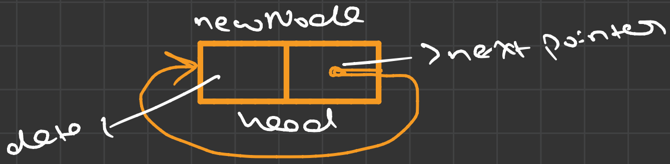
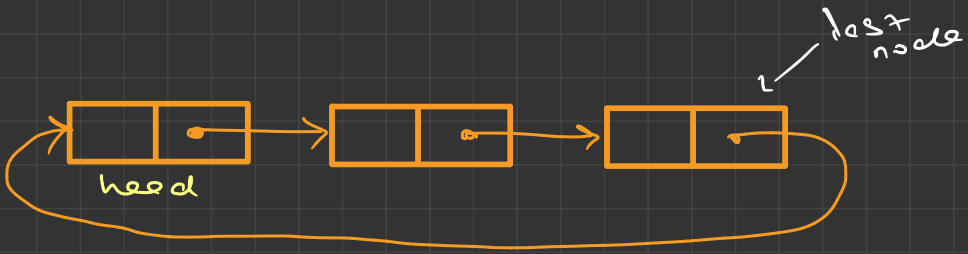
Circular LL → forms a circle

↳ last node points to head

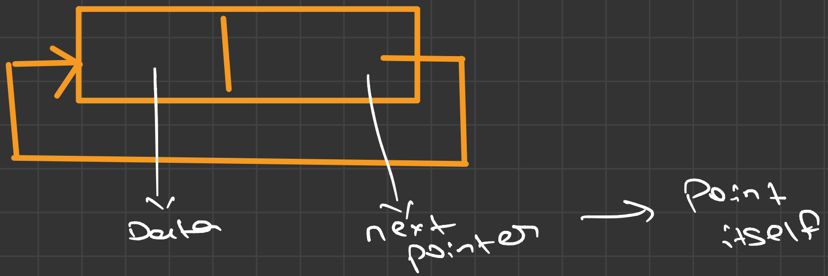
* first node & last node connected to each other

* there is no null at the end





Construct a node



```
class Node{
    public:
    int data;
    Node* next;

    Node(){
        // Point to itself initially for a single node
        this->next = this;
        this->data = 0;
    }

    Node(int d){
        // Point to itself initially for a single node
        this->data = d;
        this->next = this;
    }

    // dtor
    ~Node(){
        cout << "Successfully deleted Node" << endl;
    }
}; '
```

Traversing a LL

```
void print(Node *head){
    if (head == NULL)
    {
        cout << "no node exist" << endl;
        return;
    }

    Node* temp = head;
    do{
        cout << temp->data << " ";
        temp = temp->next;
    }while(temp != head);
}
```

find length of a LL

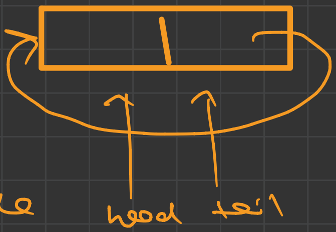
```
int findLen(Node* head){
    if (head == NULL){
        return 0;
    }

    int count = 0;
    Node* temp = head;

    do{
        count++;
        temp = temp->next;
    }while(temp != head);

    return count;
}
```

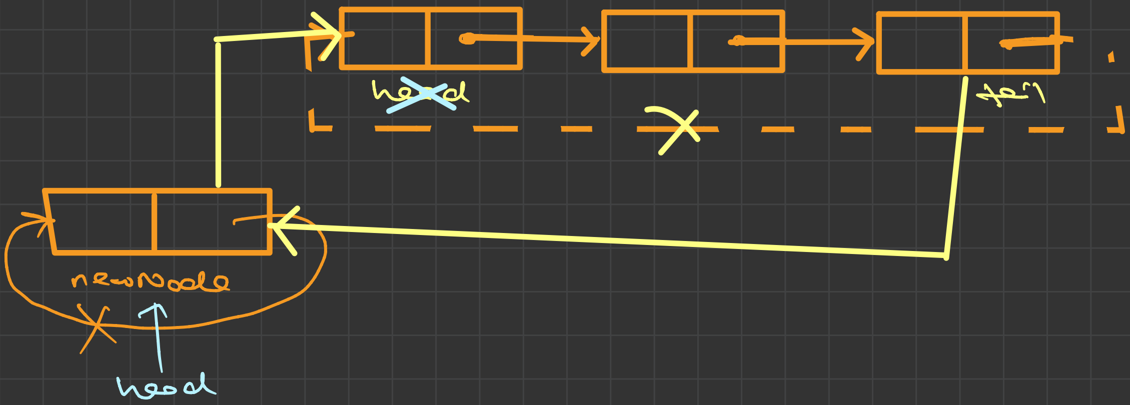
Insert At Head

Edge case

↳ head == null

head = newNode
tail = newnode





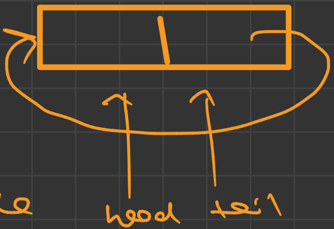
```
void insertAtHead(Node* &head, Node*
&tail, int data){
    if (head == NULL)
    {
        Node* newNode = new Node(data);
        head = newNode;
        tail = newNode;
        return;
    }
    else
    {
        Node* newNode = new Node(data);
        newNode->next = head;
        head = newNode;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

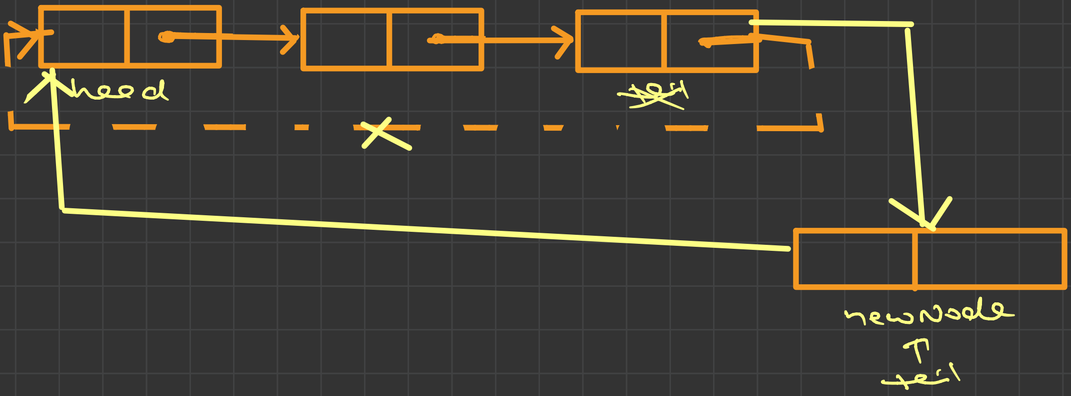
Insert At End

Edge case

↳ head == null

head = newNode
tail = newnode





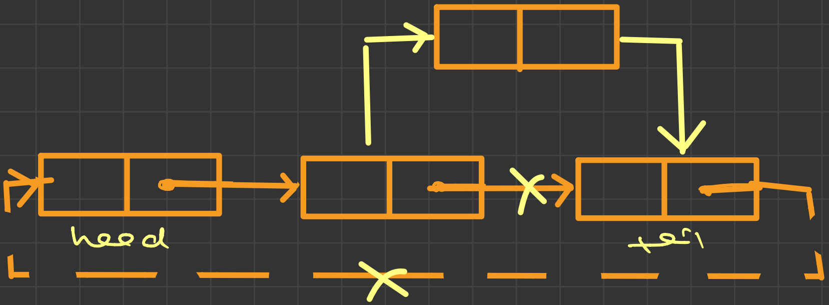
```
void insertAtTail(Node* &head, Node*
&tail, int data){
    if (head == NULL)
    {
        Node* newNode = new Node(data);
        head = newNode;
        tail = newNode;
        return;
    }
    else
    {
        Node* newNode = new Node(data);
        tail->next = newNode;
        tail = newNode;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

Insert At Pos

edge case

- ↳ Pos < 1
- ↳ Pos > len+1
- ↳ head == null

Pos=2



```
void insertAtPosition(Node* &head, Node*
&tail, int data, int position){
    if (head == NULL)
    {
        Node* newNode = new Node(data);
        head = newNode;
        tail = newNode;
        return;
    }

    int len = findLen(head);

    if (position < 1)
    {
        cout << "enter a valid position" << endl;
        return;
    }

    if (position > len+1)
    {
        cout << "enter a valid position" << endl;
        return;
    }


    if (position == 1)
    {
        insertAtHead(head, tail, data);
    }
    else if (position == len+1)
    {
        insertAtTail(head, tail, data);
    }
    else
    {
        Node* newNode = new Node(data);
        Node* currNode = head;
        Node* prevNode = NULL;

        while (position > 1)
        {
            position--;
            prevNode = currNode;
            currNode = currNode->next;
        }

        prevNode->next = newNode;
        newNode->next = currNode;
    }
}
```

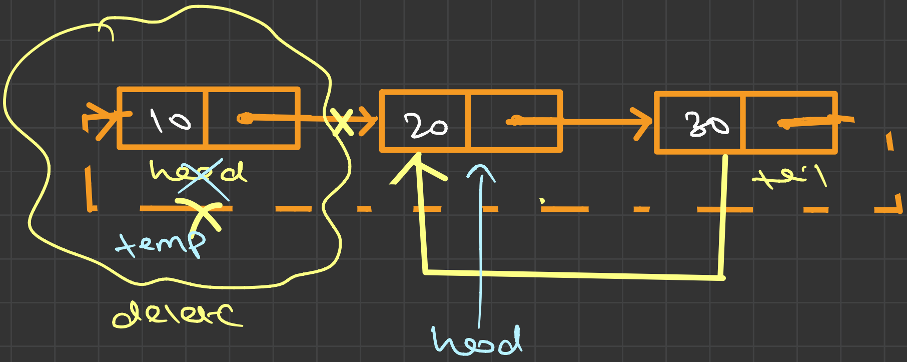
Delete from head

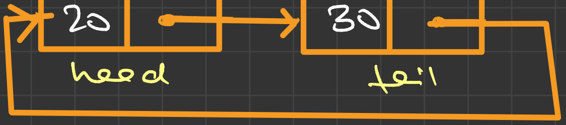
Edge case

↳ head == null ⟶ return

↳ head == tail

temp = head
head = null
tail = null
delete temp



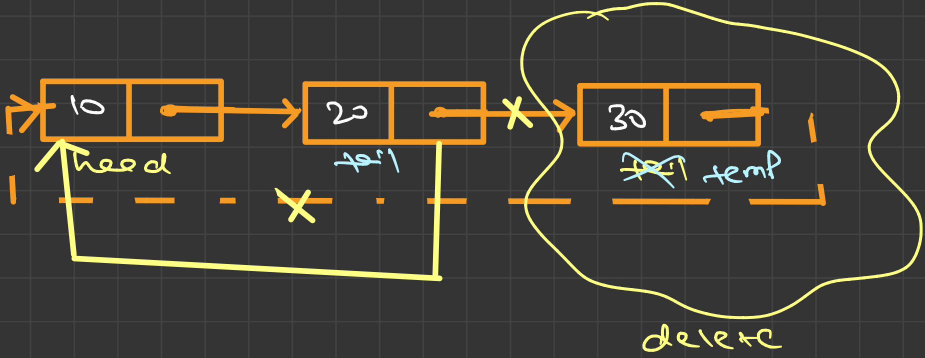


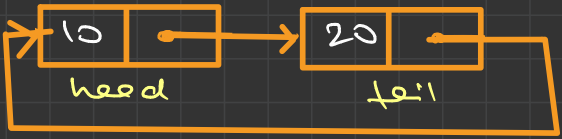
Delete from tail

Edge case

- head == null → return
- head == tail

  temp = head
  head = null
  tail = null
  delete temp

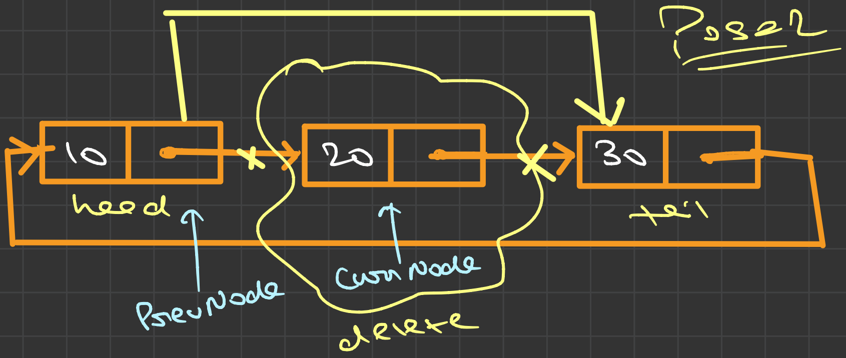


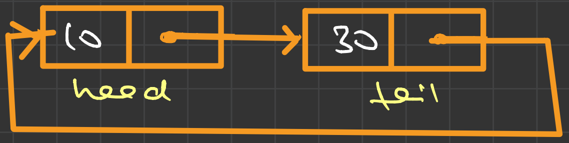


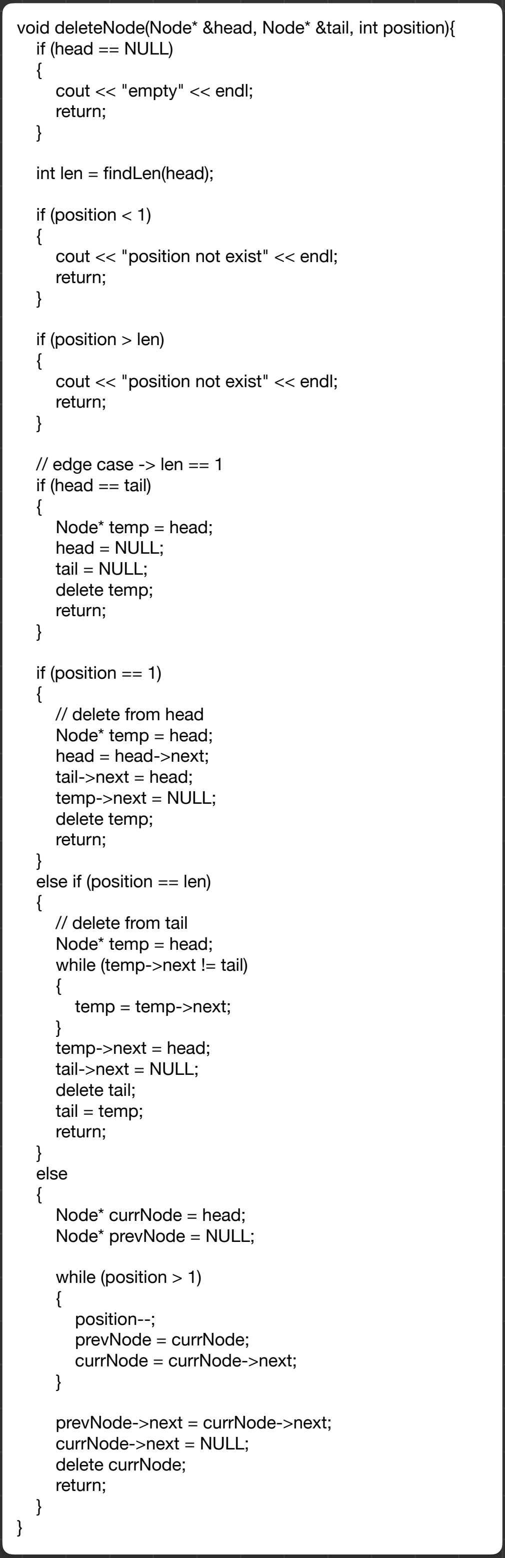
Delete At Pos

Edge Case

- ↳ Pos < 1 → invalid pos
- ↳ Pos > len → invalid pos
- ↳ head == tail → single element
- ↳ head == null → no element







```
void deleteNode(Node* &head, Node* &tail, int position){
    if (head == NULL)
    {
        cout << "empty" << endl;
        return;
    }

    int len = findLen(head);

    if (position < 1)
    {
        cout << "position not exist" << endl;
        return;
    }

    if (position > len)
    {
        cout << "position not exist" << endl;
        return;
    }

    // edge case -> len == 1
    if (head == tail)
    {
        Node* temp = head;
        head = NULL;
        tail = NULL;
        delete temp;
        return;
    }

    if (position == 1)
    {
        // delete from head
        Node* temp = head;
        head = head->next;
        tail->next = head;
        temp->next = NULL;
        delete temp;
        return;
    }
    else if (position == len)
    {
        // delete from tail
        Node* temp = head;
        while (temp->next != tail)
        {
            temp = temp->next;
        }
        temp->next = head;
        tail->next = NULL;
        delete tail;
        tail = temp;
        return;
    }
    else
    {
        Node* currNode = head;
        Node* prevNode = NULL;

        while (position > 1)
        {
            position--;
            prevNode = currNode;
            currNode = currNode->next;
        }

        prevNode->next = currNode->next;
        currNode->next = NULL;
        delete currNode;
        return;
    }
}
```

Driver Code

```
int main(){
    Node* head = NULL;
    Node* tail = NULL;

    // insertAtHead(head, tail, 50);
    // insertAtHead(head, tail, 40);
    // insertAtHead(head, tail, 30);
    // insertAtHead(head, tail, 20);
    insertAtHead(head, tail, 10);

    print(head);
    cout << endl;

    // insertAtTail(head, tail, 100);
    // insertAtTail(head, tail, 200);

    // insertAtPosition(head, tail, 100, 6);

    deleteNode(head, tail, 1);

    print(head);
    cout << endl;

    cout << "len: " << findLen(head) << endl;

    cout << "head: " << head << endl;
    cout << "tail: " << tail << endl;

    // cout << "head->data: " << head->data << endl;
    // cout << "tail->data: " << tail->data << endl;

    // cout << "head->next->data: " << head->next->data << endl;
    // cout << "tail->next->data: " << tail->next->data << endl;
}
```