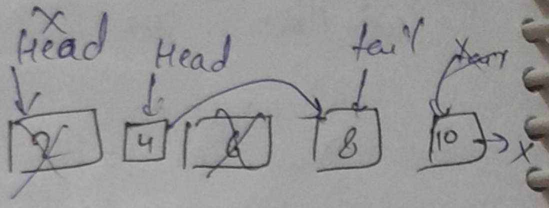
Lecture- Linked List class-2 Date- 04/11/2022

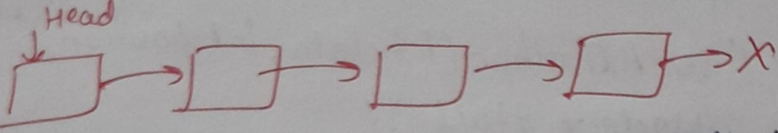
* Magical line → LL is hinde

deletion → delete from tail.

delete from head

delete from middle





① LL empty hai
head == NULL
tail == NULL

② if (pos==1)
delete from head

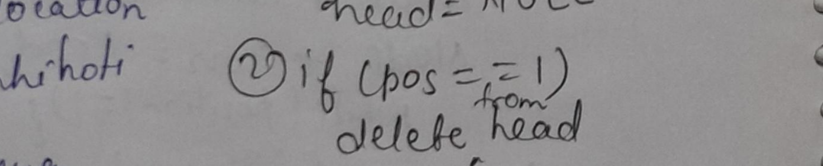
③ if (pos==len)
delete from tail

④ else
middle.

phle node ke deletion ke andar head change ho rha hai

last node ke deletion ke andar ~~tale~~ tail change ho rha hai

Middle node me head tail change nhi ho rha hai



dynamic memory allocation automatic delete nhi hoti

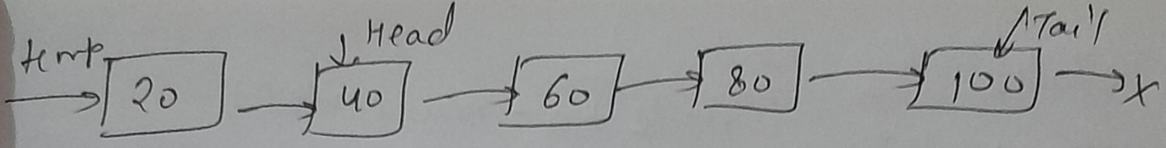
hume delete karna padta hai use delete keyword ka use kar ke

① empty list
head= NULL

② if (pos ==1)
delete from head

③ if (pos==len)
delete from tail

④ else middle

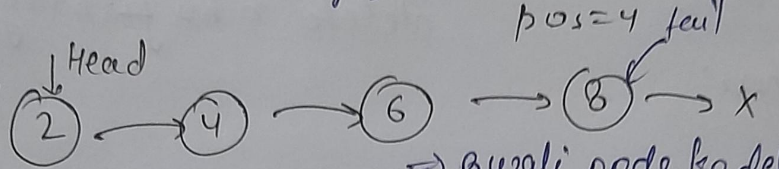


(A) Node * temp = head;

(B) head= temp->next; => the head = head->next;

(C) temp->next = NULL;

(D) delete temp;



I - delete Node
II node k prevnode->NU
III tail update

=> 8 wali node ko delete karna hai

(A) traverse second lastnode -> prev

-> traverse kar ke puchna hai second last element pe.

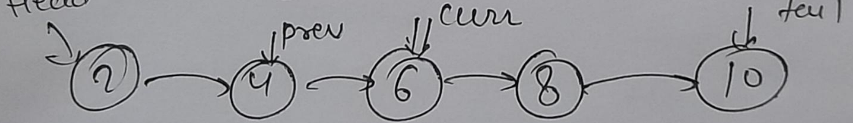
(B) prev->next = NULL

(C) delete tail

(D) tail = prev.

} en dono ka agar order change kargye toh 6 node delete ho jayega jabki hume toh 8 ko delete karna hai.

* Delete element from middle ->



(A) traverse LL for prev/curr

(B) prev->next= curr->next

(C) curr-> next = NULL

(D) delete curr

position=3.

Single element ke ~~head~~ case me head bhi wahi hoga tail bhi whi hoga

Time complexity -> O(n).

```
Single elt -> if (head==tail)
                  node * temp = head
                      delete temp;
                      head=NULL;
                      temp = NULL;
```

Static

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

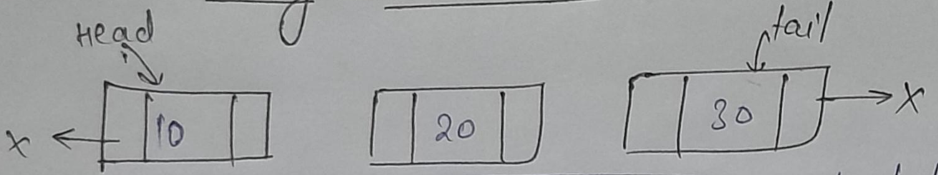
(a.data) se access

kar sakte hai

```
Node * a = new Node();
    (*a).data
        ↓
    a → data.
```

Node ko isolate kar ke delete karo taki aage ke elt delete n ho ja paaye

Doubly Linked List



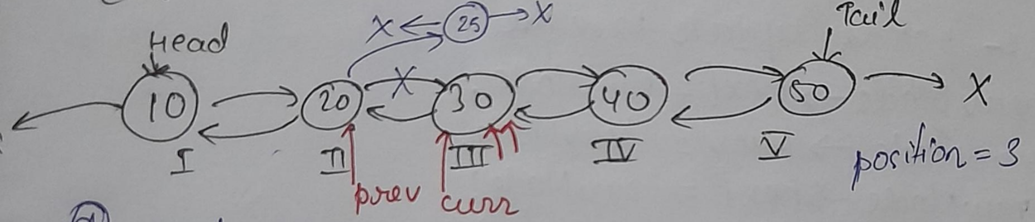
peeche jaane ke liye ~~tail~~ → previous ka use karte hai.

```
#include<iostream>
using namespace std;
class Node{
  public:
  int data;
  Node * prev;
  Node * next;

 Node(){
   this->prev = NULL;
   this->next = NULL;
  }
 Node(int data){
   this->data =data
    this -> prev = NULL;
    this -> next = NULL!
 }
};
~~int main(){~~
void print(Node * head){
  Node * temp = head;
 While( temp != NULL){
   cout << temp->data<<" ";
    temp = temp->next;
```

```
int~~void~~ findLength(Node * &head){
   Node * temp = head;
    int len=0;
  While(temp!=NULL){
     len++;
    temp= temp->next;
  }
  return len;
}

void insertAtHead(Node * &head, Node *
   &tail, int data){
  if(head == NULL){
   Node * newNode = new Node
                  (data);
    head = newNode;
     tail = newNode;
  else{
  Node * newNode = new
          Node(data);
   ~~tail~~ head -> prev = newNode;
   newNode -> next = head;
      head = newNode;
   }
 }
```

```
void insertAtTail(Node * &head, Node * &tail, int data){
    if(head==NULL){
      Node * newNode =new Node(data);
        head= newNode;
         tail= newNode;
    }
  else{
      Node* newNode = new Node (data);
         tail->next = newNode;
          newNode -> prev= ~~NULL~~tail;
               tail= new Node;
     }
  }
```

Insert At Middle.



(a) create node

(B) set prev/curr Node

(c) prev~~ss~~node next = newNode

(d) newNode -> prev = prevNode

(e) newNode -> next = currNode

(f.) currNode -> prev = newNode

```
void insertAtPosition(Node * &head, Node * &tail, int data, int pos)
{
  if(head==NULL){
    Node * newNode=new Node (data);
      head = newNode;
       tail = newNode;
   }
  else{ int length=findLength(head);
       if(position==1){
         insertAtHead(head, tail, data);
elseif(position==len+1){
 insertAtTail(head, tail, data);
 }
 else{
 // insert in middle
 // Step1: create Node
  Node* newNode = newNode(data);
  // step2: set prev & curr pointer
  Node* prev~~Node~~= NULL;
  Node* currNode=head;
   While(position!=1) {
       position--;
       prevNode= currNode;
  currNode~~curr~~ = currNode-> next;
  }
 // step3:- pointers update krne the
      prevNode->next = newNode;
      newNode -> prev = prevNode;
      newNode->next = currNode;
        currNode->prev = newNode;
  }
} }
int main(){
  Node* head = NULL;
  Node* tail =NULL;
 insertAtHead(head, tail, 40);
  insertAtHead(head, tail, 30);
  insertAtHead(head, tail, 20);
  insertAtHead(head, tail, 10)~~00~~;
  print(head);
   cout<<endl;
  insertAtposition(head, tail, 1000, 4)
  print(head);
  return 0;
```

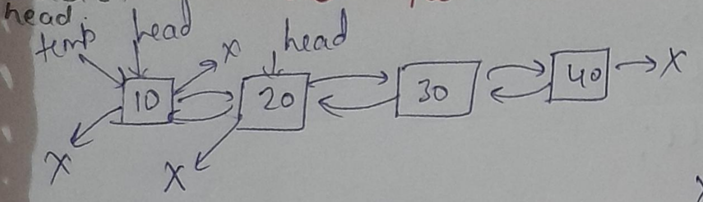
output ->

```
10 -> 20 -> 30 -> 40 ->
10 -> 20 -> 30 -> 1000 -> 40 -
```

Deletion

Head Tail Middle

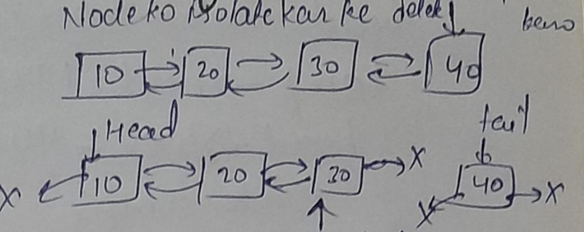
*Delete from head



```
Node * temp = head;
head = head -> next;
temp ->next = NULL;
head ->prev = NULL;
delete temp;
```

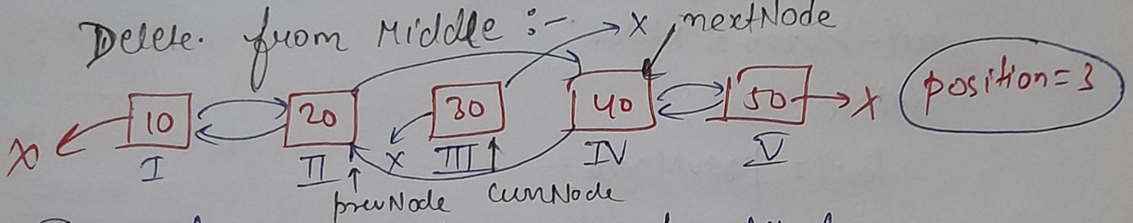
* Delete from tail

Nodeko isolate kar ke delete



```
Node * prevNode = tail-> prev;
prevNode ->next = NULL
tail-> prev = NULL
delete tail
tail = prevNode
```

Delete from Middle :-



(A) set prevNode / currNode / NextNode

(B) * prevNode -> next = nextNode
- * currNode ->prev = NULL
- * currNode -> next = NULL
- * nextNode -> prev = prevNode
- * delete currNode

```
void deleteNode( Node * head, Node *tail, int position) {
        if (head == NULL) {
            // LL is empty
    cout<< " cannot delete because LL is empty "<<endl;
    return;
    }
    int len = findlength(head);
    // one node wala case
    if(head == tail)
    Node * temp = head;
    delete temp;
    head = NULL;    tail=NULL;
```

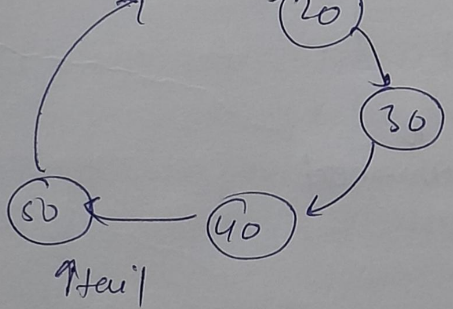
```
if (position ==1){
Node * temp = head;
   head = head->next;
    temp ->next = NULL;
     head ->prev = NULL;
      delete temp;
   }
else if (position== len){
  // delete from tail
  Node * prevNode = tail->prev;
    prevNode ->next=NULL;
     tail->prev = NULL;
      delete tail;
        tail= prevNode;
  }
 else {
       //delete from middle
  // Step1: set prevNode/currNode/nextNode
           Node * prevNode = NULL;
           Node * currNode = ~~Nut~~ head;
           while (position != 1) {
                 position--;
                 prevNode = currNode;
                 currNode = currNode->next;
           }
        Node * nextNode = currNode -> next;
     prevNode -> next = nextNode;
       currNode -> prev = NULL;
        currNode -> next = NULL;
         nextNode -> prev = prevNode;
        delete currNode;
```

```
int main(){
    Node *head= NULL
      Node *tail= NULL
    insertAtHead( head, tail, 40);
    insertAtHead ( head, tail, 30);
    insertAtHead ( head, tail, 20);
    insertAtHead ( head, tail, 10);
    print(head);
      Cout<<endl;
    deleteNode(head, tail, 2);
      print(head);
  return 0;
}
```

10->20->30->40->
10->30->40->

[Doubly Linked List
TC-> O(n)
SC-> O(1)]

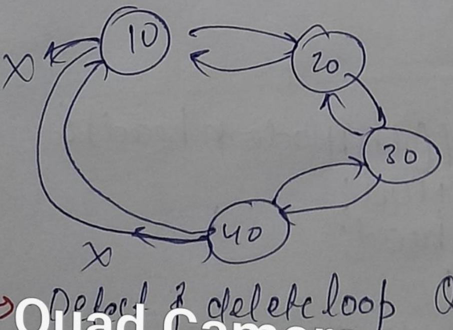
=> Circular Singly Linked List
=> Circular Doubly Linked List.



tail->next=head.

head tail koi bhi ho sakta hai

CDLL



tail-> next=head

head->prev=tail.

Imp-> Detect & delete loop Ques.