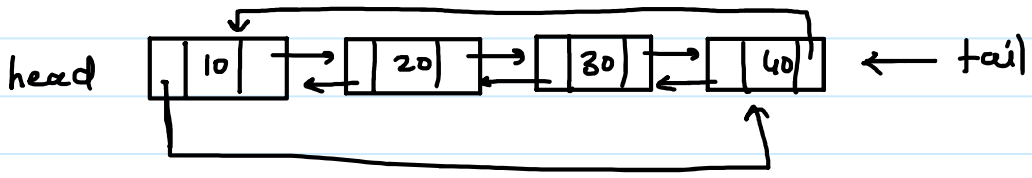
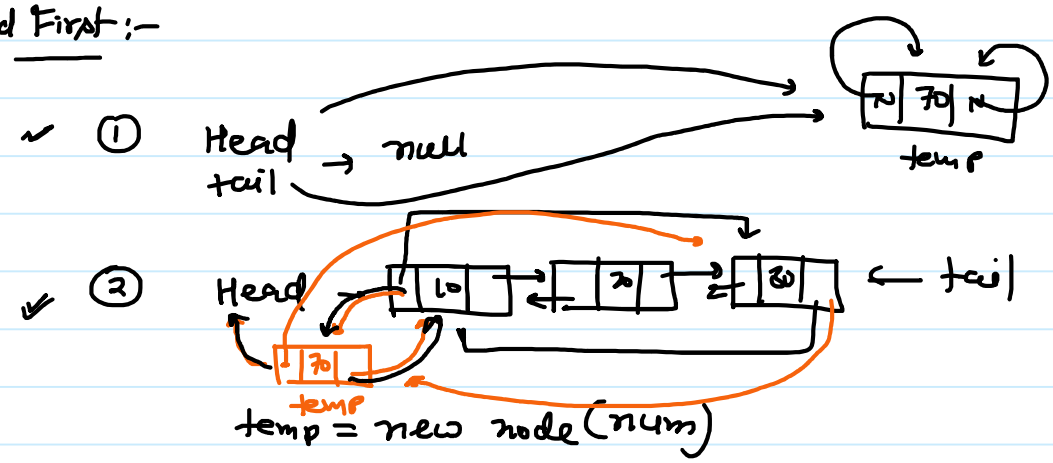
Day 17

05 April 2025 08:32 AM

Circular Doubly linked List :-

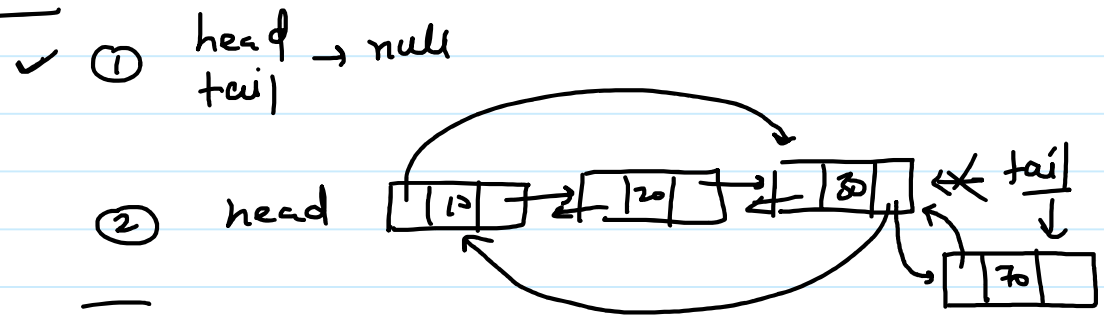


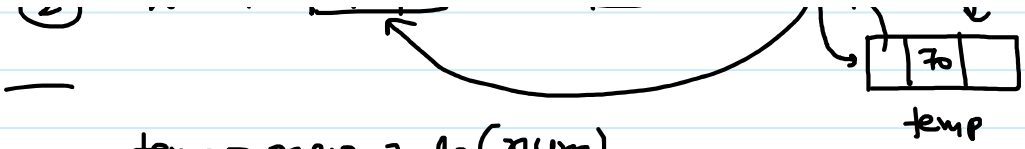
Add First :-




```
① if (head == nullptr)
  {
      head = tail = temp;
      head -> next = head -> prev = head;
  }
  else
  {
      temp -> next = head;
      head -> prev = temp
      head = temp;
      head -> prev = tail
      tail -> next = head
  }
```

Add Last :-





temp = new node(num)

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

② else
{
tail → next = temp
temp → prev = tail
tail = temp
head → prev = tail
tail → next = head
}

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

```
            head = temp;
            head->prev=tail;
            tail->next=head;
        }
    }
    void addLast(int num)
    {
        node *temp = new node(num);
        if(head == nullptr){
            head = tail = temp;
            head->next = head->prev = head;
        }
        else
        {
            temp->prev = tail;
            tail->next = temp;
            tail = temp;
            head->prev=tail;
            tail->next=head;
        }
    }
    void output()
    {
        if(head == nullptr)
        {
            cout<<"Empty List\n";
            return;
        }
        node *ptr = head;
        while(ptr!= tail)
        {
            cout<<ptr->value<<" ";
            ptr = ptr->next;
        }
        cout<<tail->value<<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 != tail && ptr->value != loc)
            ptr = ptr->next;
        if(ptr == tail && ptr->value!=loc)
        {
            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 == tail)
        {
            head = tail = nullptr;
        }
        else{
            head=head->next;
            head->prev = tail;
            tail->next = head;
        }
        cout<<ptr->value<<" deleted\n";
        delete ptr;
    }
    void delLast()
    {
        if(head == nullptr)
        {
            cout<<"Empty List\n";
            return;
        }
        node *ptr = tail;
        if(head == tail)
        {
            head = tail = nullptr;
        }
```

```
            else{
                tail=tail->prev;
                tail->next = head;
                head->prev=tail;
            }
            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()
        {
            if(head==tail)
            {
                return;
            }
            swap(head->value,tail->value);
            node *i=head->next,*j=tail->prev;
            while(i!=j && i->prev!=j)
            {
                swap(i->value,j->value);
                i=i->next;
                j=j->prev;
            }
        }
};
int main()
{
    CircularDoublyLinkedList list;
    list.addFirst(10);
```

```
    list.addFirst(20);
    list.addFirst(30);
    list.addFirst(40);
    list.output();
    list.addLast(60);
    list.addLast(70);
    list.output();
    list.addBefore(100,20);
    list.output();
    list.addAfter(200,20);
    list.output();
    list.delFirst();
    list.output();
    list.delLast();
    list.output();
    list.delNode(20);
    list.output();
    list.reverse();
    list.output();
}
```

Pre defined List → STL → List → Doubly
→ forward_List → Singly

include<List>

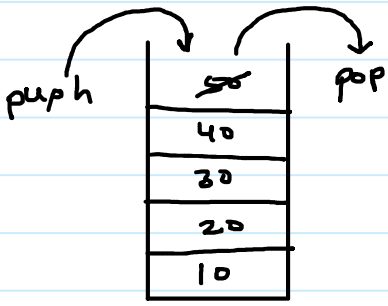
List<int> list1;

```
#include<iostream>
#include<list>
using namespace std;
void output(list<int> &l)
{
    for(int i:l)
        cout<<i<<" ";
    cout<<endl;
}
bool fun(int a)
{
    return a>40;
}
int main()
{
    list<int> l1;
    l1.push_front(10);          //addFirst
```

```
    l1.push_front(20);
    l1.push_front(30);
    output(l1);
    l1.push_back(40);     //addLast
    l1.push_back(50);
    output(l1);
    list<int>::iterator i=l1.begin();
    advance(i,2);
    l1.insert(i,100);
    output(l1);
    l1.erase(i);
    output(l1);
    l1.remove(100);
    output(l1);
    i=l1.begin();
    advance(i,1);
    auto j = l1.begin();
    advance(j,3);
    l1.erase(i,j);
    output(l1);
    l1.assign({10,20,30,40,50,60,70,80,90});
    output(l1);
    l1.remove_if(fun);
    output(l1);
    cout<<l1.front()<<endl;
    cout<<l1.back()<<endl;

}



#include<iostream>
#include<forward_list>
using namespace std;
void output(forward_list<int> &l)
{
    for(int i:l)
        cout<<i<<" ";
    cout<<endl;
}
bool fun(int a)
{
    return a>40;
```

```
}
int main()
{
    forward_list<int> l1;
    l1.push_front(10);
    l1.push_front(20);
    l1.push_front(30);
    l1.push_front(40);
    output(l1);
    auto i=l1.begin();
    advance(i,2);
    l1.insert_after(i,100);
    output(l1);
    l1.assign({100,200,3000,4000});
    output(l1);
    i=l1.begin();
    advance(i,2);
    l1.erase_after(i);
    output(l1);
    l1.pop_front();
    output(l1);
}
```

stack :-



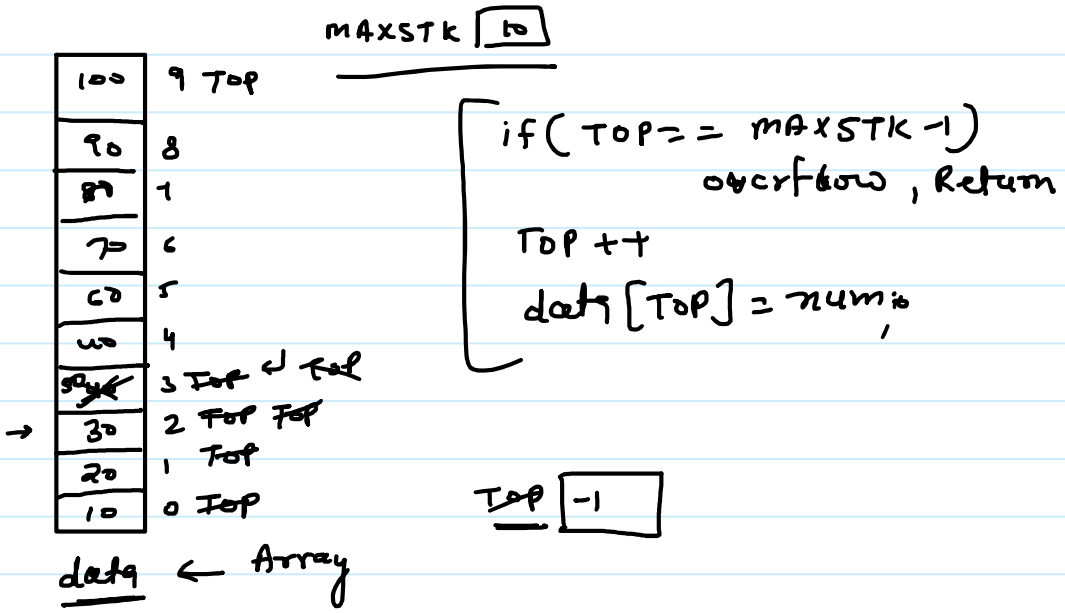
LIFO → List

- ✓ push → Add a new element at top of the stack
- ✓ pop → Remove top most element
- ✓ peek | peep → Display top most element

Overflow → try to add a new element in full stack

Underflow → try to delete an element from empty stack.

① Using Array ② Using Linked List



a+b*c/d+e

postf → abc*d/+e+

pre → ++a/*bcde

s "({}[])({})"

Leetcode-20

Pre defined stack → STL

#include <stack>

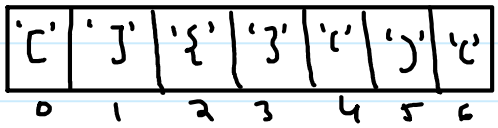
stack<int> s1;

s1.push

s1.pop

s1.top → peep

s = " [ ] { } ( ) ("




```
class Solution {
public:
    bool isValid(string s) {
        stack<char> stk;
        for(char i:s)
        {
            if(i=='[' || i=='(' || i=='{')
                stk.push(i);
            else
            {
                if(stk.empty() || i==')' and stk.top()!='(' || i=='}' and stk.top()!
='{'|| i==']' and stk.top()!='[')
                    return false;
                else
                    stk.pop();
            }
        }
        return stk.empty();

    }
};
```

```
#include<iostream>
using namespace std;
#define MAXSTK 10
class Mystack{
    int data[MAXSTK];
    int top;
    public:
    Mystack()
    {
        top=-1;
    }
    void push(int num)
    {
```

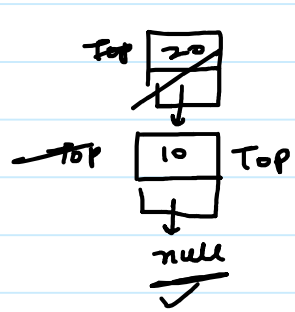
```
        if(top == MAXSTK-1)
        {
            cout<<"Overflow\n";
            return;
        }
        top++;
        data[top]=num;
    }
    void pop()
    {
        if(top == -1)
        {
            cout<<"Underflow\n";
            return;
        }
        top--;
    }
    int peep()
    {
        if(top == -1)
        {
            cout<<"Underflow\n";
            return 0;
        }
        return data[top];
    }
    bool isEmpty()
    {
        if(top == -1)
            return true;
        return false;
    }
};
int main()
{
    Mystack s1;
    s1.push(10);
    s1.push(20);
    s1.push(30);
    s1.push(40);
    s1.push(50);
    s1.push(60);
    s1.push(70);
    s1.push(80);
    s1.push(90);
```

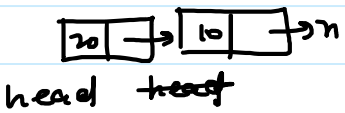
```
    s1.push(100);
    s1.push(10);
    // cout<<s1.peep()<<endl;
    // s1.pop();
    // cout<<s1.peep()<<endl;
    while(! s1.isEmpty())
    {
        cout<<s1.peep()<<endl;
        s1.pop();
    }
}
```

```
#include<iostream>
#include<stack>
using namespace std;
int main()
{
    stack<int> s1;
    s1.push(10);
    s1.top();
    s1.pop();

}
```

stack using Linked List

Push → Add First
pop → Del First
~~top~~

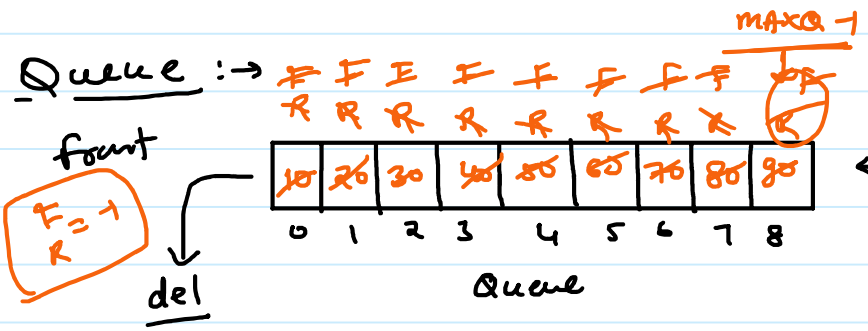





```
#include<iostream>
using namespace std;
class node{
    public:
        int value;
```

```
        node *next;
        node(int x)
        {
            value = x;
            next=nullptr;
        }
};
class Mystack{
    node *top;
    public:
        Mystack()
        {
            top=nullptr;
        }
        void push(int num)
        {
            node *temp = new node(num);
            temp->next = top;
            top = temp;
        }
        void pop()
        {
            if(top == nullptr)
            {
                cout<<"underflow\n";
                return;
            }
            node * ptr=top;
            top = top->next;
            delete ptr;
        }
        bool isEmpty()
        {
            if(top = nullptr)
                return true;
            return false;
        }
        int peep()
        {
            if(top == nullptr)
            {
                cout<<"Underflow\n";
                return 0;
            }
            return top->value;
        }
};
int main()
```

```
{
    Mystack s;
    s.push(10);
    s.push(20);
    s.push(30);
    s.push(40);
    cout<<s.peep()<<endl;
    s.pop();
    cout<<s.peep()<<endl;
}
```



Operations

① Add Queue

enqueue

② Del Queue

dequeue()