**Stack**

**Stack: “** A **Stack** is a linear data structure in which element can **be inserted and deleted** only from **one side of the list,** called **top() ”**. A **Stack** follows the **LIFO (Last In First Out)** Mechanism.

**push(x)** => insert value, **pop()**=> delete(top), **top(), empty()**

#define n 100       // random no.

class Stack{

    int\* arr;

    int top;

    public:

        Stack(){

            arr = new int[n];

            top = -1;

        }

        void push(int x){

            if(top==n-1){

                cout<<"stack overflow"<<endl;

                return;

            }

            top++;

            arr[top]=x;

        }

        void pop(){

            if(top==-1){

                cout<<"No element to pop"<<endl;

                return;

            }

            top--;

        }

        int Top(){        // int value pass

            if(top==-1){

                cout<<"No element to Top"<<endl;

                return -1;

            }

            return arr[top];

        }

        bool empty(){

            return top==-1;

        }

};

int main(){

    Stack st;

    st.push(1);  st.push(2);  st.push(3);

    cout<<st.Top()<<endl;

    st.pop();

    cout<<st.Top()<<endl;

    st.pop();  st.pop();  st.pop();

    cout<<st.empty()<<endl;

    return 0;

}

**Output:**

3

2

No element to pop

1

**Reverse Sentence:**

#include<bits/stdc++.h>

using namespace std;

void ReverseSentance(string s){

    stack<string> st;

    for(int i=0; i<s.length(); i++){

        string word = "";

        while(s[i] != ' ' && i<s.length()){

            word += s[i];

            i++;

        }

        st.push(word);

    }

    while(!st.empty()){

        cout<<st.top()<<" ";

        st.pop();

    }

    cout<<endl;

}

int main(){

    string s = "Hey, how are you doing?";

    ReverseSentance(s);

}

**Output:**

doing? you are how Hey,

**Reverse a Stack:**

void insertAtBottom(stack<int> &st, int ele){

    if(st.empty()){

        st.push(ele);

        return;

    }

    int topele = st.top();

    st.pop();

    insertAtBottom(st,ele);

    st.push(topele);

}

void reverse(stack<int> &st){

    if(st.empty()){

        return;

    }

    int ele = st.top();

    st.pop();

    reverse(st);

    insertAtBottom(st,ele);

}

int main(){

    stack <int> st;

    st.push(1);  st.push(2);

    st.push(3);  st.push(4);

    reverse(st);

    while(!st.empty()){

        cout<<st.top()<<" ";

        st.pop();

    }

    cout<<endl;

    return 0;

}

**Output:**

1 2 3 4

**Queue**

**Queue:** “A **Queue** is a linear data structure in which element can be **inserted only from one side** of the list called **rear()**, and the element can be **deleted only from the other** side called the **front()**”

A **Queue** follows the **FIFO (First In First Out)** Mechanism.

**Queue using two Stack with costly enQueue()**

struct Queue{

    stack<int> s1, s2;

    void enQueue(int x){

        // Move all element from s1 to s2

        while(!s1.empty()){

            s2.push(s1.top());

            s1.pop();

        }

        s1.push(x);

        // Push everything back to s1

        while(!s2.empty()){

            s1.push(s2.top());

            s2.pop();

        }

    }

    int deQueue(){

        if(s1.empty()){

            cout<<"Q is Empty";

            exit(0);

        }

        // return top of s1

        int x = s1.top();

        s1.pop();

        return x;

    }

};

**Queue using One Stack and Recursive call Stack.**

struct Rqueue{

    stack<int> s;

    void enQueue(int x){

        s.push(x);

    }

    int deQueue(){

        if(s.empty()){

            cout<<"Q is empty";

            exit(0);

        }

        int x = s.top();

        s.pop();

//stack becomes empty, return the popped item.

        if(s.empty()){

            return x;

        }

        int item = deQueue();

        s.push(x);

        return item;

    }

};

int main(){

    Queue q;

    q.enQueue(1); q.enQueue(2); q.enQueue(3);

    cout<<q.deQueue()<<endl;

    cout<<q.deQueue()<<endl;

    cout<<q.deQueue()<<"\n-------"<<endl;

    Rqueue r;

    r.enQueue(1); r.enQueue(2); r.enQueue(3);

    cout<<r.deQueue()<<endl;

    cout<<r.deQueue()<<endl;

    cout<<r.deQueue()<<endl;

    return 0;

}