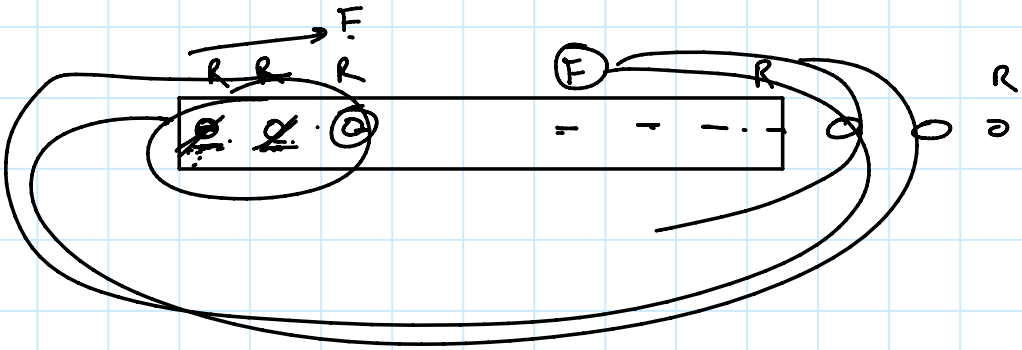
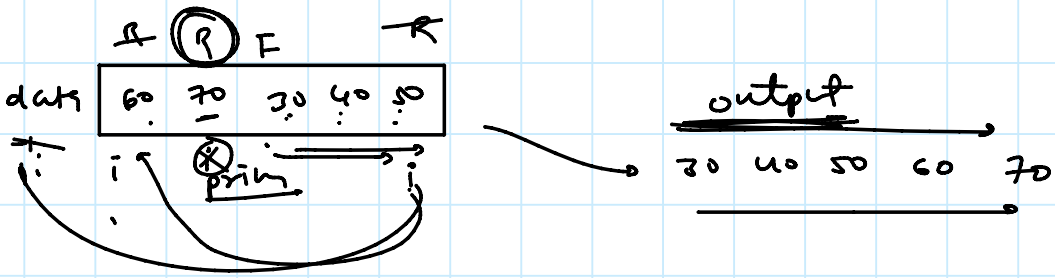
DAY 16

11 May 2024 01:03 PM






```
void output ()
{
    for(i=front ; i != rear ; i++)
    {
        cout << data[i]
        if (i == maxQ-1)
            i = -1;
    }
    cout << data[rear];
```

.

```
#include<iostream>
using namespace std;
#define MAXQ 5
class CirQueue
{
    int data[MAXQ];
    int rear,front;
    public:
    CirQueue()
    {
        front=rear=-1;
    }
    void add(int no)
    {
```

```
        if(front == 0 and rear==MAXQ-1 or rear == front-1)
        {
            cout<<"Overflow\n";
            return;
        }
        if(front==-1)
            front = rear = 0;
        else if(rear==MAXQ-1)
            rear=0;
        else
            rear++;
        data[rear]=no;
    }
    void del()
    {
        if(front == -1)
        {
            cout<<"Unverflow\n";
            return;
        }
        if(front == rear)
            front = rear = 0;
        else if(front ==  MAXQ-1)
            front = 0;
        else
            front ++;
    }
    int firstElement()
    {
        if(front == -1)
            return 0;
        return data[front];
    }
    int lastElement()
    {
        if(front == -1)
            return 0;
        return data[rear];
    }
};
int main()
{
    CirQueue q1;
    q1.add(10);
    q1.add(20);
    q1.add(30);
```

```
    q1.add(40);
    q1.add(50);
    q1.add(60);
    cout<<q1.firstElement()<<" "<<q1.lastElement()<<endl;
    q1.del();
    cout<<q1.firstElement()<<" "<<q1.lastElement()<<endl;
    q1.add(60);
    cout<<q1.firstElement()<<" "<<q1.lastElement()<<endl;
    q1.del();
    q1.add(70);


}
```

Priority queue :→

F
R R R R

50 30 20 10

0 1 2 3 4 5

queue

Add Queue

```
if(Rear== maxQ-1)
        Overflow, Return

if ( Front == -1)
{
    front = Rear = 0;
    data [rear] = no;
    return;
}
Rear++;
for (i=rear-1; i>=front and data[i]<no; i--)
        data[i+1] = data[i];

data[i+1] = no;
```

F
R R R

20 10
30 20

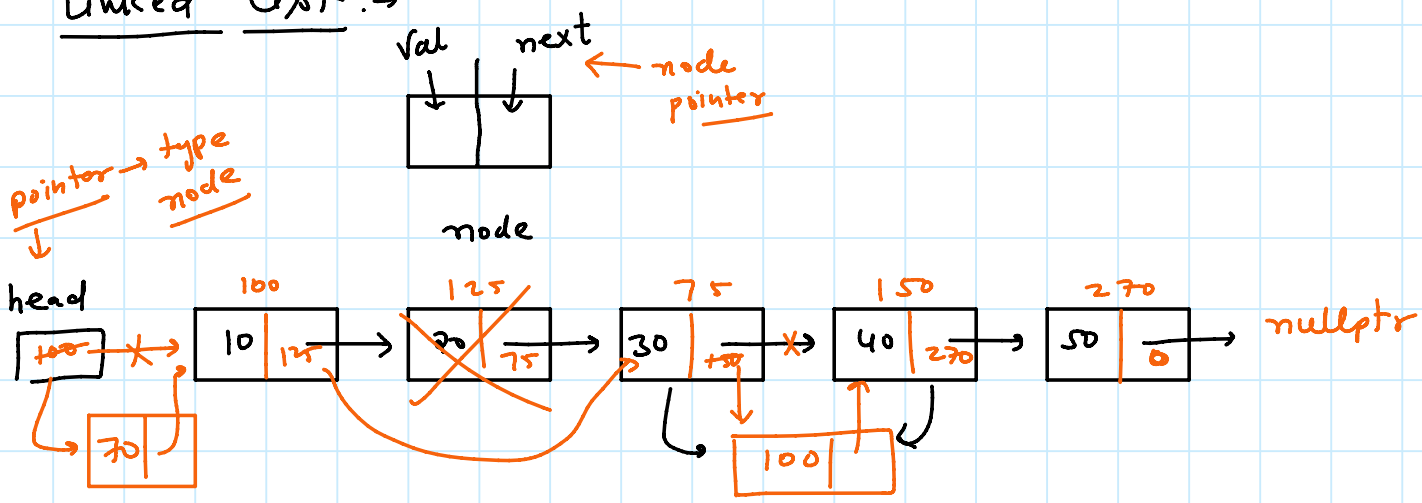
i ↑ ↑

30
no

```
#include<iostream>
using namespace std;
#define MAXQ 5
class PriorityQueue
{
    int data[MAXQ];
    int rear,front;
    public:
    PriorityQueue()
    {
        front=rear=-1;
    }
    void add(int no)
    {
        if(rear==MAXQ-1)
        {
            cout<<"Overflow\n";
            return;
        }
        if(front==-1)
        {   front = rear = 0;
            data[rear]=no;
            return;
        }
        rear++;
        int i;
        for(i=rear-1;i>=front and data[i]<no;i--)
            data[i+1]=data[i];
        data[i+1]=no;
    }
    void del()
    {
        if(front == -1)
        {
            cout<<"Unverflow\n";
            return;
        }
        if(front == rear)
            front = rear = -1;
        else
            front ++;
    }
    int firstElement()
    {
        if(front == -1)
```

```
            return 0;
        return data[front];
    }
    int lastElement()
    {
        if(front == -1)
            return 0;
        return data[rear];
    }
};
int main()
{
    PriorityQueue q1;
    q1.add(20);
    q1.add(30);
    cout<<q1.firstElement()<<" "<<q1.lastElement()<<endl;
    q1.add(50);
    q1.add(40);
    cout<<q1.firstElement()<<" "<<q1.lastElement()<<endl;
    q1.del();
    cout<<q1.firstElement()<<" "<<q1.lastElement()<<endl;
}
```

Linked List :→



operations

1. Add First
2. Add last
3. Add before
4. Add After

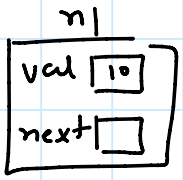
1. del first
2. del last
3. del node

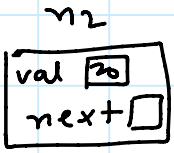
③ Add before ③ del
④ Add After

1. Linear Linked list (Singly)

① Add First :→

static object
```
node n1(10);
node n2(20);
```



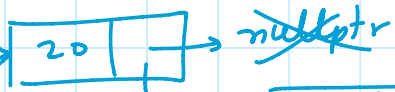



```
class node
{ public:
    int val;
    node * next;
    node (int no)
    {
        val = no;
        next = nullptr;
    }
};
```

```
node *head = new node(10);
```

head → val → 10

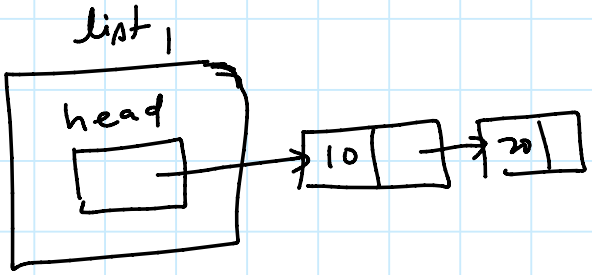
```
head -> next = new node (20)
```



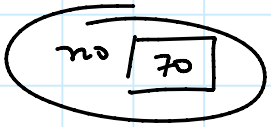

```
head -> next -> next = new node (30)
```

```
class LinkedList
{
    node * head;
```

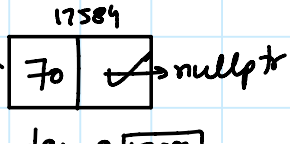
```
LinkedList list1;
```



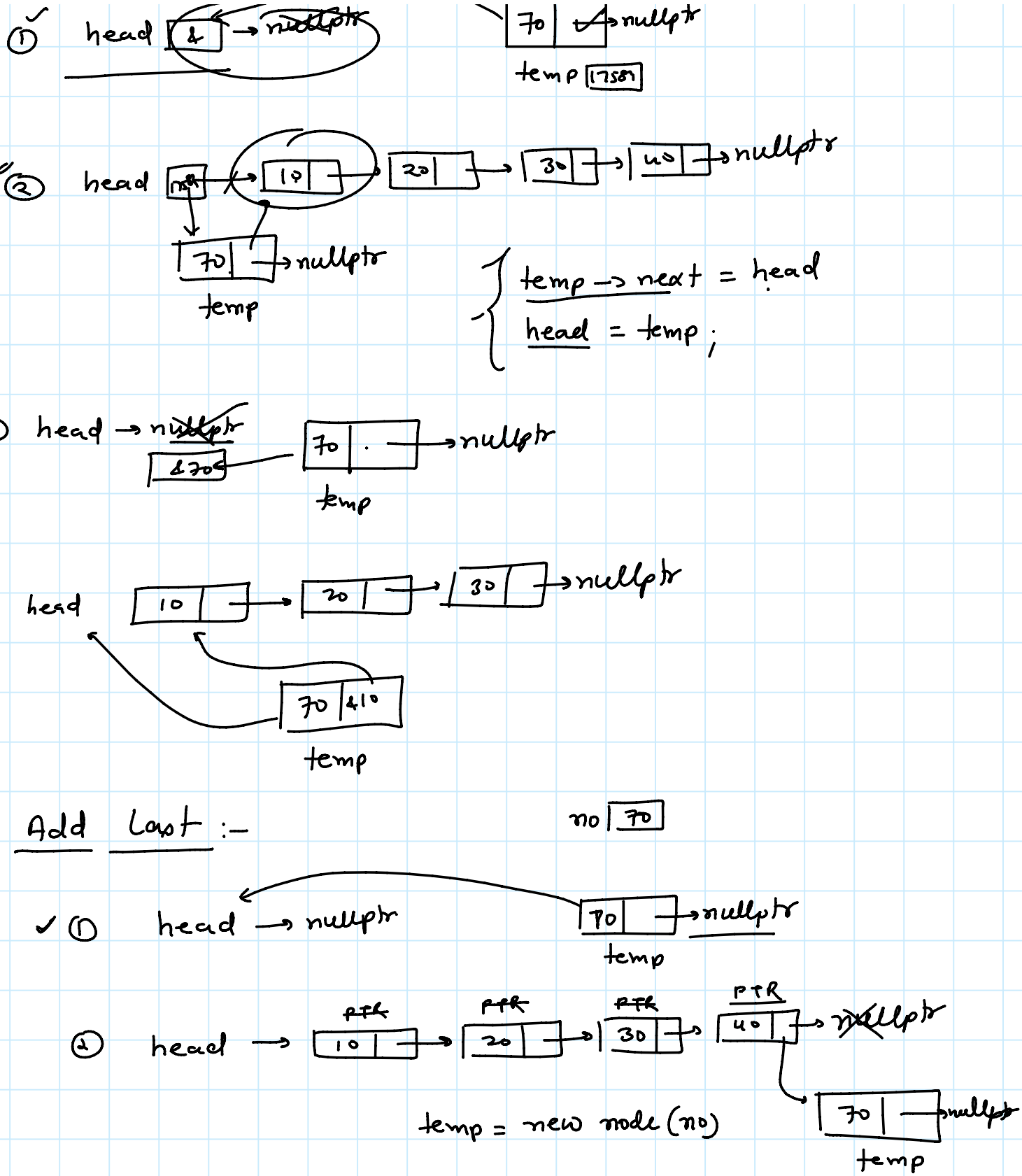
Add First




```
node * temp = new node (no)
```



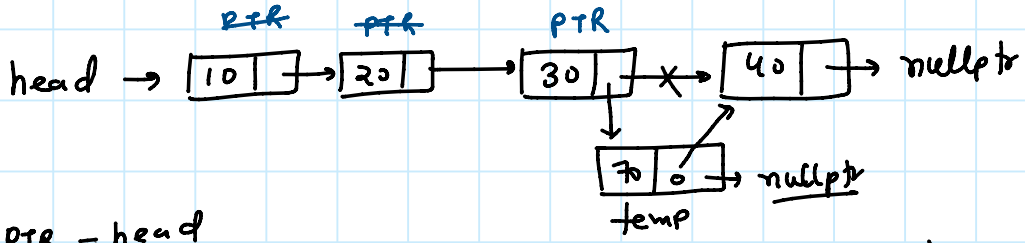
① ✓ head 4 → nullptr




```
PTR = head
while (PTR -> next != nullptr)
{
        PTR = PTR -> next;
```

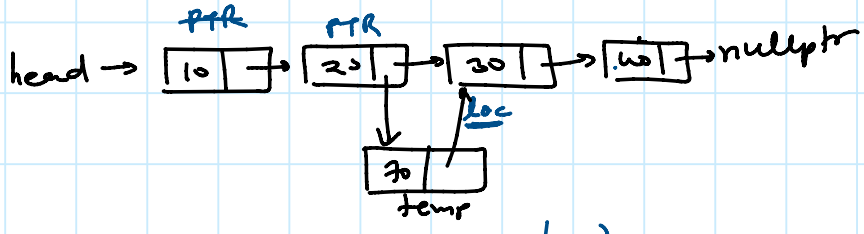
```
{
        PTR = PTR -> next;
}
PTR -> next = temp;
```

Add After :→ no 70 loc 30




```
PTR = head
while (PTR != nullptr && PTR -> val != loc)
        PTR = PTR -> next;
if (PTR == nullptr)
{       cout << " location not found ";
        return
}
temp = new node(no)
temp -> next = ptr -> next;
PTR -> next = temp;
```

Add Before :- no 70 loc 30




```
if (head -> val == loc)
{       AddFirst(no)
        return;
}
```

```
}
return;
```

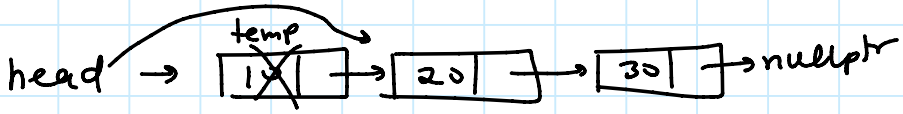
```
PTR = head
while (PTR -> next != nullptr and PTR -> next -> val != loc)
{
    PTR = PTR -> next;
}
if(PTR -> next == nullptr)
        loc not found return

temp = new node(no)
temp -> next = PTR -> next
PTR -> next = temp
```

Del first:-

① head → nullptr ⟶ Underflow

② 

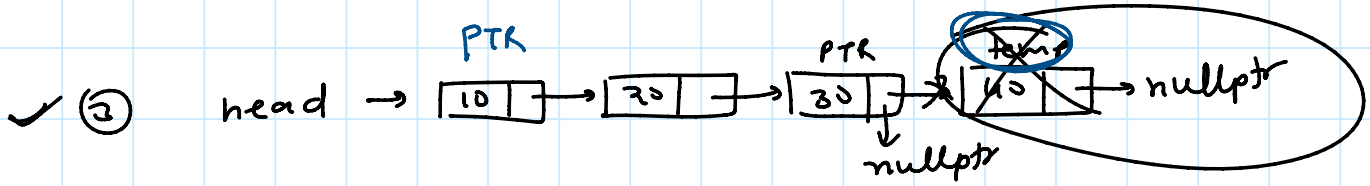
③ 


```
temp = head
head = head -> next;
delete temp;
```

del last:-

× ① head → nullptr ⟶ Underflow

→ ② head → [10] → nullptr ⟶ delfirst , Return

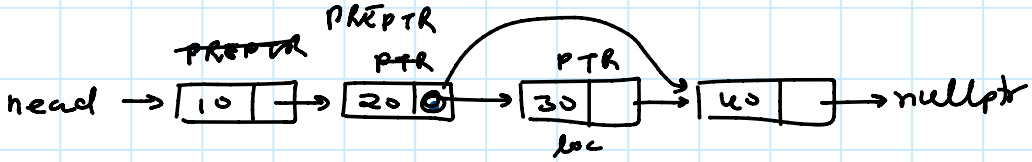



```
PTR = head
while (PTR->next->next != nullptr)
        PTR = PTR->next;

temp = PTR->next;   ⟶ last node
PTR->next = nullptr ⟶ second last node
delete temp;
```

del node :-

loc [30]



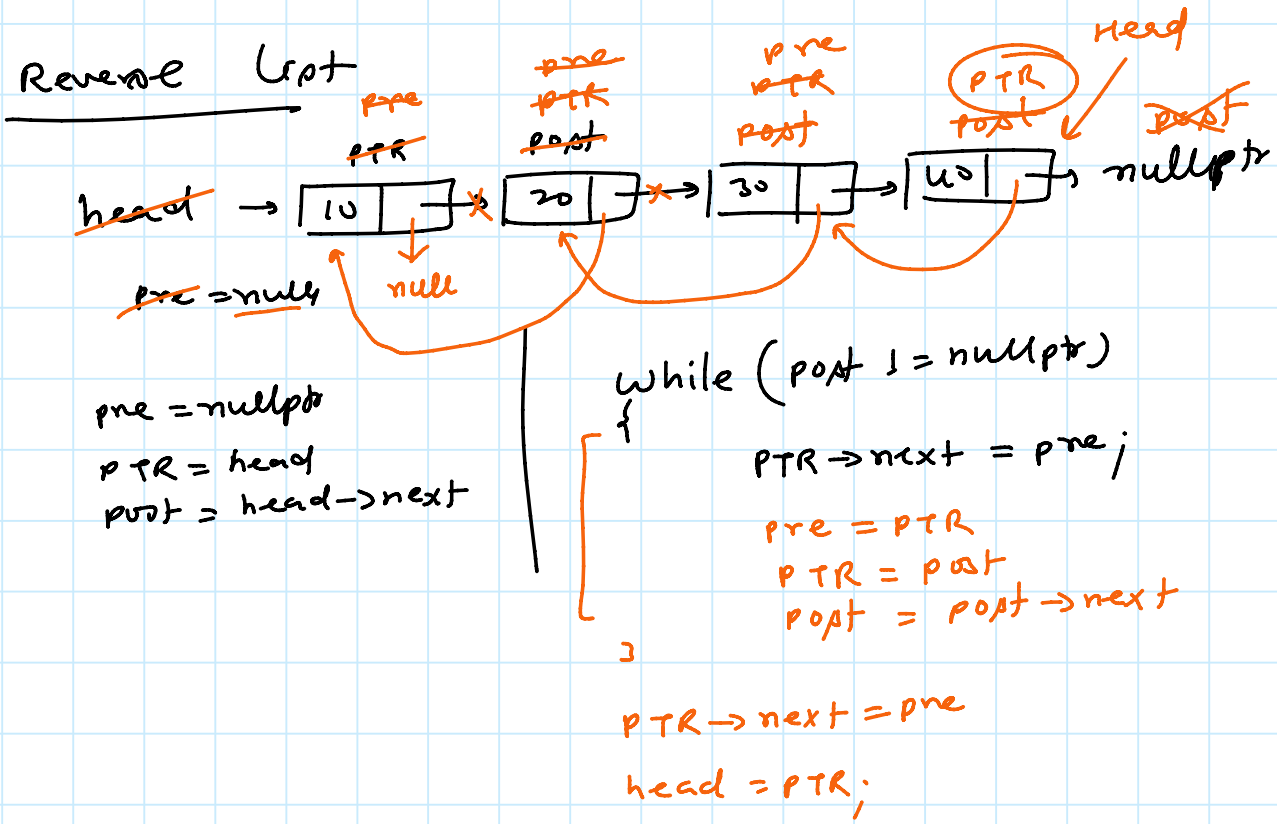

```
if ( head->val == loc)
        delfirst()
        return
PTR = head->next
PREPTR = head
while (PTR != nullptr && PTR->val != loc)
{
        PREPTR = PTR
        PTR = PTR->next
}
if (PTR == nullptr)
```

if (PTR == nullptr)

loc not found, Return

PREPTR -> next = PTR -> next

delete PTR




```
#include<iostream>
using namespace std;
class node
{
    public:
        int val;
        node *next;
        node(int no)
        {
            val=no;
            next=nullptr;
        }
};
```

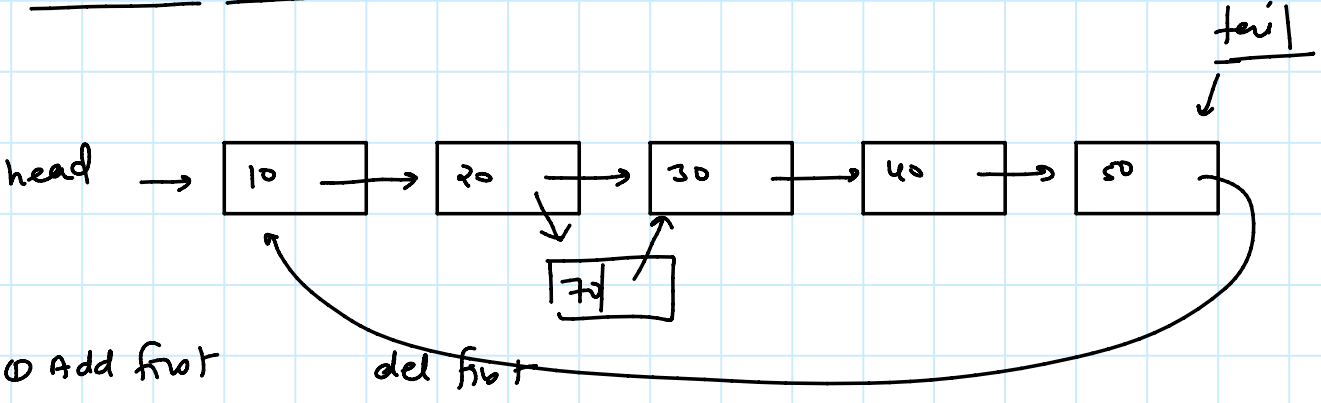
```
class LinkedList
{
    node *head;
    public:
        LinkedList()
        {
            head=nullptr;
        }
        void addFirst(int no)
        {
            node *temp=new node(no);
            temp->next=head;
            head=temp;
        }
        void addLast(int no)
        {
            node *temp=new node(no);
            if(head==nullptr)
            {
                head=temp;
                return;
            }
            node *ptr=head;
            while (ptr->next!=nullptr)
            {
                ptr=ptr->next;
            }
            ptr->next=temp;
        }
        void addAfter(int no, int loc)
        {
            node *ptr=head;
            while(ptr!=nullptr and ptr->val!=loc)
                ptr=ptr->next;
            if(ptr==nullptr)
            {
                cout<<"Location not found\n";
                return;
            }
            node *temp=new node(no);
            temp->next=ptr->next;
            ptr->next=temp;
        }
        void addBefore(int no, int loc)
        {
            if(head->val == loc)
```

```
        {
            addFirst(no);
            return;
        }
        node *ptr=head;
        while(ptr->next!=nullptr and ptr->next->val!=loc)
            ptr=ptr->next;
        if(ptr->next==nullptr)
        {
            cout<<"Location not found\n";
            return;
        }
        node *temp=new node(no);
        temp->next=ptr->next;
        ptr->next=temp;
    }
    void output()
    {
        node *ptr=head;
        while(ptr!=nullptr)
        {
            cout<<ptr->val<<" ";
            ptr=ptr->next;
        }
        cout<<endl;
    }
    void delFirst()
    {
        if(head==nullptr)
        {
            cout<<"Underflow\n";
            return;
        }
        node *temp=head;
        head=head->next;
        cout<<temp->val<<" deleted\n";
        delete temp;
    }
    void delLast()
    {
        if(head==nullptr)
        {
            cout<<"Underflow\n";
            return;
        }
        if(head->next == nullptr)
```

```
        {
            delFirst();
            return;
        }
        node *ptr=head;
        while (ptr->next->next != nullptr)
        {
            ptr=ptr->next;
        }
        node *temp=ptr->next;
        ptr->next = nullptr;
        cout<<temp->val<<" deleted\n";
        delete temp;
    }
    void delNode(int loc)
    {
        if(head==nullptr)
        {
            cout<<"Underflow\n";
            return;
        }
        if(head->val == loc)
        {
            delFirst();
            return;
        }
        node *preptr=head,*ptr=head->next;
        while (ptr != nullptr and ptr->val != loc)
        {
            preptr=ptr;
            ptr=ptr->next;
        }
        if(ptr==nullptr)
        {
            cout<<"Location not found\n";
            return;
        }
        preptr->next=ptr->next;
        delete ptr;
    }
    void reverse()
    {
        if(head==nullptr)
            return;
        node *pre=nullptr, *ptr=head, *post=head->next;
        while (post!=nullptr)
```

```
            {
                ptr->next=pre;
                pre=ptr;
                ptr=post;
                post=post->next;
            }
            ptr->next=pre;
            head=ptr;
        }
};
int main()
{
    LinkedList list1;
    list1.addFirst(50);
    list1.addFirst(20);
    list1.addLast(30);
    list1.output();
    list1.addAfter(10,20);
    list1.output();
    list1.addBefore(40,30);
    list1.output();
    list1.delFirst();
    list1.output();
    list1.delLast();
    list1.output();
    list1.delNode(50);
    list1.output();
    list1.addBefore(20,40);
    list1.output();
    list1.reverse();
    list1.output();
}
```

Circular linked list :-



② Add last del last

```
class LinkedList
{
    node *head, *tail;
public
    LinkedList ()
    {
        head = tail = nullptr;
```