STL STACK

```
#include <bits/stdc++.h>
using namespace std;

void showstack(stack <int> s)
{
	while (!s.empty())
	{
		cout << '\t' << s.top()<<" ";
		s.pop();
	}
	cout << '\n';
}

int main ()
{
	stack <int> s;
	int ch,n;
	do
	{
		cout<<"Which operation do u want to perform? "<<endl;
		cout<<"1.Push"<<endl<<"2.Pop"<<endl<<"3.size"<<endl<<"4.Top
Element"<<endl<<"5.Print Stack"<<endl;
		cin>>ch;
		switch(ch)
		{
			case 0:
				break;
			case 1:
				cout<<"Enter the element u want to push"<<endl;
				cin>>n;
				s.push(n);
				break;
			case 2:
				s.pop();
				break;
			case 3:
				cout<<s.size()<<endl;
				break;
			case 4:
```

```
                cout<<s.top()<<endl;                                    1
                break;                                                  2
            case 5:
                showstack(s);
                break;
            default:
                cout<<"Invalid Option"<<endl;

        }

    }while(ch!=0);

    return 0;
}
```

STACK USING ARRAY

```
#include<iostream>
using namespace std;
#define N 10

template<class T>
class stack
{
    public:
        int top;
        T item;
        T arr[N];

        stack()
        {
            top=-1;
        }
        bool isempty()
        {
            if(top==-1)
                return true;
            return false;
        }

        bool isfull()
        {
            if(top==N-1)
```

```
                return true;
        return false;                                                    3
    }

    void push(T x)
    {
        if(isfull())
        {
            cout<<"can't push "<<endl<<"stack is in overflow
state"<<endl;
        }
        else
        {
            arr[++top]=x;
        }
    }

    T pop()
    {
        if(isempty())
        {
            cout<<"can't pop"<<endl<<"stack is in underflow
state"<<endl;
            return 0;
        }
        else
        {
            return arr[top--];
        }
    }

    T topelement()
    {
        if(isempty())
            cout<<"stack is empty"<<endl;
        else
            return arr[top];
    }

    int sizeofstack()
    {
        int k = top+1;
```

```
            return k;
        }                                                                    4

        void printstack()
        {
            if(isempty())
            {
                cout<<"stack is empty"<<endl;
            }
            else
            {
                for(int i=top;i>=0;i--)
                {
                    cout<<arr[i]<<endl;
                }
            }
        }

};


int main()
{
    int option;
    stack <int> k;
    do
    {
        cout<<"which operation do u want to perform"<<endl;
        cout<<"1. to push a element"<<endl;
        cout<<"2. to pop a element"<<endl;
        cout<<"3. size of the stack"<<endl;
        cout<<"4. top element"<<endl;
        cout<<"5. print stack"<<endl;
        cin>>option;

        switch(option)
        {
            case 0:
                break;
            case 1:
                cout<<"enter the element to push into the stack"<<endl;
                cin>>k.item;
```

```
                    k.push(k.item);
                    break;                                                    5
            case 2:
                    cout<<"popped element is "<<k.pop()<<endl;
                    break;
            case 3:
                    cout<<"size of the stack is "<<k.sizeofstack()<<endl;
                    break;
            case 4:
                    cout<<"top element of the stack is
"<<k.topelement()<<endl;
                    break;
            case 5:
                    cout<<"stack consists"<<endl;
                    k.printstack();
                    break;
            default:
                    cout<<"enter position number from 1-5"<<endl;
        }

    }while(option!=0);

    return 0;
}
```

STACK USING SINGLE LINKED LIST

```
#include<iostream>
using namespace std;

struct node
{
    public:
    int data;
    node *next;
};
template <class T>
class StackLL
{
    node *last;
    int noe;
    public:
    StackLL() {last=NULL;noe=0;}
```

```
        void push(T x);
        T pop();                                                    6
        T topElement();
        int sizeofStack();
        void display();
};
template <class T>
int StackLL<T>::sizeofStack()
{
        return noe;
}
template <class T>
T StackLL<T>::topElement()
{
        if(last!=NULL)
                return last->next->data;
}
template <class T>
void StackLL<T>::push(T x)
{
        node *temp=new node;
        temp->data=x;
        temp->next=NULL;
        if(last==NULL)
        {
                last=temp;
                last->next=last;
        }
        else
        {
                temp->next=last->next;
                last->next=temp;
        }
        noe++;
}
template <class T>
T StackLL<T>::pop()
{
        node *temp;
        T x;
        if(last==NULL)    //When there are no elements in list
                cout<<endl<<"LIST IS EMPTY";
```

```
        else if(last->next==last) //when there is single element in list
        {                                                                    7
                temp=last;
                last=NULL;
                x=temp->data;
                delete temp; noe--;
                return x;
        }
        else
        {
                temp=last->next;
                last->next=temp->next;
                x=temp->data;
                delete temp; noe--;
                return x;
        }
}
template <class T>
void StackLL<T>::display()
{
        node *p;
        p=last->next;
        while(p!=last)
        {
                cout<<"\t\t"<<p->data<<endl;
                p=p->next;
        }
        cout<<"\t\t"<<p->data<<endl;
}
int main()
{
        int ch,x;
        StackLL<int> ob;
        while(1)
        {
                cout<<endl<<"1.Push"<<endl<<"2.Pop"<<endl<<"3.Size";
                cout<<endl<<"4.Top
element"<<endl<<"5.Display"<<endl<<"6.Exit"<<endl<<"Enter Your
Choice:";
                cin>>ch;
                switch(ch)
                {
```

```
                case 1:cout<<endl<<"Enter an element to push:";
                        cin>>x;                                                 8
                        ob.push(x);
                        break;
                case 2:     cout<<endl<<"Popped element is:"<<ob.pop();
                        break;
                case 3:cout<<endl<<"Size of stack is:"<<ob.sizeofStack();
                        break;
                case 4:cout<<endl<<"Top Elements is:"<<ob.topElement();
                        break;
                case 5:cout<<endl<<"The stack is:"<<endl;
                        ob.display();
                        break;
                case 6:return 0;
                default:cout<<endl<<"WRONG CHOICE";
            }
        }
}
```

MATCHING PARANTHESIS

```
#include<iostream>
#include<string>
using namespace std;
#define N 10

template<class T>
class stack
{
        public:
                int top;
                T item;
                T arr[N];

                stack()
                {
                        top=-1;
                }
                void balanced_paranthesis(string str);
                bool isempty()
                {
                        if(top==-1)
                                return true;
```

```
        return false;
    }

    bool isfull()
    {
        if(top==N-1)
            return true;
        return false;
    }

    void push(T x)
    {
        if(isfull())
        {
            cout<<"can't push "<<endl<<"stack is in overflow
state"<<endl;
        }
        else
        {
            arr[++top]=x;
        }
    }

    T pop()
    {
        if(isempty())
        {
            cout<<"can't pop"<<endl<<"stack is in underflow
state"<<endl;
            return 0;
        }
        else
        {
            return arr[top--];
        }
    }

    T topelement()
    {
        if(isempty())
            cout<<"stack is empty"<<endl;
        else
```

```
                return arr[top];
        }

        int sizeofstack()
        {
                int k = top+1;
                return k;
        }

        void printstack()
        {
                if(isempty())
                {
                        cout<<"stack is empty"<<endl;
                }
                else
                {
                        for(int i=top;i>=0;i--)
                        {
                                cout<<arr[i]<<endl;
                        }
                }
        }

};


balanced_parenthesis(string str)
{
        stack <char> p;
        string x;
        int a=0;
        for(int i=0;i<str.length();i++)
        {
                if( str[i] == '(' || str[i] == '[' || str[i] == '{' || str[i] =='<' )
                {
                        p.push(str[i]);
                }
                else if(str[i] == ')')
                {
                        if(p.topelement()=='(')
                        {
```

```
                x[a++]=p.pop();
            }
            else
                break;
        }
        else if(str[i] == ']')
        {
            if(p.topelement()=='[')
            {
                x[a++]=p.pop();
            }
            else
                break;
        }
        else if(str[i] == '}')
        {
            if(p.topelement()=='{')
            {
                x[a++]=p.pop();
            }
            else
                break;
        }
        else
        {
            if(p.topelement() == '<')
            {
                x[a++]=p.pop();
            }
            else
                break;
        }
    }
    if(p.isempty())
    {
        cout<<"paranthesis balanced"<<endl;
    }
    else
    {
        cout<<"paranthesis unbalanced"<<endl;
    }
}
```

```
int main()                                                              12
{                                                                       13
        string s;
        cout<<"enter the string to check whether its balanced or not"<<endl;
        cin>>s;
        balanced_parenthesis(s);

        return 0;
}
```

STL QUEUE

```
#include <iostream>
#include <queue>
using namespace std;
void showq(queue<int> gq)
{
   queue<int> g = gq;
   while (!g.empty()) {
      cout << '\t' << g.front();
      g.pop();
   }
   cout << '\n';
}
int main()
{
        queue<int> q;
        int option,n;
        do
        {
                cout<<"which operation do u want to perform"<<endl;
                cout<<"1.enqueue"<<endl;
                cout<<"2.dequeue"<<endl;
                cout<<"3.firstelement"<<endl;
                cout<<"4.lastelement"<<endl;
                cout<<"5.sizeofqueue"<<endl;
                cout<<"6.printqueue"<<endl;

                cin>>option;

                switch(option)
                {
```

```
                case 0:
                        break;
                case 1:
                        cout<<"Enter element to push into queue"<<endl;
                        cin>>n;
                        q.push(n);
                        break;
                case 2:
                        q.pop();
                        break;
                case 3:
                        cout<<"the first element in the queue is
"<<q.front()<<endl;
                        break;
                case 4:
                        cout<<"the last element in the queue is
"<<q.back()<<endl;
                        break;
                case 5:
                        cout<<"the size of the queue is "<<q.size()<<endl;
                        break;
                case 6:
                        cout<<"the elements in the queue are"<<endl;
                        showq(q);
                        break;
                default:
                        cout<<"enter valid option"<<endl;
        }

    }while(option!=0);
    return 0;
}
```

QUEUE USING ARRAY

```
#include<iostream>
using namespace std
#define N 20

;class queue
{
    public:
```

```
int front,rear,arr[N];
queue()
{
        front=0;
        rear=-1;
}

bool isempty()
{
        if(front>rear)
                return true;
        return false;
}

bool isfull()
{
        if(rear==N-1)
                return true;
        return false;
}

void enqueue()
{
        if(isfull())
        {
                cout<<"the queue is full"<<endl;
        }
        else
        {
                int value;
                cout<<"enter a value to enqueue"<<endl;
                cin>>value;
                arr[++rear]=value;
        }
}

void dequeue()
{
        if(isempty())
        {
                cout<<"the queue is empty"<<endl<<"nothing to
dequeue"<<endl;
```

```
        }
        else
        {
            front=front+1;
        }
    }

    int firstelement()
    {
        if(isempty())
        {
            cout<<"the queue is empty"<<endl;
        }
        else
        {
            return arr[front];
        }
    }

    int lastelement()
    {
        if(isempty())
        {
            cout<<"the queue is empty"<<endl;
        }
        else
        {
            return arr[rear];
        }
    }

    int sizeofqueue()
    {
        return rear-front+1;
    }

    void display()
    {
        if(isempty())
        {
            cout<<"the queue is empty"<<endl<<"nothing to
display"<<endl;
```

```
                }
                else                                                    17
                {
                        int n=front;
                        while(n!=rear)
                        {
                                cout<<arr[n]<<"\t";
                                n++;
                        }
                        cout<<arr[rear];
                }
        }



};

int main()
{
        int option;
        queue q;

        do
        {
                cout<<"which operation do u want to perform"<<endl;
                cout<<"1.enqueue"<<endl;
                cout<<"2.dequeue"<<endl;
                cout<<"3.firstelement"<<endl;
                cout<<"4.lastelement"<<endl;
                cout<<"5.sizeofqueue"<<endl;
                cout<<"6.printqueue"<<endl;

                cin>>option;

                switch(option)
                {
                        case 0:
                                break;
                        case 1:
                                q.enqueue();
                                break;
                        case 2:
```

```
                        q.dequeue();
                        break;                                          18
                case 3:
                        cout<<"the first element in the queue is
"<<q.firstelement()<<endl;
                        break;
                case 4:
                        cout<<"the last element in the queue is
"<<q.lastelement()<<endl;
                        break;
                case 5:
                        cout<<"the size of the queue is
"<<q.sizeofqueue()<<endl;
                        break;
                case 6:
                        cout<<"the elements in the queue are"<<endl;
                        q.display();
                        break;
                default:
                        cout<<"enter valid option"<<endl;
            }

        }while(option!=0);

        return 0;

}
```

STL DEQUEUE

```
#include <iostream>
#include <deque>
using namespace std;
void showdq(deque <int> g)
{
        deque <int> :: iterator it;
        for (it = g.begin(); it != g.end(); ++it)
                cout << '\t' << *it;
        cout << '\n';
}

int main()
```

```
{
        deque <int> q1;                                                    19
        int option,n;
        do
        {
                cout<<"which operation do u want to perform"<<endl;
                cout<<"1.enqueue at front"<<endl;
                cout<<"2.enqueue at rear"<<endl;
                cout<<"3.dequeue at front"<<endl;
                cout<<"4.dequeue at rear"<<endl;
                cout<<"5.size of queue"<<endl;
                cout<<"6.first element of queue"<<endl;
                cout<<"7.last element of queue"<<endl;
                cout<<"8.printing queue"<<endl;

                cin>>option;

                switch(option)
                {
                        case 0:
                                break;
                        case 1:
                                cout<<"Enter an element to enqueue at front"<<endl;
                                cin>>n;
                                q1.push_front(n);
                                break;
                        case 2:
                                cout<<"Enter an element to enqueue at rear"<<endl;
                                cin>>n;
                                q1.push_back(n);
                                break;
                        case 3:
                                cout<<"dequeue operation at front"<<endl;
                                q1.pop_front();
                                break;
                        case 4:
                                cout<<"dequeue operation at rear"<<endl;
                                q1.pop_back();
                                break;
                        case 5:
                                cout<<"size of the queue is"<<q1.size();
                                break;
```

```
                case 6:
                        cout<<"first element of the queue                    20
is"<<endl<<q1.front()<<endl;
                        break;
                case 7:
                        cout<<"last element of the queue
is"<<endl<<q1.back()<<endl;
                        break;
                case 8:
                        showdq(q1);
                        break;
                default:
                        cout<<"Enter valid option"<<endl;

        }

    }while(option!=0);


    return 0;
}
```

DEQUEUE USING ARRAY

```
#include<iostream>
using namespace std;
#define N 20

class queue
{
    public:
        int front,rear,*arr;
        queue()
        {
            front=-1;
            rear=-1;
            arr=new int[N];
        }

        bool isempty()
        {
            if(front==-1 && rear==-1)
```

```
                return true;
        else
                return false;
}

bool isfull()
{
        if(front==0 && rear==N-1)
                return true;
        else
                return false;
}

void enqueue_front()
{
        int value;
        cout<<"enter value to be inserted before front"<<endl;
        cin>>value;
        if(front==-1)
        {
                front=0;
                arr[++rear]=value;
                cout<<"element inserted at empty queue"<<endl;
        }
        else if(front!=0)
        {
                arr[--front]=value;
                cout<<"element inserted before front"<<endl;
        }
        else
        {
                cout<<"cant insert! \t queue is in underflow
condition"<<endl;
        }
}

void enqueue_rear()
{
        int value;
        cout<<"enter value to be inserted after rear"<<endl;
        cin>>value;
        if(rear>=N-1)
```

```
		{
			cout<<"can't insert! \t queue is in overflow condition"<<endl;
		}
		else if(front==-1)
		{
			front=0;
			arr[++rear]=value;
			cout<<"element inserted in empty queue"<<endl;
		}
		else
		{
			arr[++rear]=value;
			cout<<"element inserted after rear"<<endl;
		}
	}

	int dequeue_front()
	{
		if(isempty())
		{
			cout<<"the queue is empty, cant delete element"<<endl;
		}
		else
		{
			int x = arr[front];
			if(front==rear)
			{
				front=rear=-1;
			}
			else
			{
				front++;
			}
			return x;
		}
	}

	int dequeue_rear()
	{
		if(isempty())
		{
```

```
		cout<<"the queue is empty! cant delete element"<<endl;
	}
	else
	{
		int x = arr[rear];
		if(front==rear)
		{
			front=rear=-1;
		}
		else
		{
			rear--;
		}
		return x;
	}
}

int first_element()
{
	return arr[front];
}

int last_element()
{
	return arr[rear];
}

int sizeofqueue()
{
	return rear-front+1;
}

void print_queue()
{
	if(isempty())
	{
		cout<<"the queue is empty\t nothing to display"<<endl;
	}
	else
	{
		int i=front;
		cout<<"the queue is"<<endl;
```

```
                while(i<=rear)
                {
                        cout<<arr[i]<<"\t\t";
                        i++;
                }
                cout<<endl;
        }
    }


};

int main()
{
    queue q1;
    int option;
    do
    {
        cout<<"which operation do u want to perform"<<endl;
        cout<<"1.enqueue at front"<<endl;
        cout<<"2.enqueue at rear"<<endl;
        cout<<"3.dequeue at front"<<endl;
        cout<<"4.dequeue at rear"<<endl;
        cout<<"5.size of queue"<<endl;
        cout<<"6.first element of queue"<<endl;
        cout<<"7.last element of queue"<<endl;
        cout<<"8.printing queue"<<endl;

        cin>>option;

        switch(option)
        {
            case 0:
                break;
            case 1:
                cout<<"enqueue operation at front"<<endl;
                q1.enqueue_front();
                break;
            case 2:
                cout<<"enqueue operation at rear"<<endl;
                q1.enqueue_rear();
                break;
```

```
                case 3:
                        cout<<"dequeue operation at front"<<endl;
                        q1.dequeue_front();
                        break;
                case 4:
                        cout<<"dequeue operation at rear"<<endl;
                        q1.dequeue_front();
                        break;
                case 5:
                        cout<<"size of the queue is"<<q1.sizeofqueue();
                        break;
                case 6:
                        cout<<"first element of the queue
is"<<endl<<q1.first_element()<<endl;
                        break;
                case 7:
                        cout<<"last element of the queue
is"<<endl<<q1.last_element()<<endl;
                        break;
                case 8:
                        q1.print_queue();
                        break;
                default:
                        cout<<"Enter valid option"<<endl;

        }

    }while(option!=0);

    return 0;

}
```


DEQUEUE USING DOUBLE LINKED LIST

```
#include<iostream>
using namespace std;

struct node
{
    int data;
    node* prev;
```

```
        node* next;
};
class DEQ
{
        public:
                node *front, *rear;
                DEQ()
                {
                        front=rear=NULL;
                }

                bool isempty()
                {
                        if(front==NULL && rear==NULL)
                                return true;
                        else
                                return false;
                }

                void enqueue_begin()
                {
                        node *temp = new node;
                        temp->prev=temp->next=NULL;
                        cout<<"enter the data of the node to insert at beginning of the
queue"<<endl;
                        cin>>temp->data;
                        if(front==NULL && rear==NULL)
                        {
                                front=temp;
                                rear=temp;
                                cout<<"node inserted in empty queue"<<endl;
                        }
                        else
                        {
                                front->prev=temp;
                                temp->next=front;
                                front=temp;
                                cout<<"node inserted at beginning"<<endl;
                        }
                }

                void enqueue_end()
```

```
{
        node *temp = new node;
        temp->prev=temp->next=NULL;
        cout<<"enter the data of the node to insert at end of the
queue"<<endl;
        cin>>temp->data;
        if(front==NULL && rear==NULL)
        {
                front=temp;
                rear=temp;
                cout<<"node inserted in empty queue"<<endl;
        }
        else
        {
                rear->next=temp;
                temp->prev=rear;
                rear=temp;
                cout<<"node inserted at end"<<endl;
        }
}

void dequeue_begin()
{
        node *temp=new node;
        temp=front;
        if(isempty())
        {
                cout<<"the queue is empty! nothing to delete"<<endl;
        }
        else if(front==rear && front!=NULL)
        {
                delete front;
                front=rear=NULL;
        }
        else
        {
                front=front->next;
                front->prev=NULL;
                delete temp;
                cout<<"the first node of the queue is deleted"<<endl;
        }
}
```

```
void dequeue_end()
{
	node *temp=new node;
	temp=rear;
	if(isempty())
	{
		cout<<"the queue is empty! nothing to delete"<<endl;
	}
	else if(front==rear)
	{
		delete front;
		front=rear=NULL;
	}
	else
	{
		rear=rear->prev;
		rear->next=NULL;
		delete temp;
		cout<<"the last node of the queue is deleted"<<endl;
	}
}

int sizeofqueue()
{
	node *temp=new node;
	temp=front;
	int i;
	for(i=1;temp->next!=NULL;i++)
	{
		temp=temp->next;
	}
	return i;
}

int first_element()
{
	return front->data;
}

int last_element()
{
```

```
            return rear->data;
        }

        void print_queue()
        {
            node *temp=new node;
            temp=front;
            if(front==NULL && rear==NULL)
            {
                cout<<"the queue is empty"<<endl;
            }
            else
            {
                while(temp->next!=NULL)
                {
                    cout<<temp->data<<"<=>";
                    temp=temp->next;
                }
                cout<<temp->data<<endl;
            }
        }

};

int main()
{
    DEQ q1;
    int option;
    do
    {
        cout<<"which operation do u want to perform"<<endl;
        cout<<"1.enqueue at beginning"<<endl;
        cout<<"2.enqueue at end"<<endl;
        cout<<"3.dequeue at beginning"<<endl;
        cout<<"4.dequeue at end"<<endl;
        cout<<"5.size of queue"<<endl;
        cout<<"6.first element of queue"<<endl;
        cout<<"7.last element of queue"<<endl;
        cout<<"8.printing queue"<<endl;

        cin>>option;
```

```
        switch(option)
        {
            case 0:
                break;
            case 1:
                cout<<"enqueue operation at front"<<endl;
                q1.enqueue_begin();
                break;
            case 2:
                cout<<"enqueue operation at rear"<<endl;
                q1.enqueue_end();
                break;
            case 3:
                cout<<"dequeue operation at front"<<endl;
                q1.dequeue_begin();
                break;
            case 4:
                cout<<"dequeue operation at rear"<<endl;
                q1.dequeue_end();
                break;
            case 5:
                cout<<"size of the queue is"<<q1.sizeofqueue()<<endl;
                break;
            case 6:
                cout<<"first element of the queue
is"<<endl<<q1.first_element()<<endl;
                break;
            case 7:
                cout<<"last element of the queue
is"<<endl<<q1.last_element()<<endl;
                break;
            case 8:
                q1.print_queue();
                break;
            default:
                cout<<"give valid option"<<endl;

        }

    }while(option!=0);
    return 0;
}
```

```
#include <iostream>
#include <list>
#include <iterator>
using namespace std;
void showlist(list <int> g)
{
	list <int> :: iterator it;
	for(it = g.begin(); it != g.end(); ++it)
		cout << '\t' << *it;
	cout << '\n';
}

int main()
{

	list <int> s;

	int option,position,n;
	do
	{
		cout<<"which operation do u want to perform"<<endl;
		cout<<"1.insert node at begin"<<endl;
		cout<<"2.insert node at end"<<endl;
		cout<<"3.deletenode at front"<<endl;
		cout<<"4.delete node at end"<<endl;
		cout<<"5.front element"<<endl;
		cout<<"6.last element"<<endl;
		cout<<"7.reverse a list"<<endl;
		cout<<"8.printlist"<<endl;

		cin>>option;

		switch(option)
		{
			case 0:
				break;
			case 1:
				cout<<"Enter element to insert"<<endl;
				cin>>n;
				s.push_front(n);
```

```
                break;
            case 2:
                cout<<"Enter element to insert"<<endl;
                cin>>n;
                s.push_back(n);
                break;
            case 3:
                s.pop_front();
                break;
            case 4:
                s.pop_back();
                break;
            case 5:
                cout<<s.front()<<endl;
                break;
            case 6:
                cout<<s.back()<<endl;
                break;
            case 7:
                s.reverse();
                break;
            case 8:
                showlist(s);
                break;
            default:
                cout<<"Enter valid option"<<endl;
        }

    }while(option!=0);

    return 0;

}
```

ITERATORS

```
// C++ code to demonstrate the working of
// iterator, begin() and end()
#include<iostream>
#include<iterator> // for iterators
#include<vector> // for vectors
using namespace std;
```

```
int main()
{
	vector<int> ar = { 1, 2, 3, 4, 5 };

	// Declaring iterator to a vector
	vector<int>::iterator ptr;

	// Displaying vector elements using begin() and end()
	cout << "The vector elements are : ";
	for (ptr = ar.begin(); ptr < ar.end(); ptr++)
		cout << *ptr << " ";

	return 0;
}
```

REVERSING VECTOR USING STACK

```
#include<iostream>
#include<vector>
#include<bits/stdc++.h>
using namespace std;
void reverse(vector<int>& V)
{
// reverse a vector
	stack<int> S[V.size()];
	for (int i = 0; i < V.size(); i++) // push elements onto stack
		S.push(V[i]);
	for (int i = 0; i < V.size(); i++)
	{ // pop them in reverse order
		V[i] = S.top();
		S.pop();
	}
}
int main()
{
	vector<int> s1;
	s1.push_back(20);
	s1.push_back(10);
	s1.push_back(89);
	s1.push_back(90);
	return 0;
}
```