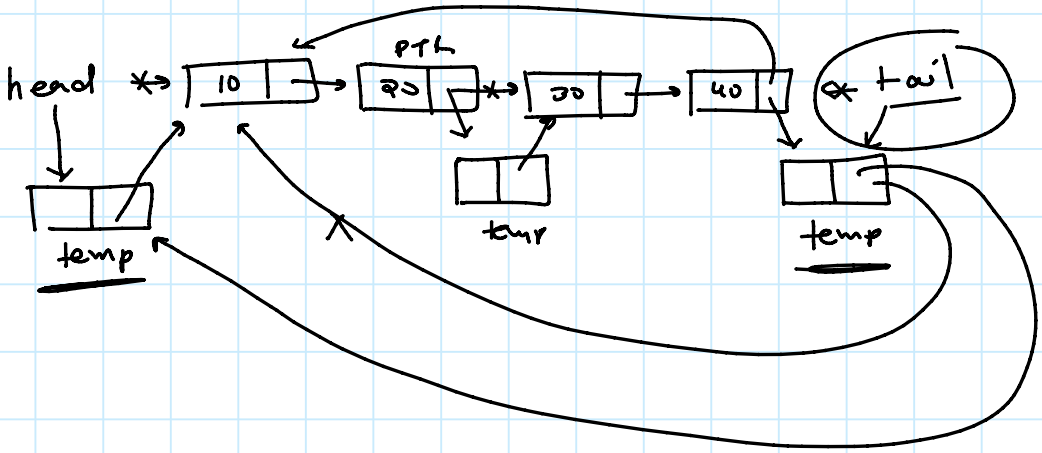
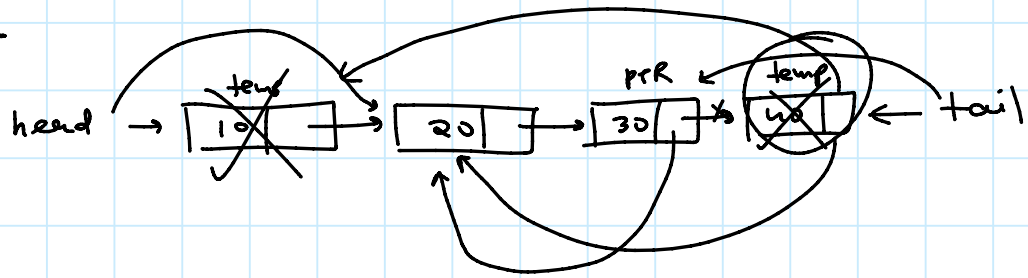
DAY 17

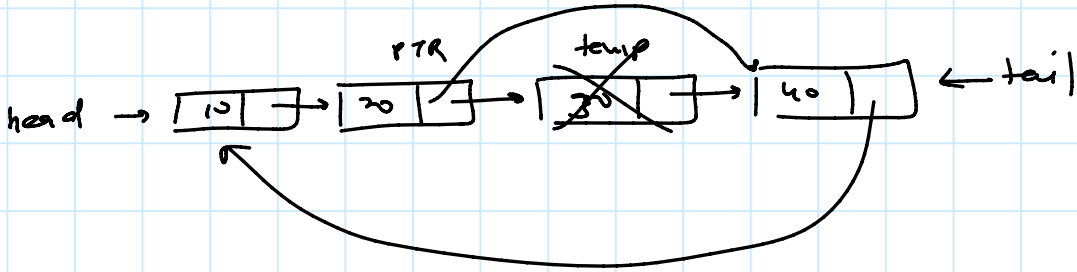
13 May 2024 01:05 PM

Circular linked list :-



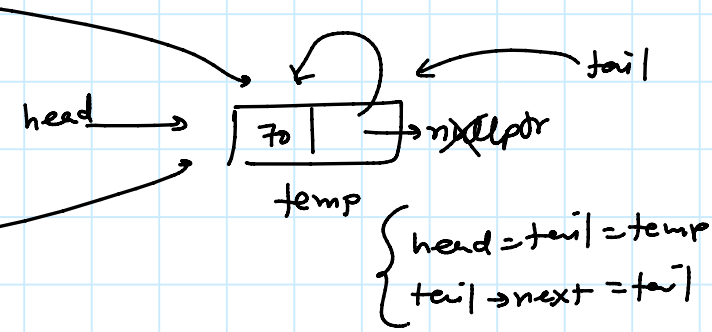
delete



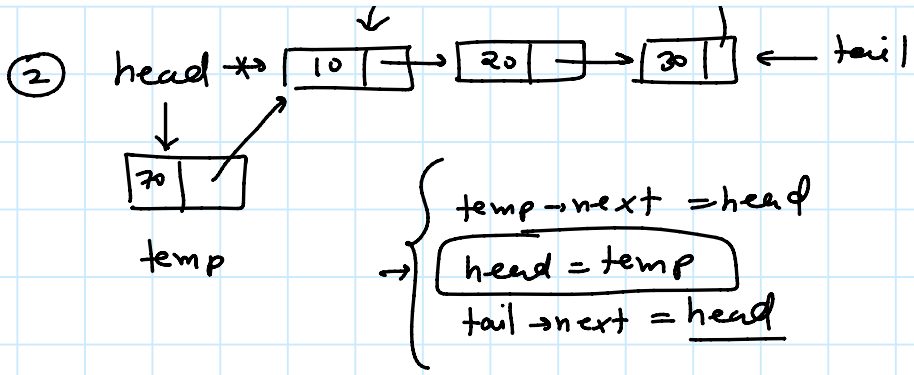


Add :→

① head = tail = nullptr



② head → 10 → 20 → 30 ← tail



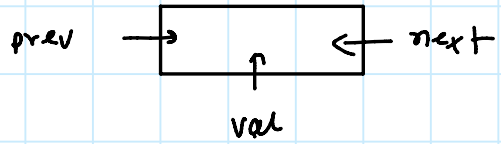

```
#include<iostream>
using namespace std;
class node
{
    public:
        int val;
        node *next;
        node(int no)
        {
            val=no;
            next=nullptr;
        }
};
class CirLinkedList
{
    node *head,*tail;
    public:
        CirLinkedList()
        {
            head=nullptr;
            tail=nullptr;
        }
        void addFirst(int no)
        {
            node *temp=new node(no);
            if(head==nullptr)
            {
                head=tail=temp;
                temp->next=temp;
            }
            else
            {
                temp->next=head;
                head=temp;
                tail->next=head;
            }
        }
        void addLast(int no)
        {
            node *temp=new node(no);
            if(head==nullptr)
            {
```

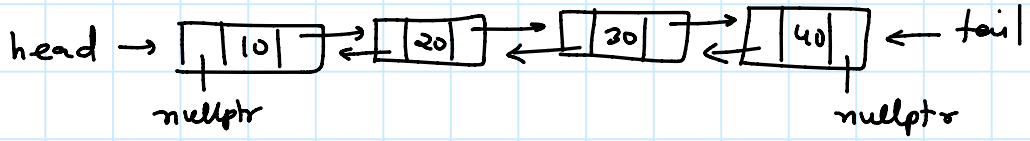
```
            head=tail=temp;
            temp->next=temp;
        }
        else
        {
            tail->next=temp;
            tail=temp;
            tail->next=head;
        }
    }
    void addAfter(int no, int loc)
    {
        if(loc == tail->val)
        {
            addLast(no);
            return;
        }
        node *ptr=head;
        while(ptr!=tail and ptr->val!=loc)
            ptr=ptr->next;
        if(ptr==tail)
        {
            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!=tail and ptr->next->val!=loc)
            ptr=ptr->next;
        if(ptr->next==tail)
        {
            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!=tail)
        {
            cout<<ptr->val<<" ";
            ptr=ptr->next;
        }
        cout<<tail->val<<endl;
    }
```

```
void delFirst()
{
    if(head==nullptr)
    {
        cout<<"Underflow\n";
        return;
    }
    node *temp=head;
    if(head==tail)
    {
        head=tail=nullptr;
    }
    else
    {
        head=head->next;
        tail->next=head;
    }
    cout<<temp->val<<" deleted\n";
    delete temp;
}
void delLast()
{
    if(head==nullptr)
    {
        cout<<"Underflow\n";
        return;
    }
    node *temp=tail;
    if(head == tail)
    {
        head=tail=nullptr;
    }
    else
    {
        node *ptr=head;
        while (ptr->next != tail)
        {
            ptr=ptr->next;
        }
        tail=ptr;
        tail->next=head;
    }
    cout<<temp->val<<" deleted\n";
    delete temp;
}
void delNode(int loc)
{
    if(head==nullptr)
    {
        cout<<"Underflow\n";
        return;
    }
    if(head->val == loc)
    {
        delFirst();
        return;
    }
    if(tail->val == loc)
```

```
            {
                delLast();
                return;
            }
            node *preptr=head,*ptr=head->next;
            while (ptr != tail and ptr->val != loc)
            {
                preptr=ptr;
                ptr=ptr->next;
            }
            if(ptr==tail)
            {
                cout<<"Location not found\n";
                return;
            }
            preptr->next=ptr->next;
            delete ptr;
        }
        void reverse()
        {
            if(head==nullptr)
                return;
            node *pre=tail, *ptr=head, *post=head->next;
            while (ptr!=tail)
            {
                ptr->next=pre;
                pre=ptr;
                ptr=post;
                post=post->next;
            }
            ptr->next=pre;
            head=ptr;
            tail=post;
        }
};
int main()
{
    CirLinkedList l1;
    l1.addFirst(10);
    l1.addLast(50);
    l1.addLast(60);
    l1.output();
    l1.addAfter(20, 10);
    l1.addBefore(40,50);
    l1.addBefore(30,40);
    l1.output();
    l1.reverse();
    l1.output();
    l1.delFirst();
    l1.delLast();
    l1.output();
    l1.delNode(30);
    l1.output();

}
```

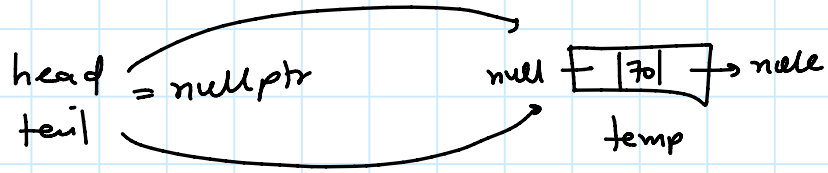
Doubly Linked List :-



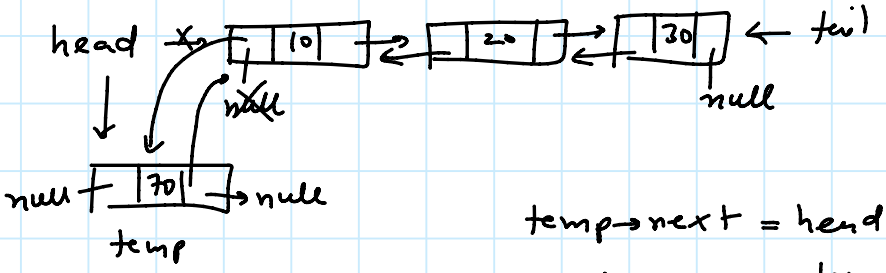


Add First :-

① head = nullptr, tail



head = tail = temp

② 

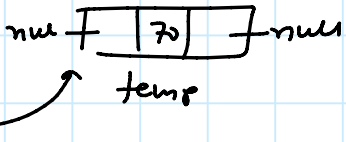
temp → next = head

head → prev = temp

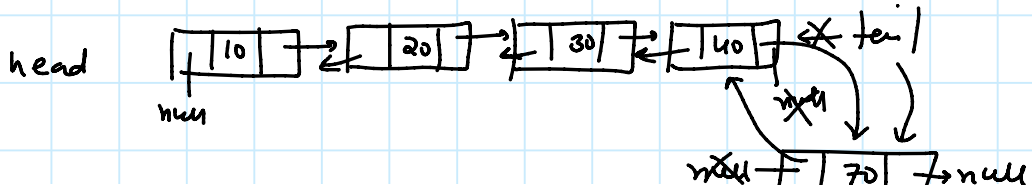
head = temp

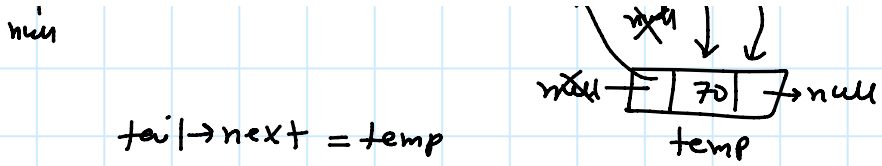
Add last

① head, tail = null



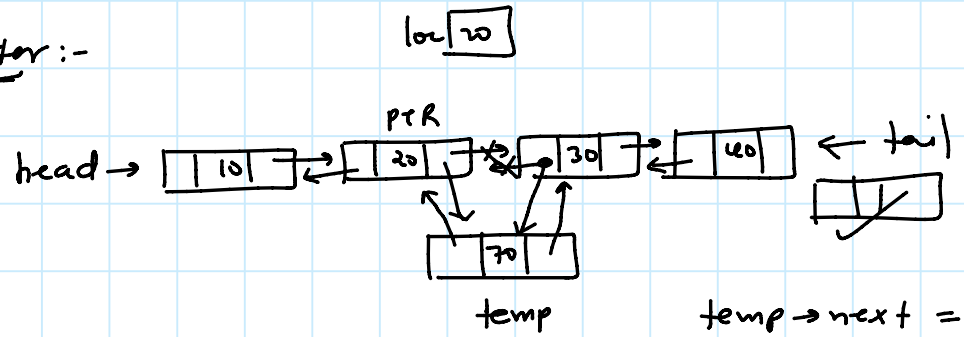
head = tail = temp





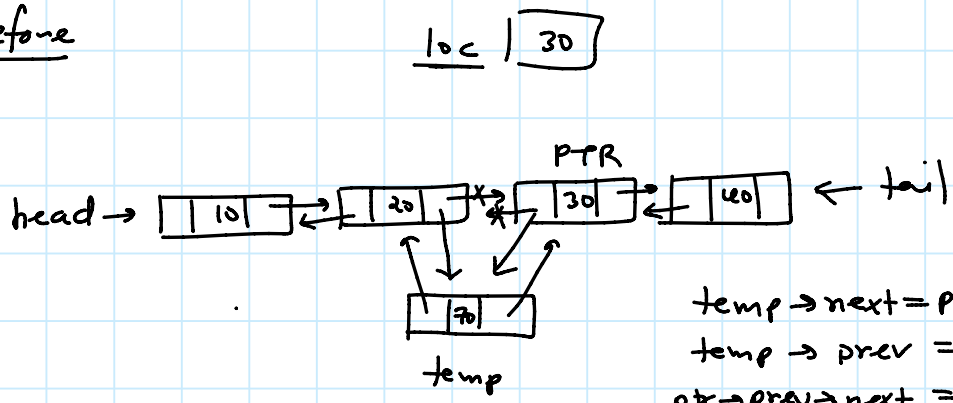

```
tail->next = temp
temp->pre = tail
tail = temp
```

Add After :-

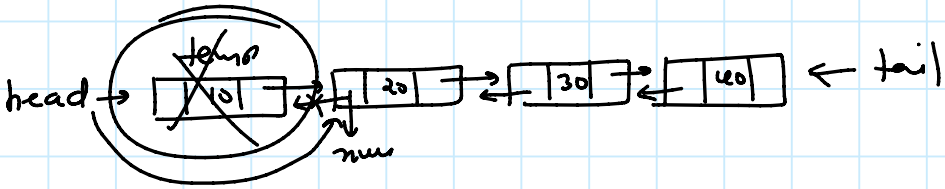


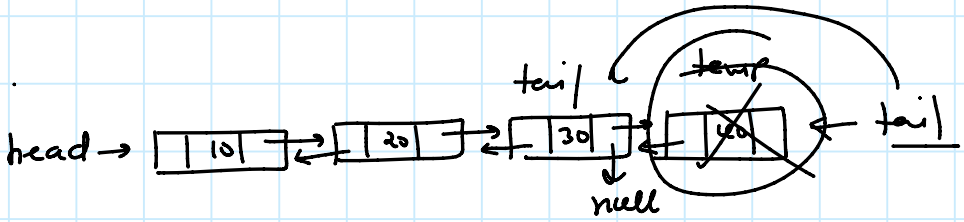

```
temp->next = ptr->next
temp->prev = ptr
ptr->next->prev = temp
ptr->next = temp
```

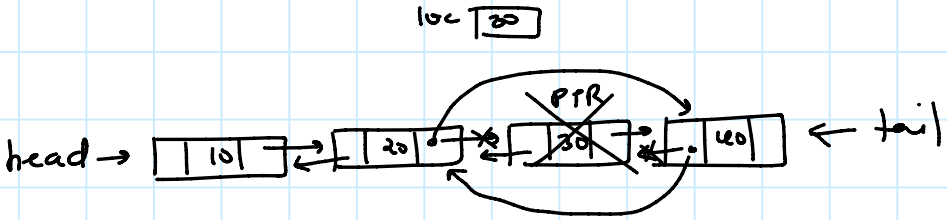
Add before




```
temp->next = PTR
temp->prev = PTR->prev
ptr->prev->next = temp
PTR->prev = temp;
```





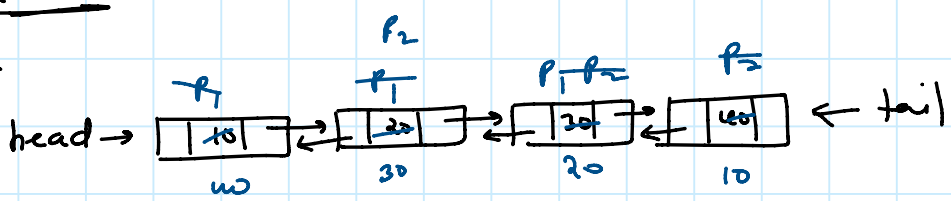


PTR → prev → next = PTR → next

PTR → next → prev = PTR → prev

delete PTR;

Reverse




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

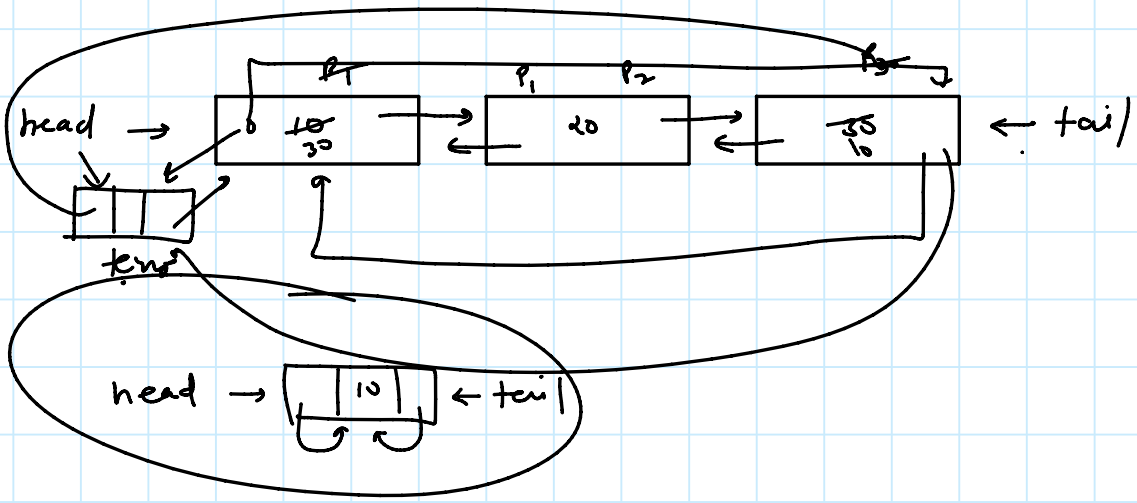
```
        tail=nullptr;
    }
    void addFirst(int no)
    {
        node *temp=new node(no);
        if(head==nullptr)
        {
            head=tail=temp;
        }
        else
        {
            temp->next=head;
            head->prev=temp;
            head=temp;
        }
    }
    void addLast(int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

```
        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;
        temp->prev=ptr->prev;
        ptr->prev->next=temp;
        ptr->prev=temp;
    }
    void output()
    {
        node *ptr=head;
        while(ptr!=tail)
        {
            cout<<ptr->val<<" ";
            ptr=ptr->next;
        }
        cout<<tail->val<<endl;
    }
    void delFirst()
    {
        if(head==nullptr)
        {
            cout<<"Underflow\n";
            return;
        }
        node *temp=head;
        if(head==tail)
        {
            head=tail=nullptr;
        }
        else
        {
            head=head->next;
            head->prev=nullptr;
        }
        cout<<temp->val<<" deleted\n";
        delete temp;
    }
    void delLast()
    {
        if(head==nullptr)
        {
            cout<<"Underflow\n";
            return;
        }
        node *temp=tail;
        if(head == tail)
        {
            head=tail=nullptr;
        }
        else
        {
```

```
                tail=tail->prev;
                tail->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;
            }
            if(tail->val == loc)
            {
                delLast();
                return;
            }
            node *ptr=head->next;
            while (ptr != tail and ptr->val != loc)
            {
                ptr=ptr->next;
            }
            if(ptr==tail)
            {
                cout<<"Location not found\n";
                return;
            }
            ptr->prev->next=ptr->next;
            ptr->next->prev=ptr->prev;
            delete ptr;
        }
        void reverse()
        {
            if(head==nullptr)
                return;
            node *p1=head, *p2=tail;
            while (p1 != p2 and p1->prev != p2)
            {
                swap(p1->val, p2->val);
                p1=p1->next;
                p2=p2->prev;
            }

        }
};
int main()
{
    DoublyLinkedList l1;
    l1.addFirst(10);
    l1.addLast(50);
    l1.addLast(60);
    l1.addAfter(20,10);
```

```
    l1.addBefore(40,50);
    l1.addBefore(30,40);
    l1.output();
    l1.reverse();
    l1.output();
    l1.delFirst();
    l1.delLast();
    l1.delNode(30);
    l1.output();
    return 0;
}
```

Doubly Circular linked list :-



.

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

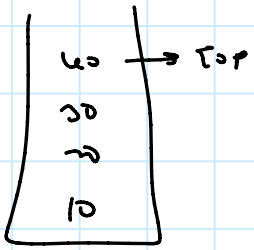
```
        tail=nullptr;
    }
    void addFirst(int no)
    {
        node *temp=new node(no);
        if(head==nullptr)
        {
            head=tail=temp;
        }
        else
        {
            temp->next=head;
            head->prev=temp;
            head=temp;
            head->prev=tail;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

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

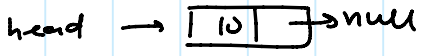
```
{
    if(head==nullptr)
    {
        cout<<"Underflow\n";
        return;
    }
    node *temp=tail;
    if(head == tail)
    {
        head=tail=nullptr;
    }
    else
    {
        tail=tail->prev;
        tail->next=head;
        head->prev=tail;
    }
    cout<<temp->val<<" deleted\n";
   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

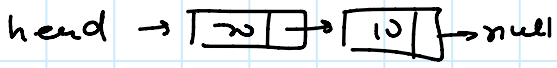
```
                return;
            swap(head->val, tail->val);
            node *p1=head->next, *p2=tail->prev;
            while (p1 != p2 and p1->prev != p2)
            {
                swap(p1->val, p2->val);
                p1=p1->next;
                p2=p2->prev;
            }
        }
};
int main()
{
    CirDoublyLinkedList l1;
    l1.addFirst(10);
    l1.addLast(50);
    l1.addLast(60);
    l1.addAfter(20,10);
    l1.addAfter(30,20);
    l1.addBefore(40,50);
    l1.output();
    l1.reverse();
    l1.output();
    l1.delFirst();
    l1.output();
    l1.delLast();
    l1.output();
    l1.delNode(40);
    l1.output();

}
```

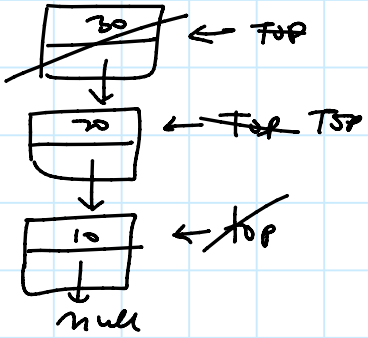
① stack using linked list :-



Add First → push

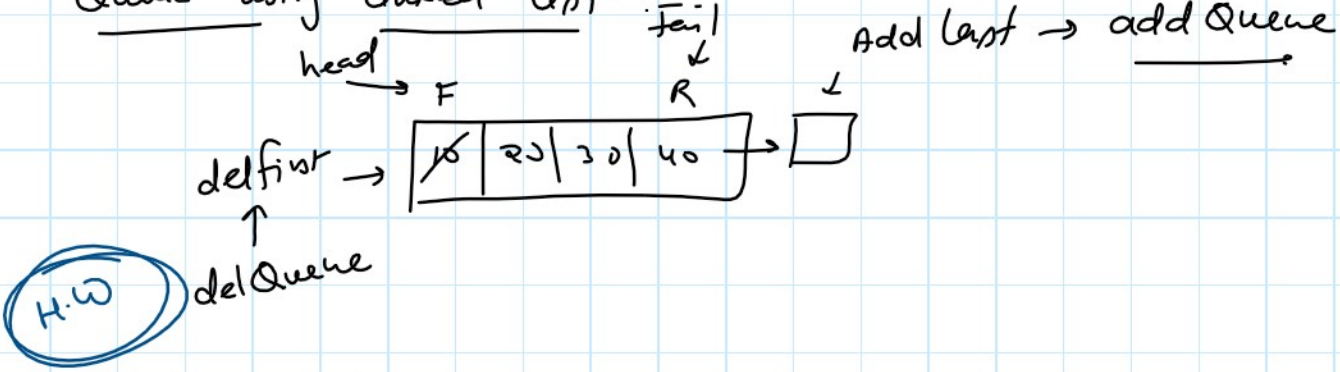




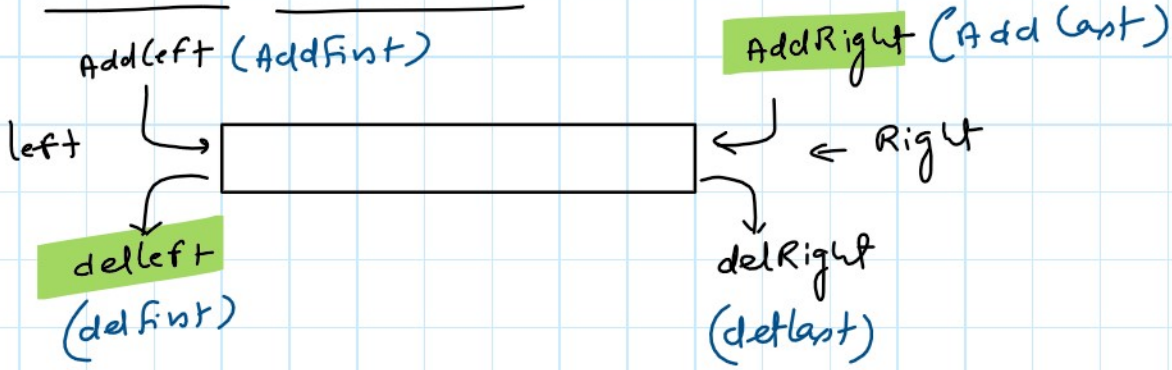


delFirst () → pop

Queue using Linked list :

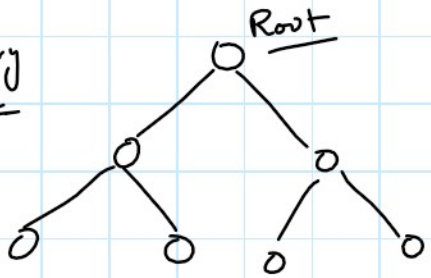


Double ended Queue :→

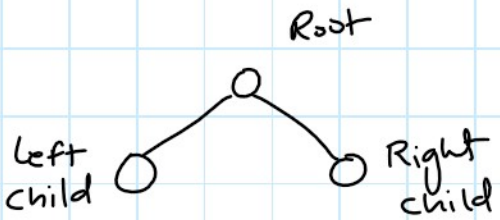


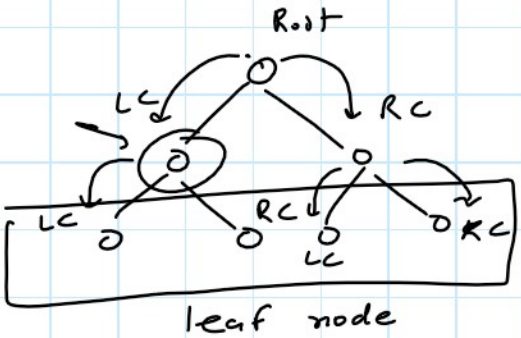
Tree :→ Non linear Data structure

① Binary tree

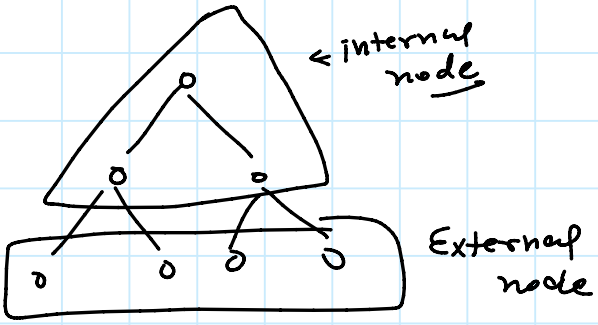


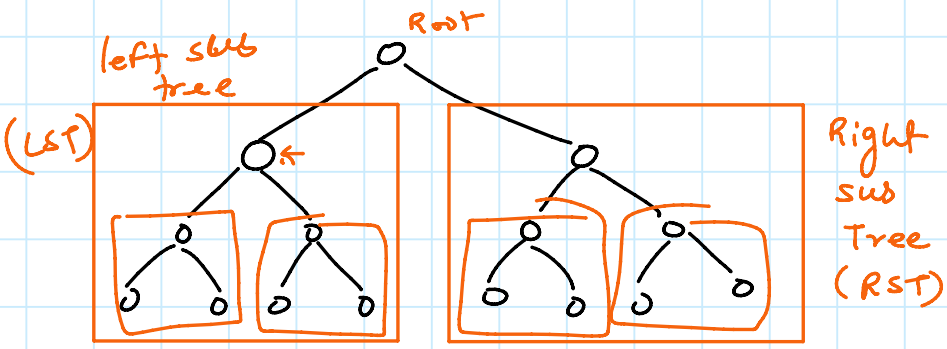
No of Child node → max → 2

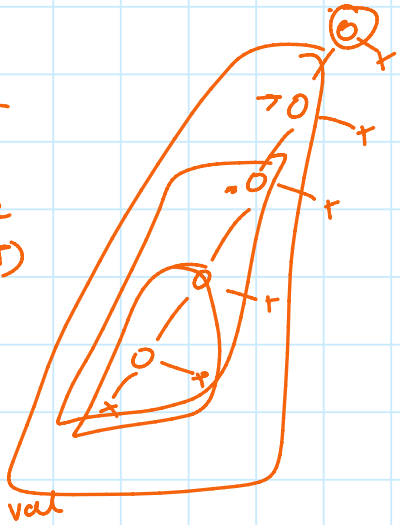


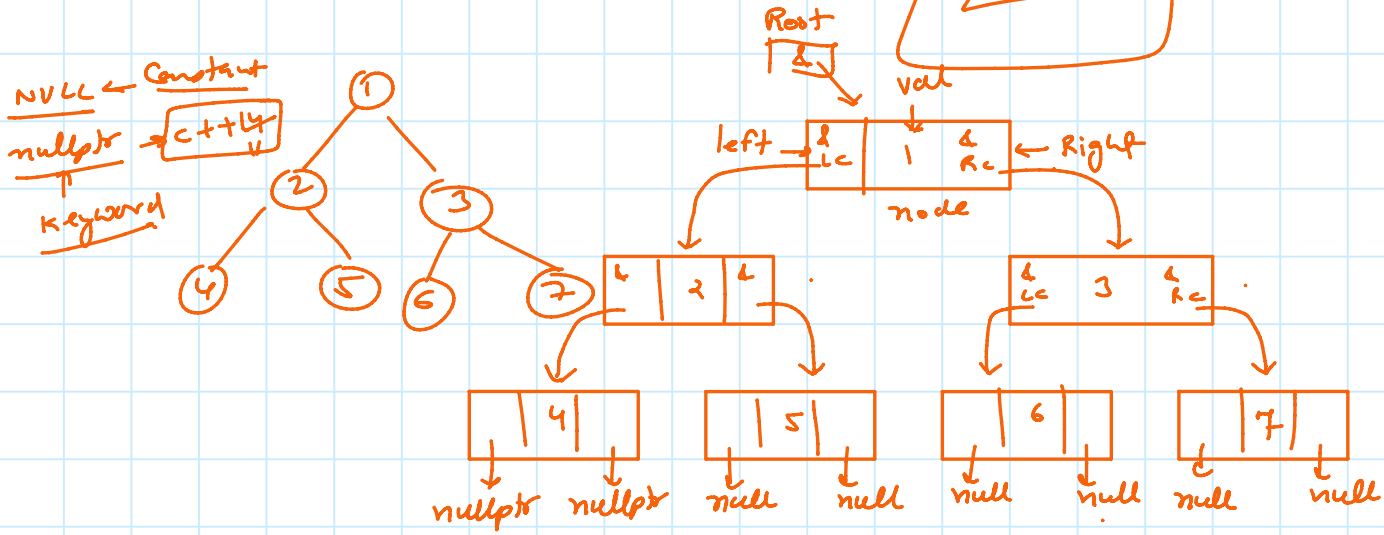


leaf node









① Binary Search tree :- (BST)

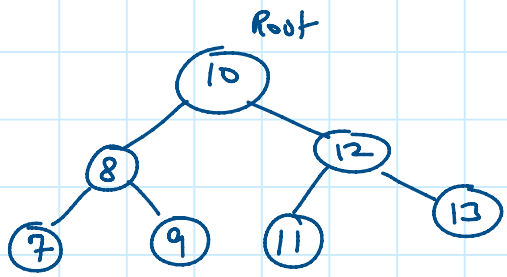
① No of child → max → 2

② left child < root < Right child

Draw a BST :-

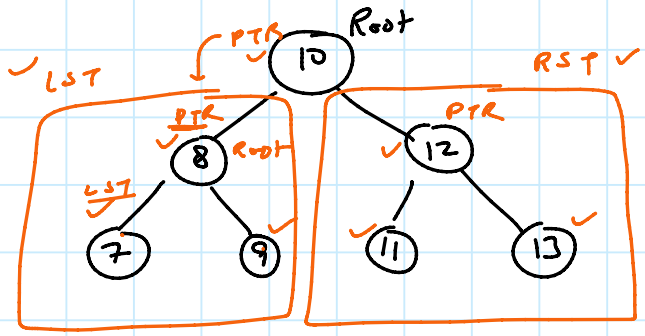
10, 8, 7, 12, 13, 11, 9

10, 8, 7, 12, 13, 11, 9



Traverse → ① Inorder traverse [LST, Root, RST]
② Preorder traverse [Root, LST, RST]
③ Post order traverse [LST, RST, Root]

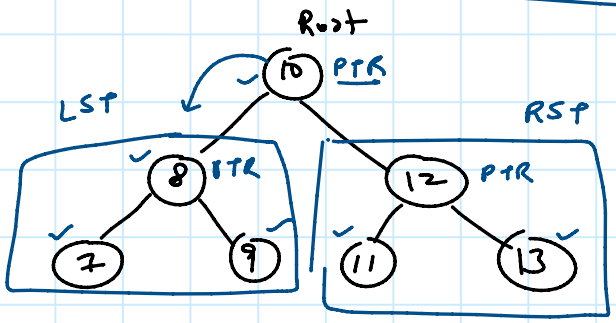
Inorder Traverse :→ [LST, Root, RST]



Sorted

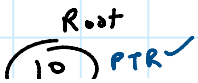
7, 8, 9, 10, 11, 12, 13

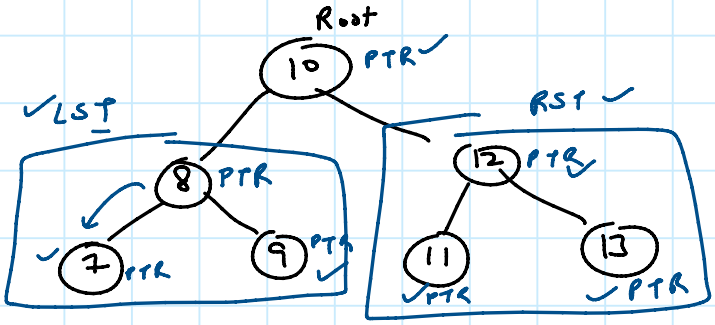
PreOrder traverse :- [Root, LST, RST]



10, 8, 7, 9, 12, 11, 13

PostOrder Traverse [LST, RST, Root]





7, 9, 8, 11, 13, 12, 10

```
Class node
{   public:
        int val;
        node *left, *right;
        node (int no)
        {
            val = no
            left = right = nullptr;
        }
};
```

```
node * root = new node (10)
root -> left = new node (8);
root -> right = new node (12);
root -> left -> left = new node (7);
root -> left -> right = new node (9)
root -> right -> left = new node (11)
root -> right -> right = new node (13)
```