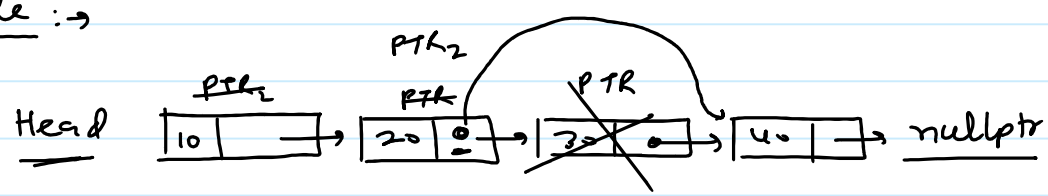
Day 16

04 April 2025 08:33 AM

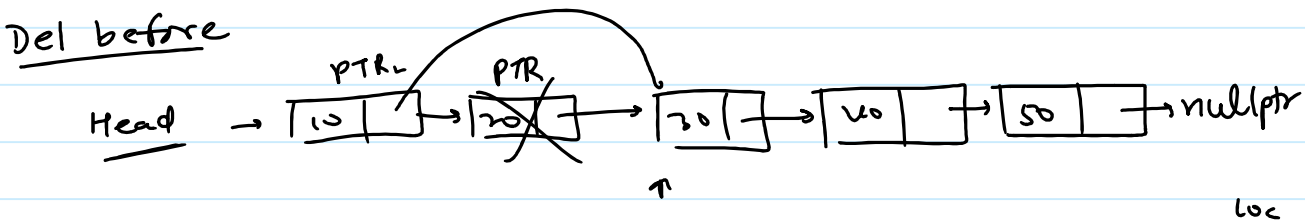
Del node :→



loc
30

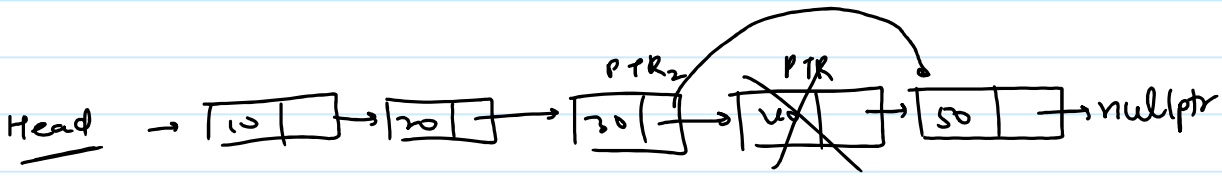
```
if(head == nullptr)
{
    ——— ,return;
}
if(head->value == loc)
{
    Del-first()
    return;
}
PTR = head->next
PTR2 = head
while (PTR != nullptr && PTR->value != loc)
{
    PTR2 = PTR;
    PTR = PTR->next;
}
if (PTR == nullptr)
{
    loc not found, Return;
}
→ PTR2->next = PTR->next;
delete PTR
```

Del before



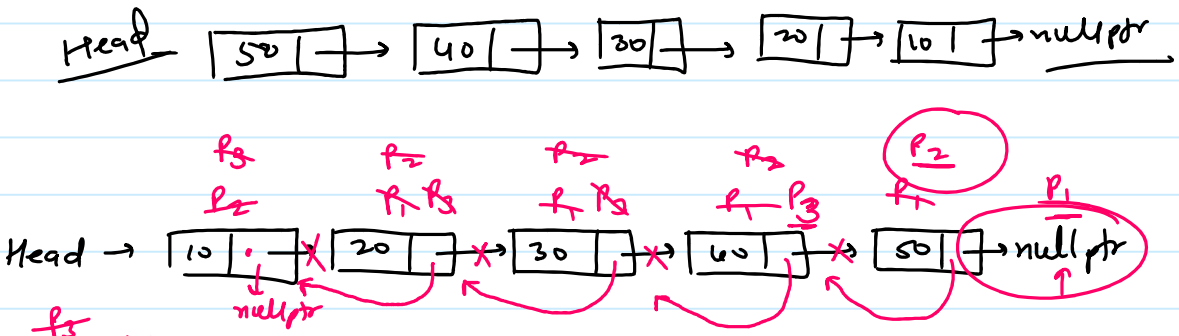
loc
30

Del After →



loc
30

Reverse List :→



$p_1 = head \rightarrow next$

$p_2 = head$

$p_3 = nullptr$

```
while (p1 != nullptr)
{
    p2->next = p3;
    p3 = p2
    p2 = p1
    p1 = p1->next;
}
p2->next = p3
Head = p2.
```

```
#include<iostream>
using namespace std;
class node{
    public:
        int value;
        node *next;
        node(int x)
        {
            value = x;
            next = nullptr;
        }
};
class LinkedList{
```

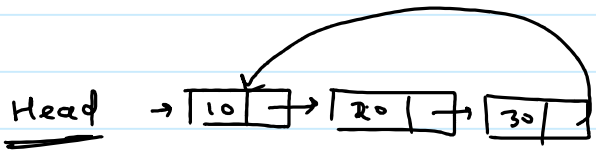
```
private:
    node *head;
public:
    LinkedList()
    {
        head = nullptr;
    }
    void addFirst(int num)
    {
        node *temp = new node(num);     //new dynamic node
        temp->next = head;
        head = temp;
    }
    void addLast(int num)
    {
        node *temp = new node(num);
        if(head == nullptr)
            head = temp;
        else
        {
            node *ptr = head;
            while(ptr->next != nullptr)
            {
                ptr = ptr->next;
            }
            ptr->next = temp;
        }
    }
    void output()
    {
        node *ptr = head;
        while(ptr != nullptr)
        {
            cout<<ptr->value<<" ";
            ptr = ptr->next;
        }
        cout<<endl;
    }
    void addAfter(int num, int loc)
    {
        node*ptr = head;
        while(ptr != nullptr && ptr->value != loc)
            ptr = ptr->next;
        if(ptr == nullptr)
        {
            cout<<"Location not found\n";
            return;
        }
        node *temp = new node(num);
        temp->next = ptr->next;
        ptr->next = temp;
    }
    void addBefore(int num, int loc)
    {
        if(head == nullptr)
        {
```

```
            cout<<"Empty List\n";
            return;
        }
        if(head->value == loc)
        {
            addFirst(num);
            return;
        }
        node*ptr = head;
        while(ptr->next != nullptr && ptr->next->value != loc)
            ptr = ptr->next;
        if(ptr->next == nullptr)
        {
            cout<<"Location not found\n";
            return;
        }
        node *temp = new node(num);
        temp->next = ptr->next;
        ptr->next = temp;
    }
    void delFirst()
    {
        if(head == nullptr)
        {
            cout<<"Empty List\n";
            return;
        }
        node *ptr = head;
        head=head->next;
        cout<<ptr->value<<" deleted\n";
        delete ptr;
    }
    void delLast()
    {
        if(head == nullptr)
        {
            cout<<"Empty List\n";
            return;
        }
        node *ptr = head;
        if(head->next == nullptr)
        {
            head = nullptr;
        }
        else{
            node *ptr2 = nullptr;
            while(ptr->next != nullptr)
            {
                ptr2 = ptr;
                ptr = ptr->next;
            }
            ptr2->next = nullptr;
        }
        cout<<ptr->value<<" deleted\n";
        delete ptr;
    }
```

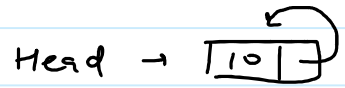
```
        void delNode(int loc)
        {
            if(head == nullptr)
            {
                cout<<"Underflow\n";
                return;
            }
            if(head->value == loc)
            {
                delFirst();
                return;
            }
            node *ptr = head->next, *ptr2=head;
            while (ptr!=nullptr && ptr->value != loc)
            {
                ptr2 = ptr;
                ptr = ptr->next;
            }
            if(ptr == nullptr)
            {
                cout<<"Location not found\n";
                return;
            }
            ptr2->next = ptr->next;
            cout<<ptr->value<<" deleted\n";
            delete ptr;
        }
        void reverse()
        {
            if(head == nullptr || head->next==nullptr)
            {
                return;
            }
            node *p1=head->next,*p2=head,*p3=nullptr;
            while (p1!=nullptr)
            {
                p2->next = p3;
                p3=p2;
                p2=p1;
                p1=p1->next;
            }
            p2->next = p3;
            head=p2;
        }
};
int main()
{
    LinkedList list;
    list.addFirst(10);
    list.addFirst(20);
    list.addFirst(30);
    list.addFirst(40);
    list.output();
    list.addLast(70);
    list.addLast(60);
    list.output();
```

```
        list.addAfter(100,60);
        list.output();
        list.addBefore(200,40);
        list.output();
        list.delFirst();
        list.output();
        list.delLast();
        list.output();
        list.delNode(20);
        list.output();
        list.reverse();
        list.output();
    }
```

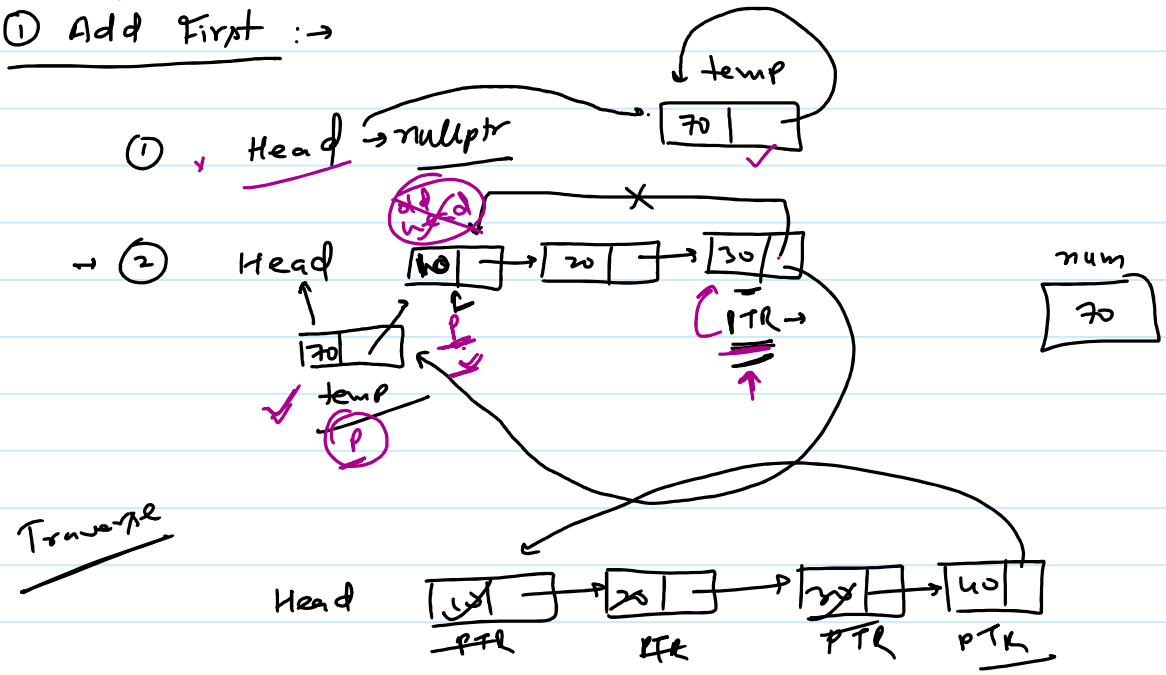
Singly Circular Linked list :→



Head → null



① Add First :→



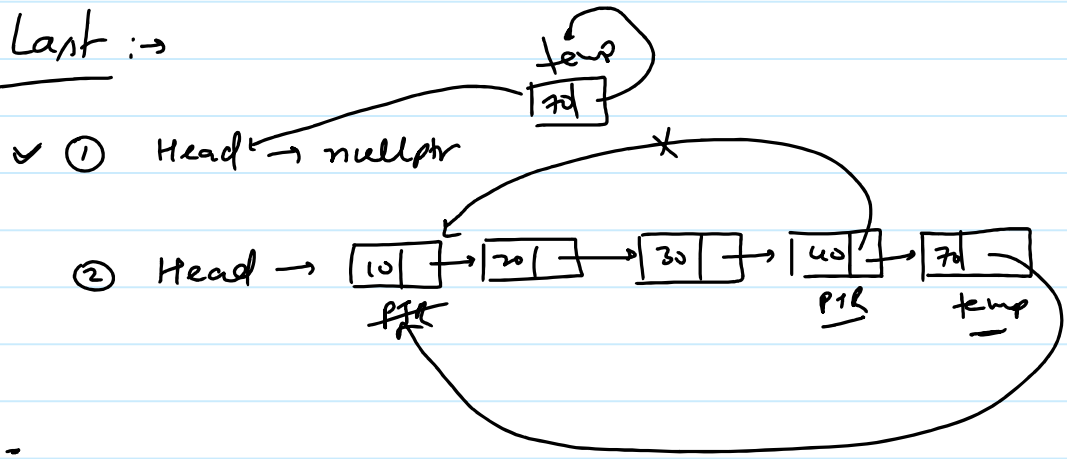

```
PTR = Head
while (PTR->next != head)
{
    cout << PTR->value;
```

PTR = PTR → next;

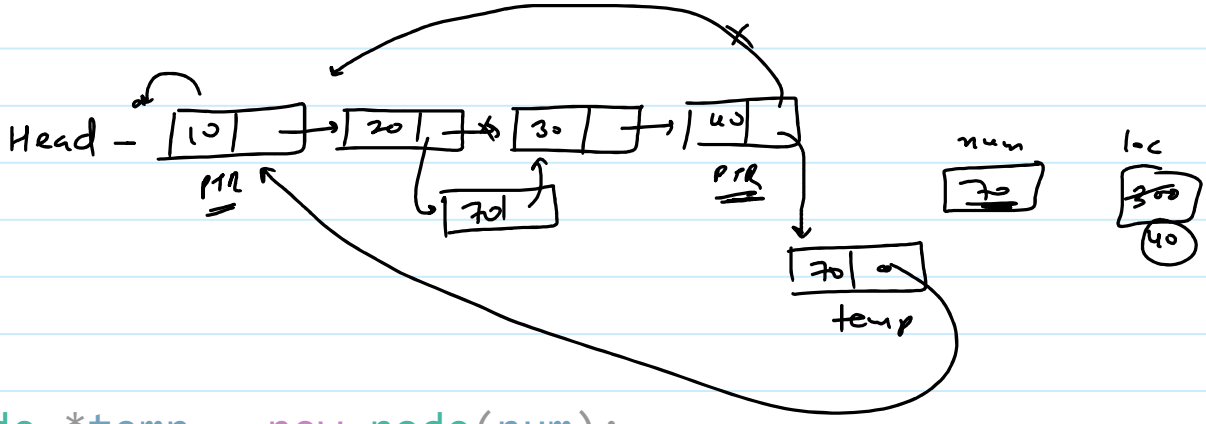
}

cout << PTR → value

② Add Last :→

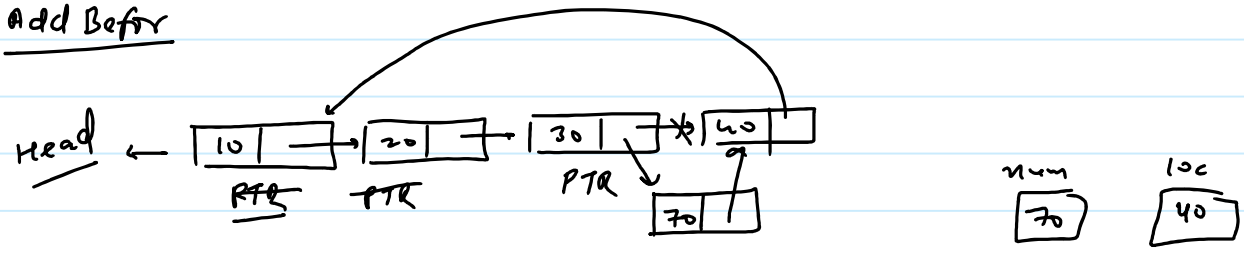


Add After




```
node *temp = new node(num);
        temp->next = ptr->next;
        ptr->next = temp;
```

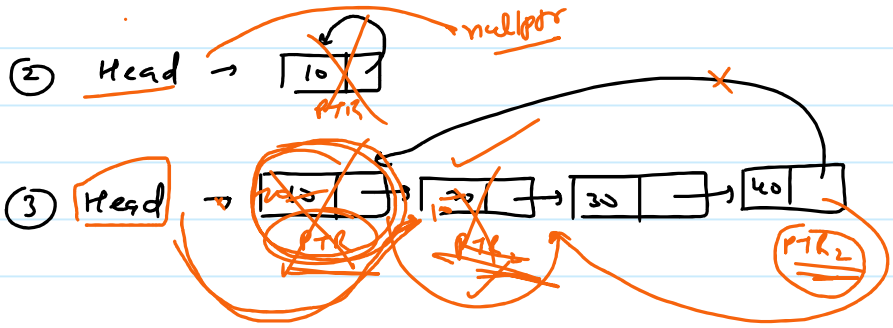
✓ Add Befor



Del first →

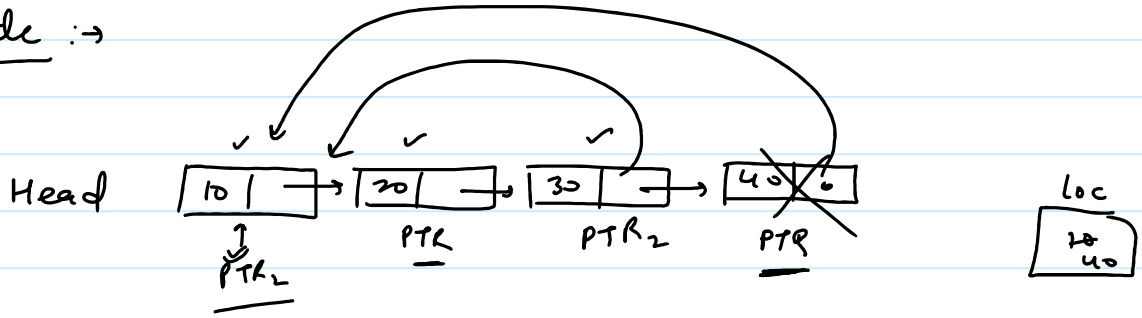
× ① Head → nullptr

② Head →

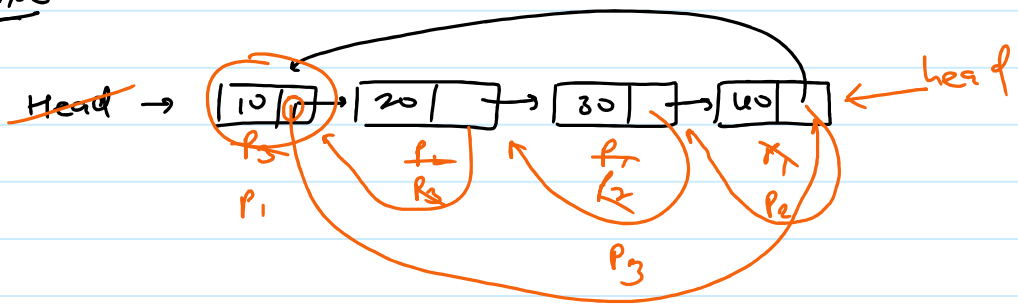


Del Last →

Del node :→



Reverse




```
}
}
P2→next = P3
head→next = P2
head = P2
```

```
#include<iostream>
using namespace std;
class node{
    public:
        int value;
        node *next;
        node(int x)
        {
```

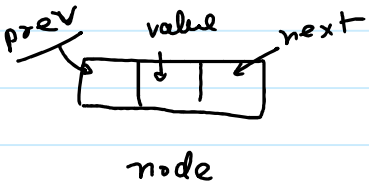
```
            value = x;
            next = nullptr;
        }
};
class CircularLinkedList{
    private:
        node *head;
    public:
        CircularLinkedList()
        {
            head = nullptr;
        }
        void addFirst(int num)
        {
            node *temp = new node(num);     //new dynamic node
            if(head == nullptr)
            {
                head = temp;
                head->next=head;
            }
            else{
                temp->next = head;
                head = temp;
                node *ptr=head->next;
                while (ptr->next != head->next)
                {
                    ptr = ptr->next;
                }
                ptr->next = head;
            }
        }
        void addLast(int num)
        {
            node *temp = new node(num);
            if(head == nullptr){
                head = temp;
                head->next = head;
            }
            else
            {
                node *ptr = head;
                while(ptr->next != head)
                {
                    ptr = ptr->next;
                }
                ptr->next = temp;
                temp->next = head;
            }
        }
        void output()
        {
            if(head == nullptr)
            {
                cout<<"Empty List\n";
                return;
            }
            node *ptr = head;
            while(ptr->next != head)
            {
                cout<<ptr->value<<" ";
```

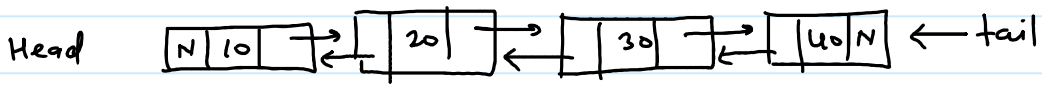
```
            ptr = ptr->next;
        }
        cout<<ptr->value<<endl;
    }
    void addAfter(int num, int loc)
    {
        node*ptr = head;
        while(ptr->next != head && ptr->value != loc)
            ptr = ptr->next;
        if(ptr->next == head && ptr->value != loc)
        {
            cout<<"Location not found\n";
            return;
        }
        node *temp = new node(num);
        temp->next = ptr->next;
        ptr->next = temp;
    }
    void addBefore(int num, int loc)
    {
        if(head == nullptr)
        {
            cout<<"Empty List\n";
            return;
        }
        if(head->value == loc)
        {
            addFirst(num);
            return;
        }
        node*ptr = head;
        while(ptr->next != head && ptr->next->value != loc)
            ptr = ptr->next;
        if(ptr->next == head)
        {
            cout<<"Location not found\n";
            return;
        }
        node *temp = new node(num);
        temp->next = ptr->next;
        ptr->next = temp;
    }
    void delFirst()
    {
        if(head == nullptr)
        {
            cout<<"Empty List\n";
            return;
        }
        node *ptr = head;
        if(head->next == head)
        {
            head = nullptr;
        }
        else{
            head=head->next;
            node *ptr2 = head;
            while (ptr2->next!=ptr)
            {
                ptr2=ptr2->next;
```

```
            }
            ptr2->next = head;
        }
        cout<<ptr->value<<" deleted\n";
        delete ptr;
    }
    void delLast()
    {
        if(head == nullptr)
        {
            cout<<"Empty List\n";
            return;
        }
        node *ptr = head;
        if(head->next == nullptr)
        {
            head = nullptr;
        }
        else{
            node *ptr2 = nullptr;
            while(ptr->next != head)
            {
                ptr2 = ptr;
                ptr = ptr->next;
            }
            ptr2->next = head;
        }
        cout<<ptr->value<<" deleted\n";
        delete ptr;
    }
    void delNode(int loc)
    {
        if(head == nullptr)
        {
            cout<<"Underflow\n";
            return;
        }
        if(head->value == loc)
        {
            delFirst();
            return;
        }
        node *ptr = head->next, *ptr2=head;
        while (ptr->next!=head && ptr->value != loc)
        {
            ptr2 = ptr;
            ptr = ptr->next;
        }
        if(ptr->next == head && ptr->value != loc)
        {
            cout<<"Location not found\n";
            return;
        }
        ptr2->next = ptr->next;
        cout<<ptr->value<<" deleted\n";
        delete ptr;
    }
    void reverse()
    {
        if(head == nullptr || head->next==nullptr)
```

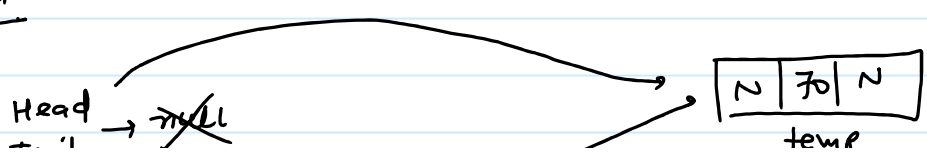
```
            {
                return;
            }
            node *p1=head->next->next,*p2=head->next,*p3=head;
            while (p1!=head)
            {
                p2->next = p3;
                p3=p2;
                p2=p1;
                p1=p1->next;
            }
            p2->next = p3;
            head->next=p2;
            head=p2;
        }
};
int main()
{
    CircularLinkedList list;
    list.addFirst(10);
    list.addFirst(20);
    list.addFirst(30);
    list.addFirst(40);
    list.output();
    list.addLast(5);
    list.output();
    list.addAfter(70,5);
    list.output();
    list.addBefore(100,10);
    list.output();
    list.delFirst();
    list.delLast();
    list.output();
    list.delNode(100);
    list.output();
    list.reverse();
    list.output();
}
```

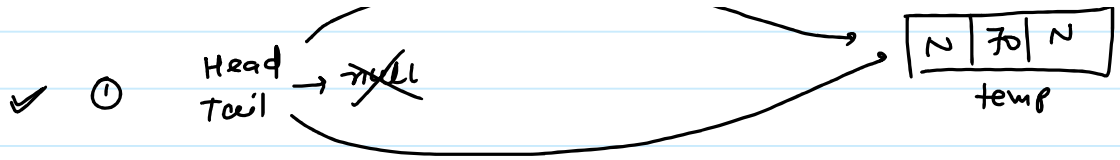
Doubly linked list :-

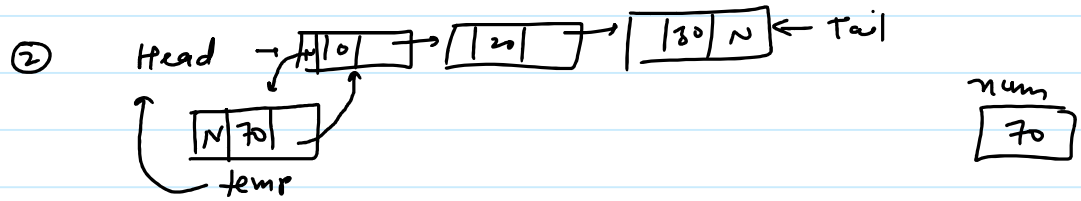




① Add First



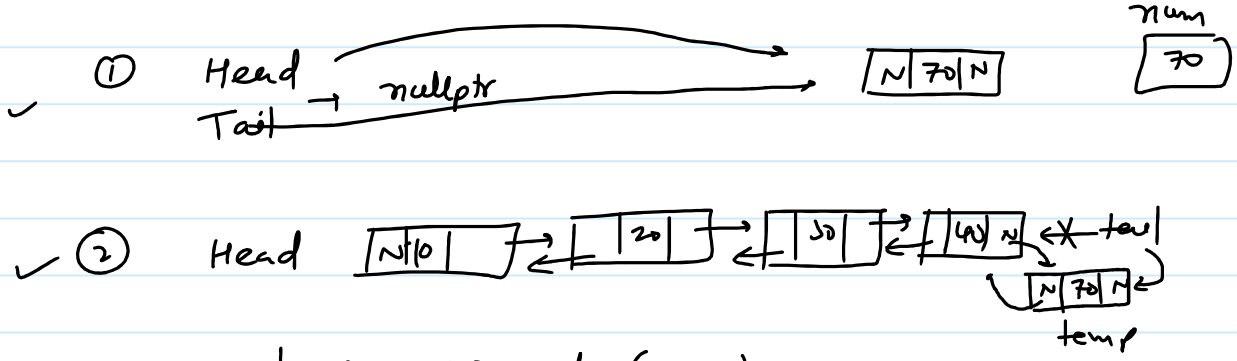




temp = new node(num)

① if(head == nullptr)
{
head = tail = temp;
}
else
{
temp → next = head
head → prev = temp
head = temp;
}

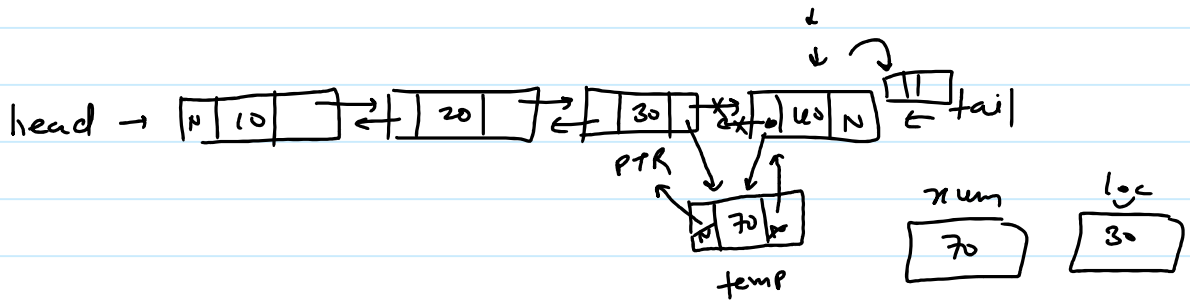
② Add Last :-



temp = new node (num)

- ① if (head == nullptr)
  head = tail = temp;
- ② else
  temp → prev = tail
  tail → next = temp
  tail = temp;

Add After :-



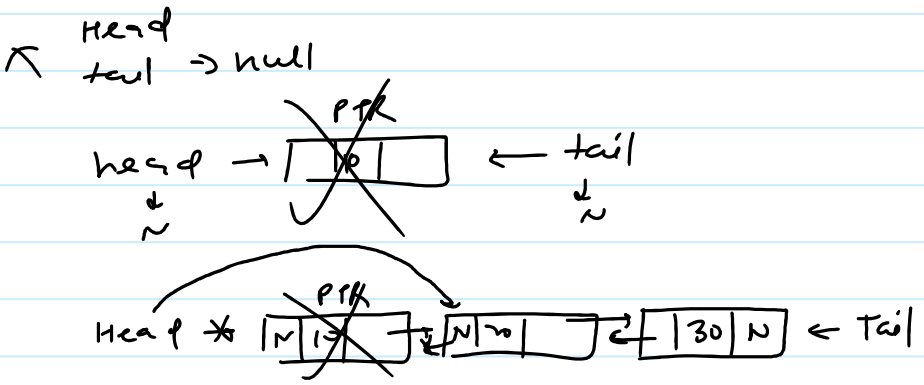
temp → next = PTR → next

PTR → next → prev = temp

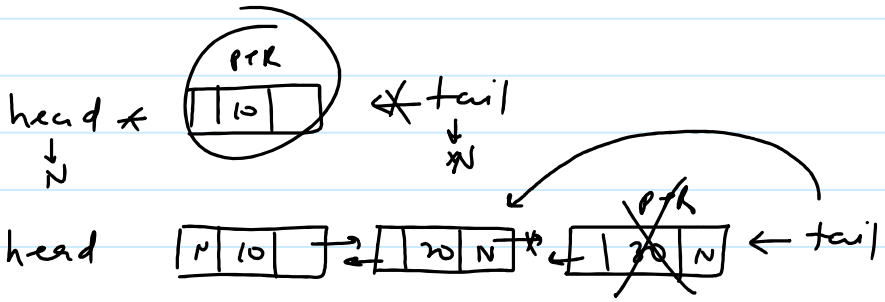
temp → prev = PTR

PTR → next = Temp

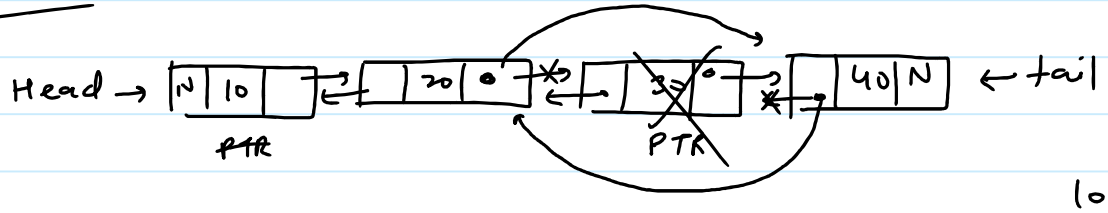
Del first



Del Last



Del Node



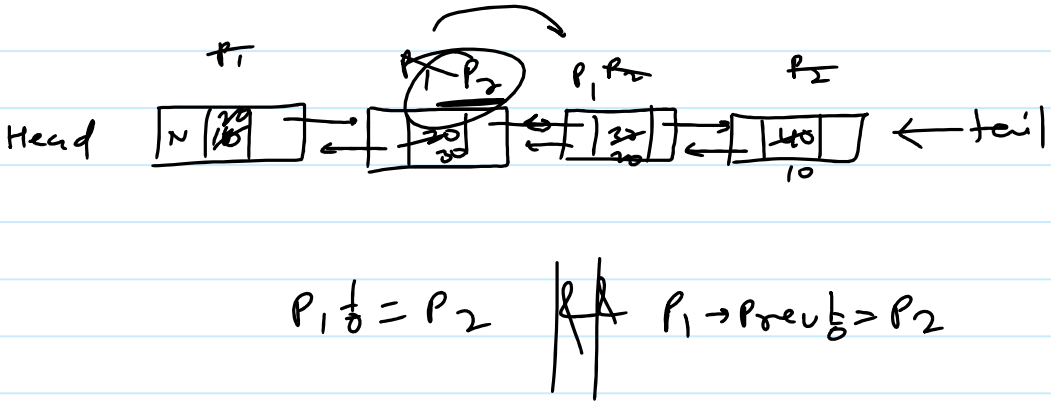
loc
30

loc
30

PTR → prev → next = PTR → next

PTR → next → prev = PTR → prev

Reverse



$P_1 \neq P_2$ $P_1 \rightarrow prev \neq P_2$

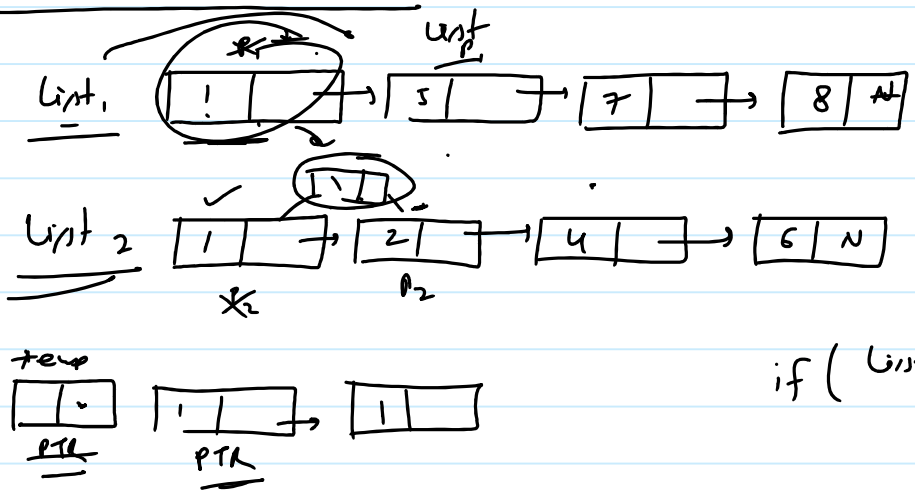
```
#include<iostream>
using namespace std;
class node{
    public:
        int value;
        node *next,*prev;
        node(int x)
        {
            value = x;
            next = prev = nullptr;
        }
};
class DoublyLinkedList{
    private:
        node *head;
        node *tail;
    public:
        DoublyLinkedList()
        {
            head = tail = nullptr;
        }
        void addFirst(int num)
        {
            node *temp = new node(num);     //new dynamic node
            if(head == nullptr)
            {
                head = tail = temp;
            }
            else{
                temp->next = head;
                head->prev = temp;
                head = temp;
            }
        }
        void addLast(int num)
        {
            node *temp = new node(num);
            if(head == nullptr){
                head = tail = temp;
            }
            else
            {
                temp->prev = tail;
                tail->next = temp;
```

```
            tail = temp;
        }
    }
    void output()
    {
        if(head == nullptr)
        {
            cout<<"Empty List\n";
            return;
        }
        node *ptr = head;
        while(ptr!= nullptr)
        {
            cout<<ptr->value<<" ";
            ptr = ptr->next;
        }
        cout<<endl;
    }
    void addAfter(int num, int loc)
    {
        if(head == nullptr)
        {
            cout<<"Empty List\n";
            return;
        }
        if(tail->value == loc)
        {
            addLast(num);
            return;
        }
        node*ptr = head;
        while(ptr != tail && ptr->value != loc)
            ptr = ptr->next;
        if(ptr == tail)
        {
            cout<<"Location not found\n";
            return;
        }
        node *temp = new node(num);
        temp->next = ptr->next;
        ptr->next->prev = temp;
        temp->prev = ptr;
        ptr->next = temp;
    }
    void addBefore(int num, int loc)
    {
        if(head == nullptr)
        {
            cout<<"Empty List\n";
            return;
        }
        if(head->value == loc)
        {
            addFirst(num);
            return;
        }
        node*ptr = head;
        while(ptr != nullptr && ptr->value != loc)
            ptr = ptr->next;
        if(ptr == nullptr)
        {
            cout<<"Location not found\n";
            return;
        }
        node *temp = new node(num);
        temp->next = ptr;
        ptr->prev->next = temp;
        temp->prev = ptr->prev;
        ptr->prev = temp;
    }
    void delFirst()
    {
        if(head == nullptr)
        {
            cout<<"Empty List\n";
            return;
        }
        node *ptr = head;
        if(head->next == nullptr)
        {
            head = tail = nullptr;
```

```
            }
            else{
                head=head->next;
                head->prev = nullptr
            }
            cout<<ptr->value<<" deleted\n";
            delete ptr;
        }
        void delLast()
        {
            if(head == nullptr)
            {
                cout<<"Empty List\n";
                return;
            }
            node *ptr = tail;
            if(head->next == nullptr)
            {
                head = tail = nullptr;
            }
            else{
                tail=tail->prev;
                tail->next = nullptr
            }
            cout<<ptr->value<<" deleted\n";
            delete ptr;
        }
        void delNode(int loc)
        {
            if(head == nullptr)
            {
                cout<<"Underflow\n";
                return;
            }
            if(head->value == loc)
            {
                delFirst();
                return;
            }
            if(tail->value == loc)
            {
                delLast();
                return;
            }
            node *ptr = head;
            while (ptr=tail && ptr->value != loc)
            {
                ptr = ptr->next;
            }
            if(ptr == tail)
            {
                cout<<"Location not found\n";
                return;
            }
            ptr->prev->next = ptr->next;
            ptr->next->prev = ptr->prev;
            cout<<ptr->value<<" deleted\n";
            delete ptr;
        }
        void reverse()
        {

        }
};
int main()
{
    DoublyLinkedList list;
    list.addFirst(10);
    list.addFirst(20);
    list.addFirst(30);
    list.addFirst(40);
    list.output();
    list.addLast(60);
    list.addLast(70);
    list.output();
}
```

merge two sorted lists



if (list1->val < list2->val)

leetcode → 21