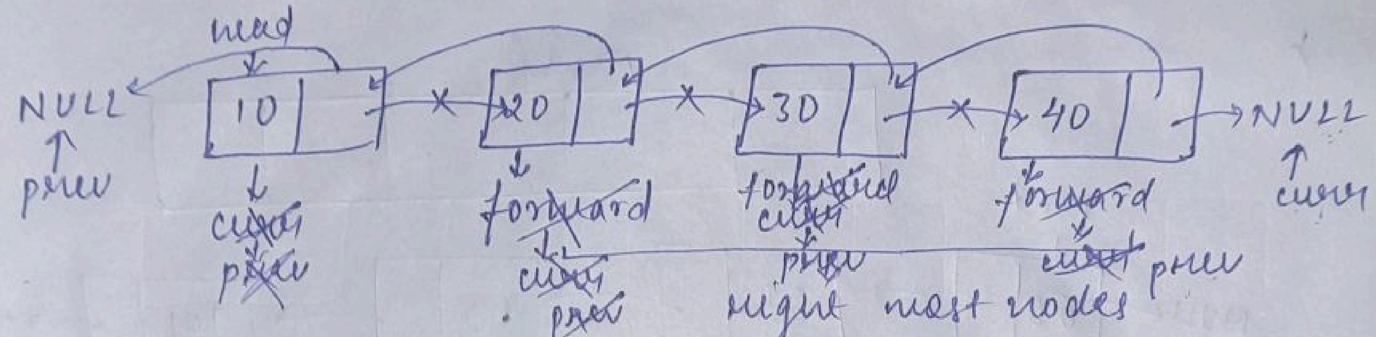
Input :



Output:



→



We can reverse the nodes with recursion:

Steps (A) create prev Node = NULL.

(B) Then curr → next = prev;

(C) create forward = curr → next,

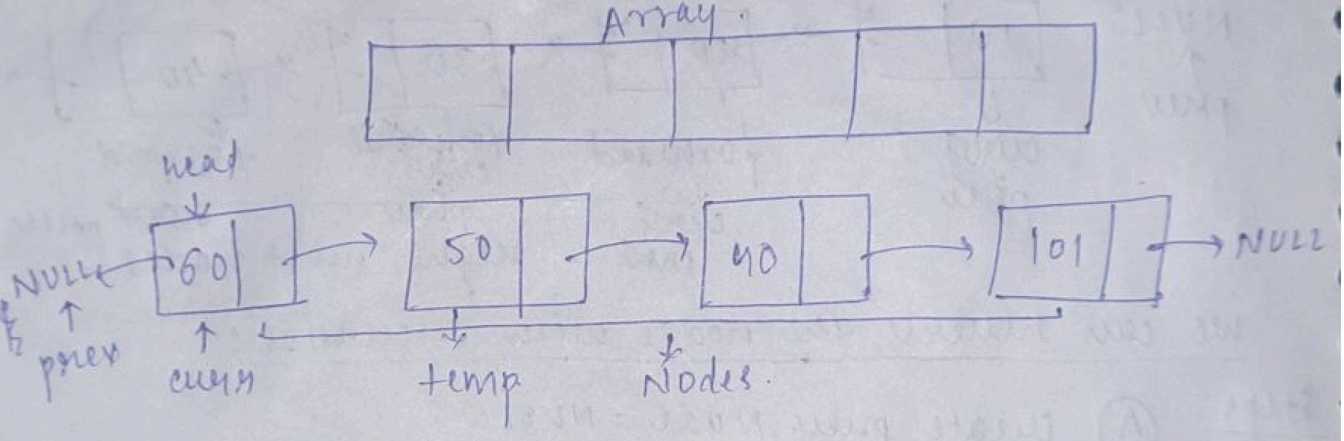
↓ it is necessary to create forward node once we update curr → next = prev we will loose the track of right most nodes which we don't want.

(D) Base case is when curr node become NULL we will return prev node.

We can also ~~do~~ reverse the LL with Loop

Steps.

```
Node * prev = NULL;
Node * curr = head;

while (curr != NULL){

    Node * temp = curr→next;
    ~~head~~ curr → next = prev;
      prev = curr;
      curr = temp;

 }
 return prev;

int main(){

int head = reverseWithLoop(head);
 print(head);

}
```


Array.
head
60
50
40
101
NULL
NULL
prev
curr
temp
Nodes.