* **Stack implementation using array**

**Code**

**#include<iostream>**

**using namespace std;**

**class Stack**

**{**

**public:**

**int stack[100], val;**

**int num, top;**

**Stack()**

**{**

**num=100;**

**top=-1;**

**}**

**void Insert()**

**{**

**if(top >= num-1)**

**{**

**cout<<"Stack overflow";**

**}**

**else**

**{**

**top++;**

**cout<<"Insert Value : ";**

**cin>>val;**

**stack[top]=val;**

**}**

**}**

**void Delete()**

**{**

**if(top==-1)**

**{**

**cout<<"Stack Underflow";**

**}**

**else**

**{**

**cout<<"Poped element : "<<stack[top]<<endl;**

**top--;**

**}**

**}**

**void Display()**

**{**

**cout<<"\n\*\*\*\*Elements in stacks\*\*\*\*\n";**

**for(int i=top; i>=0; i--)**

**{**

**cout<<stack[i]<<" ";**

**cout<<endl;**

**}**

**}**

**};**

**int main()**

**{**

**Stack obj;**

**int opt;**

**do**

**{**

**int option;**

**cout<<"\n----Stack using array----\n";**

**cout<<"Enter 1 to insert \n";**

**cout<<"Enter 2 to delete \n";**

**cout<<"Enter 3 to display \n";**

**cout<<"choice : ";**

**cin>>option;**

**switch(option)**

**{**

**case 1:**

**obj.Insert();**

**break;**

**case 2:**

**obj.Delete();**

**break;**

**case 3:**

**obj.Display();**

**break;**

**default:**

**cout<<"Invalid option";**

**break;**

**}**

**cout<<"Press 1 to continue : ";**

**cin>>opt;**

**}while(opt==1);**

**}**

* **Stack implementation using link list(insertion at front and deletion from front)**

**Code**

**#include<iostream>**

**using namespace std;**

**class Stack**

**{**

**public:**

**int number;**

**Stack \*next;**

**Stack \*start=NULL;**

**void InsertInFront()**

**{**

**Stack \*temp = new Stack();**

**cout<<"Enter number : ";**

**cin>>temp->number;**

**temp->next = start;**

**start = temp;**

**}**

**void DeleteInFront()**

**{**

**if(start==NULL)**

**{**

**cout<<"No node exists";**

**}**

**else**

**{**

**Stack \*temp=start;**

**Stack \*nodeDel = temp;**

**start = temp->next;**

**delete nodeDel;**

**}**

**}**

**void Display()**

**{**

**if(start == NULL)**

**{**

**cout<<"No node exists";**

**}**

**else**

**{**

**Stack \*temp = start;**

**cout<<"\*\*\*\*Stack elements\*\*\*\*\n";**

**while(temp!=NULL)**

**{**

**cout<<temp->number<<endl;**

**temp=temp->next;**

**}**

**}**

**}**

**};**

**int main()**

**{**

**Stack obj;**

**int opt;**

**do**

**{**

**int option;**

**cout<<"\n---Stack using link list\n";**

**cout<<"Enter 1 to insert in start\n";**

**cout<<"Enter 2 to delete from start\n";**

**cout<<"Enter 3 to display stack elements\n";**

**cout<<"Choice : ";**

**cin>>option;**

**switch(option)**

**{**

**case 1:**

**obj.InsertInFront();**

**break;**

**case 2:**

**obj.DeleteInFront();**

**break;**

**case 3:**

**obj.Display();**

**break;**

**default:**

**cout<<"Invalid option";**

**break;**

**}**

**cout<<"Press 1 to continue : ";**

**cin>>opt;**

**}while(opt==1);**

**}**

* **stack implementation using link list (insertion at end, deletion from end)**

**Explanation:**  Here we have inserted a node at the end of a linked list but to display the nodes we have reversed the link list so that it follows the **(LIFO)** principle of the stack.

**Code**

**#include<iostream>**

**using namespace std;**

**class Stack**

**{**

**public:**

**int number;**

**Stack \*next;**

**Stack \*start=NULL;**

**void InsertAtEnd()**

**{**

**if(start == NULL)**

**{**

**Stack \*temp = new Stack();**

**cout<<"Enter number : ";**

**cin>>temp->number;**

**start = temp;**

**}**

**else**

**{**

**Stack \*temp = start;**

**while(temp->next != NULL)**

**{**

**temp = temp->next;**

**}**

**Stack \*t = new Stack();**

**cout<<"Enter number : ";**

**cin>>t->number;**

**temp->next = t;**

**}**

**}**

**void DeleteFromEnd()**

**{**

**if(start==NULL)**

**{**

**cout<<"No node exists\n";**

**}**

**else if(start->next==NULL)**

**{**

**Stack \*nodeDel = start;**

**start = NULL;**

**delete nodeDel;**

**}**

**else**

**{**

**Stack \*temp = start;**

**while(temp->next->next != NULL)**

**{**

**temp = temp->next;**

**}**

**Stack \*nodeDel = temp->next;**

**temp->next = NULL;**

**delete nodeDel;**

**}**

**}**

**int Display(Stack \*st)**

**{**

**if(st==NULL)**

**{**

**return 1;**

**}**

**else**

**{**

**Display(st->next);**

**cout<<st->number<<endl;**

**}**

**}**

**};**

**int main()**

**{**

**Stack obj;**

**int opt;**

**do**

**{**

**int option;**

**cout<<"\n---Stack using link list---\n";**

**cout<<"Enter 1 to insert in end\n";**

**cout<<"Enter 2 to delete from end\n";**

**cout<<"Enter 3 to display stack elements\n";**

**cout<<"Choice : ";**

**cin>>option;**

**switch(option)**

**{**

**case 1:**

**obj.InsertAtEnd();**

**break;**

**case 2:**

**obj.DeleteFromEnd();**

**break;**

**case 3:**

**obj.Display(obj.start);**

**break;**

**default:**

**cout<<"Invalid option";**

**break;**

**}**

**cout<<"Press 1 to continue : ";**

**cin>>opt;**

**}while(opt==1);**

**}**

* **Queue implementation using array**

**Code**

**#include <iostream>**

**using namespace std;**

**class Queue**

**{**

**public:**

**int queue[100];**

**int n;**

**int front;**

**int rear;**

**Queue()**

**{**

**n=100;**

**front=-1;**

**rear=-1;**

**}**

**void Insert()**

**{**

**int val;**

**if (rear == n - 1)**

**cout<<"Queue Overflow"<<endl;**

**else {**

**if (front == - 1)**

**front = 0;**

**cout<<"Insert the element : ";**

**cin>>val;**

**rear++;**

**queue[rear] = val;**

**}**

**}**

**void Delete()**

**{**

**if (front == - 1 || front > rear) {**

**cout<<"Queue Underflow ";**

**return ;**

**} else {**

**cout<<"Element deleted : "<< queue[front] <<endl;**

**front++;;**

**}**

**}**

**void Display() {**

**if (front == - 1)**

**cout<<"Queue is empty"<<endl;**

**else {**

**cout<<"Queue elements are : ";**

**for (int i = front; i <= rear; i++)**

**cout<<queue[i]<<" ";**

**cout<<endl;**

**}**

**}**

**};**

**int main()**

**{**

**Queue obj;**

**int opt;**

**do**

**{**

**int option;**

**cout<<"\n---Queue using array---"<<endl;**

**cout<<"press 1 to Insert element "<<endl;**

**cout<<"press 2 to Delete element "<<endl;**

**cout<<"press 3 to Display all the elements"<<endl;**

**cout<<"Choice : ";**

**cin>>option;**

**switch(option)**

**{**

**case 1:**

**obj.Insert();**

**break;**

**case 2:**

**obj.Delete();**

**break;**

**case 3:**

**obj.Display();**

**break;**

**default:**

**cout<<"Invalid option\n";**

**break;**

**}**

**cout<<"Press 1 to repeat : ";**

**cin>>opt;**

**}while(opt==1);**

**}**

* **Queue implementation using link list (insertion at front, deletion from end)**

**Explanation:**  Here we have inserted a node in the front of a linked list but to display the nodes we have reversed the link list so that it follows the **(FIFO)** principle of the **Queue.**

**Code**

**#include<iostream>**

**using namespace std;**

**class Queue**

**{**

**public:**

**int number;**

**Queue \*next;**

**Queue \*start = NULL;**

**void EnQueueFromFront()**

**{**

**Queue \*temp = new Queue();**

**cout<<"Enter number : ";**

**cin>>temp->number;**

**temp->next=start;**

**start = temp;**

**}**

**void DeQueueFromEnd()**

**{**

**if(start==NULL)**

**{**

**cout<<"No node exists\n";**

**}**

**if(start->next==NULL)**

**{**

**Queue \*nodeDel = start;**

**start=NULL;**

**delete nodeDel;**

**cout<<"Node Deleted\n";**

**}**

**else**

**{**

**Queue \*temp = start;**

**while(temp->next->next != NULL)**

**{**

**temp=temp->next;**

**}**

**Queue \*nodeDel = temp->next;**

**temp->next = NULL;**

**delete nodeDel;**

**cout<<"Node Deleted\n";**

**}**

**}**

**int Display(Queue \*st)**

**{**

**if(st==NULL)**

**{**

**return 1;**

**}**

**else**

**{**

**Display(st->next);**

**cout<<st->number<<" ";**

**}**

**}**

**};**

**int main()**

**{**

**Queue obj;**

**int opt;**

**do**

**{**

**int option;**

**cout<<"\n---Queue using link list\n";**

**cout<<"Enter 1 to Enqueue from front\n";**

**cout<<"Enter 2 to Dequeue from end\n";**

**cout<<"Enter 3 to elements of Queue\n";**

**cout<<"Choice : ";**

**cin>>option;**

**switch(option)**

**{**

**case 1:**

**obj.EnQueueFromFront();**

**break;**

**case 2:**

**obj.DeQueueFromEnd();**

**break;**

**case 3:**

**obj.Display(obj.start);**

**break;**

**default:**

**cout<<"Invalid option";**

**}**

**cout<<"\nPress 1 to continue : ";**

**cin>>opt;**

**}while(opt==1);**

**}**

* **Queue implementation using link list (insertion at end, deletion from start)**

**Code**

**#include<iostream>**

**using namespace std;**

**class Queue**

**{**

**public:**

**int number;**

**Queue \*next;**

**Queue \*start = NULL;**

**void EnQueueFromEnd()**

**{**

**Queue \*temp = new Queue();**

**Queue \*st = start;**

**if(start==NULL)**

**{**

**cout<<"Enter number : ";**

**cin>>temp->number;**

**start = temp;**

**}**

**else**

**{**

**cout<<"Enter number : ";**

**cin>>temp->number;**

**while(st->next != NULL)**

**{**

**st=st->next;**

**}**

**st->next = temp;**

**}**

**}**

**void DeQueueFromFront()**

**{**

**if(start==NULL)**

**{**

**cout<<"No node exists\n";**

**}**

**else**

**{**

**Queue \*nodeDel = start;**

**start = start->next;**

**delete nodeDel;**

**cout<<"Node deleted\n";**

**}**

**}**

**void Display()**

**{**

**if(start==NULL)**

**{**

**cout<<"No node exists\n";**

**}**

**else**

**{**

**Queue \*temp = start;**

**cout<<"Elements in Queue are : ";**

**while(temp != NULL)**

**{**

**cout<<temp->number<<" ";**

**temp = temp->next;**

**}**

**}**

**}**

**};**

**int main()**

**{**

**Queue obj;**

**int opt;**

**do**

**{**

**int option;**

**cout<<"\n---Queue using link list\n";**

**cout<<"Enter 1 to Enqueue from end\n";**

**cout<<"Enter 2 to Dequeue from front\n";**

**cout<<"Enter 3 to elements of Queue\n";**

**cout<<"Choice : ";**

**cin>>option;**

**switch(option)**

**{**

**case 1:**

**obj.EnQueueFromEnd();**

**break;**

**case 2:**

**obj.DeQueueFromFront();**

**break;**

**case 3:**

**obj.Display();**

**break;**

**default:**

**cout<<"Invalid option";**

**}**

**cout<<"\nPress 1 to continue : ";**

**cin>>opt;**

**}while(opt==1);**

**}**