**INDRAPRASTHA COLLEGE FOR WOMEN**

**(University of Delhi)**



**DATA STRUCTURES PRACTICAL**

- Surbhi Singhal

15/085

III SEMESTER

**Qs 1**

**using namespace std;**

**#include<iostream>**

**template<class T>**

**class lowertriangle**

**{**

**public:**

**T \*A;**

**int size;**

**lowertriangle(int x)**

**{**

**size=x;**

**A=new T[size\*(size+1)/2];**

**}**

**lowertriangle<T> &store(T x,int i,int j);**

**T retrieve(int i,int j);**

**};**

**template<class T>**

**lowertriangle<T> &lowertriangle<T>::store(T x,int i,int j)**

**{**

**if(i>=j)**

**A[i\*(i-1)/2+j-1]=x;**

**}**

**template<class T>**

**T lowertriangle<T>::retrieve(int i,int j)**

**{**

**if(i>=j)**

**return A[i\*(i-1)/2+j-1];**

**if(i<0||j<0)**

**cout<<"\n\n\t\t Not Allowed To Enter .";**

**else**

**return 0;**

**}**

**template<class T>**

**class uppertriangle**

**{**

**T \*A;**

**int size;**

**int z;**

**public :**

**uppertriangle(int x)**

**{**

**size=x;**

**A=new T[size\*(size+1)/2];**

**}**

**uppertriangle<T> &store(T x,int i,int j);**

**T retrieve(int i,int j)const;**

**};**

**template<class T>**

**uppertriangle<T> &uppertriangle<T>::store(T x,int i,int j)**

**{**

**if(i<=j)**

**{**

**A[size\*(i-1)-i\*(i-1)/2+j-1]=x;**

**}**

**}**

**template<class T>**

**T uppertriangle<T>::retrieve(int i,int j)const**

**{**

**if(i<=j)**

**return A[size\*(i-1)-i\*(i-1)/2+j-1];**

**if(i<0||j<0)**

**cout<<"\n\n\t\t Not Allowed To Enter .";**

**else**

**if(i!=j)**

**return 0;**

**}**

**template<class T>**

**class diagonal**

**{**

**int n;**

**T \*d;**

**public:**

**diagonal(int size=10)**

**{**

**n=size;**

**d=new T[n];**

**}**

**~diagonal()**

**{**

**delete []d;**

**}**

**diagonal<T>& store(const T& x,int i,int j);**

**T retrieve(int i,int j)const;**

**};**

**template<class T>**

**diagonal<T>& diagonal<T>::store(const T& x,int i,int j)**

**{**

**if(i==j)**

**d[i-1]=x;**

**}**

**template<class T>**

**T diagonal<T>::retrieve(int i,int j)const**

**{**

**if(i==j)**

**return d[i-1];**

**if(i<0||j<0)**

**cout<<"\n\n\t\t Not Allowed To Enter .";**

**else**

**return 0;**

**}**

**template<class T>**

**class symmetric**

**{**

**T \*A;**

**int size;**

**public :**

**symmetric(int x)**

**{**

**size=x;**

**A=new T[size\*(size+1)/2];**

**}**

**symmetric<T> &store(T x,int i,int j);**

**T retrieve(int i,int j);**

**};**

**template<class T>**

**symmetric<T> &symmetric<T>::store(T x,int i,int j)**

**{**

**A[i\*(i-1)/2+j-1]=x;**

**}**

**template<class T>**

**T symmetric<T>::retrieve(int i,int j)**

**{**

**if(i>=j)**

**return A[i\*(i-1)/2+j-1];**

**if(i<0||j<0)**

**cout<<"\n\n\t\t Not Allowed To Enter .";**

**else**

**return A[j\*(j-1)/2+i-1];**

**}**

**int main()**

**{**

**int cho;**

**int i,j,x;**

**cout<<"\n\n\t\tENTER THE MATRIX YOU WANT TO PRINT";**

**cout<<"\n\n\t\t1.LOWERTRIANGULAR MATRIX";**

**cout<<"\n\n\t\t2.UPPERTRIANGULAR MATRIX";**

**cout<<"\n\n\t\t3.DIAGONAL MATRIX";**

**cout<<"\n\n\t\t4.SYMMETRIC MATRIX";**

**cout<<"\n\n\t\tENTER YOUR CHOICE";**

**cout<<"\n\n\t\t";**

**cin>>cho;**

**switch(cho)**

**{ case 1:**

**{**

**cout<<"\n\n\t\t Enter the number of rows in a square matrix :- ";**

**int n,x,i,j;**

**cin>>n;**

**lowertriangle<int> l(n);**

**cout<<"\n\n\t\t Enter the element :- ";**

**cout<<"\n\n\t\t ";**

**for(i=1;i<=n;i++)**

**{**

**for(j=1;j<=n;j++)**

**{**

**cin>>x;**

**cout<<"\n\n\t\t ";**

**l.store(x,i,j);**

**}**

**cout<<"\n\t\t ";**

**}**

**cout<<"\n\n\t\t The elements are :- ";**

**cout<<"\n\t\t\t ";**

**for(i=1;i<=n;i++)**

**{**

**for(j=1;j<=n;j++)**

**{**

**int c=l.retrieve(i,j);**

**cout<<c<<" ";**

**}**

**cout<<"\n\t\t\t ";**

**}**

**cout<<"\n\n\t\t ";**

**break;**

**}**

**case 2:**

**{**

**cout<<"\n\n\t\t Enter the number of rows in a square matrix :- ";**

**int n,x,i,j;**

**cin>>n;**

**uppertriangle<int> u(n);**

**cout<<"\n\n\t\t Enter the element :- ";**

**cout<<"\n\n\t\t ";**

**for(i=1;i<=n;i++)**

**{**

**for(j=1;j<=n;j++)**

**{**

**cin>>x;**

**cout<<"\n\n\t\t ";**

**u.store(x,i,j);**

**}**

**cout<<"\n\t\t ";**

**}**

**cout<<"\n\n\t\t The elements are :- ";**

**cout<<"\n\t\t\t ";**

**for(i=1;i<=n;i++)**

**{**

**for(j=1;j<=n;j++)**

**{**

**int c=u.retrieve(i,j);**

**cout<<c<<" ";**

**}**

**cout<<"\n\t\t\t ";**

**}**

**break;**

**}**

**case 3:**

**{**

**diagonal<int> d(3);**

**int x,c=0;**

**cout<<"\n\n\t\t Enter the element :- ";**

**cout<<"\n\n\t\t ";**

**for(int i=1;i<=3;i++)**

**{**

**for(int j=1;j<=3;j++)**

**{**

**cin>>x;**

**cout<<"\n\n\t\t ";**

**d.store(x,i,j);**

**}**

**}**

**cout<<"\n\n\t\t The matrix is :- \n\t\t\t ";**

**for(int i=1;i<=3;i++)**

**{**

**for(int j=1;j<=3;j++)**

**{**

**c=d.retrieve(i,j);**

**cout<<c<<" ";**

**}**

**cout<<"\n\t\t\t ";**

**}**

**break;**

**}**

**case 4:**

**{**

**cout<<"\n\n\t\t Enter the number of rows in a square matrix :- ";**

**int n,x,i,j;**

**cin>>n;**

**symmetric<int> l(n);**

**cout<<"\n\n\t\t Enter the element :- \n\n\t\t";**

**cout<<"\n\n\t\t ";**

**for(i=1;i<=n;i++)**

**{**

**for(j=1;j<=n;j++)**

**{**

**cin>>x;**

**cout<<"\n\n\t\t ";**

**l.store(x,i,j);**

**}**

**cout<<"\n\t\t ";**

**}**

**cout<<"\n\n\t\t The elements are :- ";**

**cout<<"\n\t\t\t ";**

**for(i=1;i<=n;i++)**

**{**

**for(j=1;j<=n;j++)**

**{**

**int c=l.retrieve(i,j);**

**cout<<c<<" ";**

**}**

**cout<<"\n\t\t\t ";**

**}**

**break;**

**}**

**default:**

**cout<<"INVALID CHOICE";**

**}**

**system("pause");**

**return 0;**

**}**

**// system("pause");**

**//return 0;**

**QS 2**

**using namespace std;**

**#include<iostream>**

**int numpow(int m,int n)**

**{**

**if(n==1)**

**return m;**

**else**

**return(m\*numpow(m,n-1));**

**}**

**int main()**

**{**

**int num;**

**int pow;**

**int mul;**

**cout<<"ENTER THE NUMBER WHOSE POWER IS TO BE FOUND"<<endl;**

**cin>>num;**

**cout<<endl;**

**cout<<"ENTER THE POWER"<<endl;**

**cin>>pow;**

**cout<<"THE NUMBER IS"<<endl;**

**mul=numpow(num,pow);**

**cout<<mul;**

**system("pause");**

**return 0; }**

**QS 3**

**using namespace std;**

**#include<iostream>**

**#include<string>**

**void reverse(string str,int l)**

**{**

**if(l>0)**

**{**

**cout<<str[l-1];**

**reverse(str,l-1);**

**}**

**}**

**int main()**

**{**

**string str;**

**cout<<"ENTER THE STRING "<<endl;**

**getline(cin,str);**

**int l=str.length();**

**cout<<"\nREVERSE OF THE STRING IS "<<endl;**

**reverse(str,l);**

**cout<<endl;**

**system("pause");**

**return 0;**

**}**

**QS 4**

**using namespace std;**

**#include<iostream>**

**class node**

**{**

**friend class stackusinglist;**

**int data;**

**node \*next;**

**public:**

**node()**

**{**

**data=0;**

**next=NULL;**

**}**

**node(int x,node \*n=NULL)**

**{**

**data=x;**

**next=n;**

**}**

**};**

**class stackusinglist**

**{**

**node \*top;**

**public:**

**stackusinglist()**

**{**

**top=NULL;**

**}**

**void push(int element);**

**void pop();**

**void clear();**

**void print();**

**};**

**void stackusinglist::push(int element)**

**{**

**node \*newtop=new node(element);**

**if(top==NULL)**

**{**

**top=newtop;**

**top->next=NULL;**

**}**

**else**

**{**

**newtop->next=top;**

**top=newtop;**

**}**

**}**

**void stackusinglist::pop()**

**{**

**if(top==NULL)**

**cout<<"STACK IS EMPTY";**

**else**

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

**top=NULL;**

**else**

**{**

**node \*temp=top;**

**top=top->next;**

**delete temp;**

**}**

**}**

**void stackusinglist::print()**

**{**

**if(top==NULL)**

**cout<<"STACK EMPTY"<<endl;**

**else**

**{**

**node \*temp=top;**

**while(temp!=NULL)**

**{**

**cout<<temp->data;**

**cout<<endl;**

**temp=temp->next;**

**}**

**}**

**}**

**void stackusinglist::clear()**

**{**

**if(top==NULL)**

**cout<<"STACK IS ALREADY EMPTY"<<endl;**

**else**

**{**

**node \*temp=top;**

**while(top!=NULL)**

**{**

**temp=top->next;**

**delete top;**

**top=temp;**

**}**

**cout<<"STACK CLEARED"<<endl;**

**}**

**}**

**int main()**

**{**

**int len;**

**int num;**

**char ch;**

**cout<<"ENTER THE NUMBER OF ELEMENTS TO BE ADDED IN STACK"<<endl;**

**cin>>len;**

**cout<<endl;**

**if(len!=0)**

**{**

**stackusinglist s;**

**cout<<"ENTER THE ELEMENTS"<<endl;**

**for(int a=0;a<len;a++)**

**{**

**cout<<"Element"<<a+1<<":- ";**

**cin>>num;**

**cout<<endl;**

**s.push(num);**

**}**

**cout<<"THE ELEMENTS ARE "<<endl;**

**s.print();**

**cout<<"DO YOU WANT TO DELETE AN ELEMENT?"<<endl;**

**cout<<"ENTER Y FOR YES AND N FOR NO "<<endl;**

**cin>>ch;**

**if(ch=='Y'||ch=='y')**

**{**

**s.pop();**

**cout<<"THE ELEMENTS AFTER THE DELETION : "<<endl;**

**s.print();**

**}**

**cout<<"DO YOU WANT TO CLEAR THE STACK ?"<<endl;**

**cout<<"ENTER Y FOR YES AND N FOR NO "<<endl;**

**cin>>ch;**

**if(ch=='Y'||ch=='y')**

**{**

**s.clear();**

**}**

**}**

**else**

**cout<<"INVALID INPUT";**

**system("pause");**

**return 0;**

**}**

**QS 5**

**using namespace std;**

**#include<iostream>**

**template<class T>**

**class stack**

**{**

**T \*A;**

**int top,size;**

**public:**

**stack(int s);**

**void push(T x);**

**void display();**

**T pop();**

**void clear();**

**};**

**template<class T>**

**stack<T>::stack(int s)**

**{**

**size=s;**

**A=new T[size];**

**top=-1;**

**}**

**template<class T>**

**void stack<T>::push(T x)**

**{**

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

**{**

**cout<<" Stack Overflowed . ";**

**}**

**else**

**{**

**top=top+1;**

**A[top]=x;**

**}**

**}**

**template<class T>**

**void stack<T>::display()**

**{**

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

**cout<<"STACK EMPTY";**

**else**

**{**

**cout<<"THE ELEMENTS ARE :- "<<endl;**

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

**cout<<A[i]<<" "<<endl;**

**}**

**cout<<endl;**

**}**

**template<class T>**

**T stack<T>::pop()**

**{**

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

**{**

**cout<<"\n\n\t\t Stack Underflowed . ";**

**}**

**else**

**{**

**T x=A[top];**

**top=top-1;**

**return x;**

**}**

**}**

**template<class T>**

**void stack<T>::clear()**

**{**

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

**cout<<"STACK ALREADY CLEARED"<<endl;**

**else**

**{**

**while(top!=-1)**

**pop();**

**cout<<"STACK CLEARED"<<endl;**

**}**

**}**

**int main()**

**{**

**int len;**

**cout<<"ENTER THE NUMBER OF ELEMENTS TO BE ADDED IN STACK"<<endl;**

**cin>>len;**

**if(len!=0)**

**{**

**stack<int> s(len);**

**int x,c=0;**

**char ch;**

**for(int i=0;i<len;i++)**

**{**

**cout<<"ENTER THE "<<i+1<<" ELEMENT :- ";**

**cin>>x;**

**s.push(x);**

**}**

**s.display();**

**cout<<endl;**

**cout<<"DO YOU WANT TO DELETE AN ELEMENT FROM THE STACK "<<endl;**

**cout<<"ENTER Y FOR YES AND N FOR NO "<<endl;**

**cin>>ch;**

**if(ch=='Y' || ch=='y')**

**{**

**c=s.pop();**

**cout<<"ELEMENT DELETED IS "<<c<<endl;**

**cout<<"ELEMENTS AFTER 1ST DELETION"<<endl;**

**s.display();**

**}**

**cout<<"DO YO WANT TO CLEAR THE STACK "<<endl;**

**cout<<"ENTER Y FOR YES AND N FOR NO "<<endl;**

**cin>>ch;**

**if(ch=='Y' || ch=='y')**

**{**

**s.clear();**

**}**

**}**

**else**

**cout<<"INVALID ENTRY";**

**cout<<"\n\n\t\t";**

**system("pause");**

**return 0;**

**}**

**QS 6**

**#include<iostream>**

**using namespace std;**

**template<class t>**

**class arrayqueue**

**{**

**int first,last;**

**int size;**

**t \*A;**

**public:**

**arrayqueue(int s)**

**{ size=s;**

**A=new t[size];**

**first=last=-1;**

**}**

**void enqueue(t el);**

**t dequeue();**

**void print();**

**bool isfull()**

**{**

**if((first==0&&last==size-1)||(first==last+1))**

**return true;**

**else**

**return false;**

**}**

**bool isempty()**

**{**

**if(first==-1)**

**return true;**

**else**

**return false;**

**}**

**};**

**template<class t>**

**void arrayqueue<t>::enqueue(t el)**

**{**

**if(isfull()==true)**

**{**

**cout<<"QUEUE IS FULL";**

**exit(0);**

**}**

**else**

**if(last==size-1||last==-1)**

**{**

**A[0]=el;**

**last=0;**

**if(first==-1)**

**first=0;**

**}**

**else**

**A[++last]=el;**

**}**

**template<class t>**

**t arrayqueue<t>::dequeue()**

**{**

**t temp;**

**if(isempty()==true)**

**{**

**cout<<"QUEUE IS EMPTY";**

**exit(0);**

**}**

**else**

**{**

**temp=A[first];**

**if(first==last)**

**last=first=-1;**

**else**

**if(first==size-1)**

**first=0;**

**else**

**first++;**

**return temp;**

**}**

**}**

**template<class t>**

**void arrayqueue<t>::print()**

**{**

**if(isempty()==true)**

**cout<<"QUEUE IS EMPTY ";**

**else**

**{**

**int p=first;**

**while(p!=last)**

**{**

**cout<<A[p]<<endl;**

**p++;**

**}**

**cout<<A[last];**

**}**

**}**

**int main()**

**{**

**int len;**

**char ch;**

**cout<<"ENTER THE NUMBER OF ELEMENTS TO BE ADDED IN QUEUE"<<endl;**

**cin>>len;**

**if(len!=0)**

**{**

**arrayqueue<int> s(len);**

**int x,c=0;**

**for(int i=0;i<len;i++)**

**{**

**cout<<"ENTER THE "<<i+1<<" ELEMENT :- ";**

**cin>>x;**

**s.enqueue(x);**

**}**

**cout<<endl;**

**cout<<"THE ELEMENTS ARE"<<endl;**

**s.print();**

**cout<<endl;**

**cout<<"DO YOU WANT TO DELETE AN ELEMENT FROM THE QUEUE ?"<<endl;**

**cout<<"ENTER Y FOR YES AND N FOR NO "<<endl;**

**cin>>ch;**

**if(ch=='Y'|| ch=='y')**

**{**

**c=s.dequeue();**

**cout<<"ELEMENT DELETED IS "<<c<<endl;**

**cout<<"ELEMENTS AFTER 1ST DELETION"<<endl;**

**s.print();**

**}**

**}**

**else**

**cout<<"INVALID INPUT";**

**system("pause");**

**return 0;**

**}**

**QS 7**

**using namespace std;**

**#include<iostream>**

**class node**

**{**

**friend class queueusinglist;**

**int i;**

**node \*next;**

**public:**

**node()**

**{**

**i=0;**

**next=NULL;**

**}**

**node(int x,node \*n=NULL)**

**{**

**i=x;**

**next=n;**

**}**

**};**

**class queueusinglist**

**{**

**node \*first;**

**node \*last;**

**public:**

**queueusinglist()**

**{**

**first=NULL;**

**last=NULL;**

**}**

**void enqueue(int element);**

**void dequeue();**

**void print();**

**};**

**void queueusinglist::enqueue(int element)**

**{**

**node \*n=new node(element);**

**if(last==NULL)**

**{**

**first=n;**

**last=n;**

**}**

**else**

**{**

**last->next=n;**

**last=n;**

**}**

**}**

**void queueusinglist::dequeue()**

**{**

**if(first==NULL && last==NULL)**

**cout<<"QUEUE IS EMPTY";**

**else**

**if(first==last)**

**{**

**first=NULL;**

**last=NULL;**

**}**

**else**

**{**

**node \*temp=first;**

**first=first->next;**

**delete temp;**

**}**

**}**

**void queueusinglist::print()**

**{**

**if(first==NULL)**

**cout<<"QUEUE IS EMPTY";**

**else**

**{**

**node \*temp=first;**

**while(temp!=last)**

**{**

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

**temp=temp->next;**

**}**

**cout<<last->i<<endl;**

**}**

**}**

**int main()**

**{**

**int len;**

**int num;**

**char ch;**

**cout<<"ENTER THE NUMBER OF ELEMENTS TO BE ADDED IN QUEUE"<<endl;**

**cin>>len;**

**if(len!=0)**

**{**

**queueusinglist s;**

**cout<<"ENTER THE ELEMENTS"<<endl;**

**for(int a=0;a<len;a++)**

**{**

**cout<<"ELEMENT "<<a+1<<":- ";**

**cin>>num;**

**s.enqueue(num);**

**}**

**cout<<"THE ELEMENTS ARE"<<endl;**

**s.print();**

**cout<<"DO YOU WANT TO DELETE AN ELEMENT FROM THE QUEUE"<<endl;**

**cout<<"ENTER Y FOR YES AND N FOR NO"<<endl;**

**cin>>ch;**

**if(ch=='Y'|| ch=='y')**

**{**

**cout<<"ELEMENTS AFTER 1ST DELETION"<<endl;**

**s.dequeue();**

**s.print();**

**cout<<endl;**

**}**

**}**

**else**

**cout<<"INVALID INPUT";**

**system("pause");**

**return 0;**

**}**

**QS 8**

**using namespace std;**

**#include<iostream>**

**#include<string>**

**class stack**

**{**

**int \*A;**

**int top,size;**

**public:**

**stack(int s);**

**void push(int x);**

**void display();**

**int pop();**

**void add(stack s1,stack s2);**

**};**

**stack::stack(int s)**

**{**

**size=s;**

**A=new int[size];**

**top=-1;**

**}**

**void stack::push(int x)**

**{**

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

**{**

**cout<<"\n\n\t\t Stack Overflowed . ";**

**exit(0);**

**}**

**else**

**{**

**top=top+1;**

**A[top]=x;**

**}**

**}**

**void stack::display()**

**{**

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

**cout<<A[i];**

**cout<<endl;**

**}**

**int stack::pop()**

**{**

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

**{**

**cout<<"\n\n\t\t Stack Underflowed . ";**

**exit(0);**

**}**

**else**

**{**

**int x=A[top];**

**top=top-1;**

**return x;**

**}**

**}**

**void stack::add(stack s1, stack s2)**

**{**

**int result;**

**int top1, top2;**

**int carry=0;**

**while(s1.top!=-1 && s2.top!=-1)**

**{**

**top1=s1.pop();**

**top2=s2.pop();**

**result=top1+top2+carry;**

**if(result>=10)**

**{**

**carry=1;**

**result=result-10;**

**push(result);**

**}**

**else**

**{**

**push(result);**

**carry=0;**

**}**

**}**

**while(s1.top!=-1)**

**{**

**top1=s1.pop();**

**result=top1+carry;**

**if(result>=10)**

**{**

**carry=1;**

**result=result-10;**

**push(result);**

**}**

**else**

**carry=0;**

**push(result);**

**}**

**while(s2.top!=-1)**

**{**

**top2=s2.pop();**

**result=top2+carry;**

**if(result>=10)**

**{**

**carry=1;**

**result=result-10;**

**push(result);**

**}**

**else**

**carry=0;**

**push(result);**

**}**

**if(carry==1)**

**push(carry);**

**}**

**int main()**

**{**

**string a1;**

**string a2;**

**int big;**

**cout<<"ENTER THE FIRST NUMBER TO BE ADDED AS STRING"<<endl;**

**getline(cin,a1);**

**cout<<"\nENTER THE SECOND NUMBER AS STRING"<<endl;**

**getline(cin,a2);**

**int l1=a1.length();**

**int l2=a2.length();**

**if(l1>l2)**

**big=l1;**

**else**

**big=l2;**

**stack s1(big);**

**stack s2(big);**

**stack s3(big+1);**

**int c;**

**char k;**

**for(int i=0;i<l1;i++)**

**{**

**k=a1[i];**

**c=k-'0';**

**s1.push(c);**

**}**

**for(int j=0;j<l2;j++)**

**{**

**k=a2[j];**

**c=k-'0';**

**s2.push(c);**

**}**

**s3.add(s1,s2);**

**cout<<endl<<endl;**

**cout<<"NUMBER OBTAINED AFTER ADDITION"<<endl;**

**s3.display();**

**system("pause");**

**return 0;**

**}**

**QS 9**

**using namespace std;**

**#include<iostream>**

**#include<string>**

**class stack**

**{**

**int \*A;**

**int top,size;**

**public:**

**stack(int s);**

**void push(int x);**

**void display();**

**int pop();**

**void clear();**

**void result(char c);**

**};**

**stack::stack(int s)**

**{**

**size=s;**

**A=new int[size];**

**top=-1;**

**}**

**void stack::push(int x)**

**{**

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

**{**

**cout<<"\n\n\t\t Stack Overflowed . ";**

**exit(0);**

**}**

**else**

**{**

**top=top+1;**

**A[top]=x;**

**}**

**}**

**void stack::display()**

**{**

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

**cout<<"STACK EMPTY";**

**else**

**{**

**cout<<"\n\n\t\t The result is :- ";**

**for(int i=0;i<=top;i++)**

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

**}**

**}**

**int stack::pop()**

**{**

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

**{**

**cout<<"\n\n\t\t Stack Underflowed . ";**

**exit(0);**

**}**

**else**

**{**

**int x=A[top];**

**top=top-1;**

**return x;**

**}**

**}**

**void stack::result(char c)**

**{**

**int a;**

**int b;**

**int d;**

**switch(c)**

**{**

**case '+':a=pop();**

**b=pop();**

**d=a+b;**

**push(d);**

**break;**

**case '-':a=pop();**

**b=pop();**

**d=b-a;**

**push(d);**

**break;**

**case '\*':a=pop();**

**b=pop();**

**d=a\*b;**

**push(d);**

**break;**

**case '/':a=pop();**

**b=pop();**

**d=b/a;**

**push(d);**

**break;**

**case '%':a=pop();**

**b=pop();**

**d=b%a;**

**push(d);**

**break;**

**default:**

**a=c-'0';**

**push(a);**

**}**

**}**

**int main()**

**{**

**cout<<"ENTER YOUR POSTFIX EXPRESSION"<<endl;**

**string str;**

**char c;**

**getline(cin,str);**

**int l=str.length();**

**stack s(l);**

**for(int i=0;i<l;i++)**

**{**

**c=str[i];**

**s.result(c);**

**}**

**s.display();**

**system("pause");**

**return 0;**

**}**

**QS 10**

**using namespace std;**

**#include<iostream>**

**class stack**

**{**

**int \*A;**

**int top,size;**

**public:**

**stack(int s);**

**void push(int x);**

**void display();**

**int pop();**

**void reverse(int l);**

**bool isempty()**

**{**

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

**return true;**

**else**

**return false;**

**}**

**};**

**stack::stack(int s)**

**{**

**size=s;**

**A=new int[size];**

**top=-1;**

**}**

**void stack::push(int x)**

**{**

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

**{**

**cout<<"\n\n\t\t Stack Overflowed . ";**

**}**

**else**

**{**

**top=top+1;**

**A[top]=x;**

**}**

**}**

**void stack::display()**

**{**

**cout<<"THE ELEMENTS ARE :- "<<endl;**

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

**cout<<A[i]<<" "<<endl;**

**}**

**int stack::pop()**

**{**

**int x=0;**

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

**{**

**cout<<"\n\n\t\t Stack Underflowed . ";**

**}**

**else**

**{**

**x=A[top];**

**top=top-1;**

**return x;**

**}**

**}**

**void stack::reverse(int l)**

**{**

**stack s1(l);**

**int x;**

**int a;**

**int m=-1;**

**for(int i=0;i<top;i++)**

**{**

**x=pop();**

**while(top!=m)**

**{**

**a=pop();**

**s1.push(a);**

**}**

**++m;**

**push(x);**

**while(s1.top!=-1)**

**{**

**a=s1.pop();**

**push(a);**

**}**

**}**

**}**

**int main()**

**{**

**int len, a;**

**cout<<"ENTER THE NUMBER OF ELEMENTS TO BE ENTER IN STACK"<<endl;**

**cin>>len;**

**if(len!=0)**

**{**

**stack s(len);**

**stack s1(len);**

**int x,c=0;**

**for(int i=0;i<len;i++)**

**{**

**cout<<"ENTER THE "<<i+1<<" ELEMENT :- ";**

**cin>>x;**

**s.push(x);**

**}**

**s.display();**

**cout<<endl;**

**cout<<"REVERSED ELEMENTS OF STACK ARE"<<endl;**

**s.reverse(len);**

**s.display();**

**}**

**else**

**cout<<"INVALID ENTRY";**

**system("pause");**

**return 0;**

**}**

**QS 11**

**using namespace std;**

**#include<iostream>**

**template<class t>**

**class arrayqueue**

**{**

**int first,last;**

**int size;**

**t \*A;**

**public:**

**arrayqueue(int s)**

**{ size=s;**

**A=new t[size];**

**first=last=-1;**

**}**

**void enqueue(t el);**

**t dequeue();**

**void print();**

**bool isfull()**

**{**

**if((first==0&&last==size-1)||(first==last+1))**

**return true;**

**else**

**return false;**

**}**

**bool isempty()**

**{**

**if(first==-1)**

**return true;**

**else**

**return false;**

**}**

**};**

**template<class t>**

**void arrayqueue<t>::enqueue(t el)**

**{**

**if(isfull()==true)**

**{**

**cout<<"QUEUE IS FULL";**

**exit(0);**

**}**

**else**

**if(last==size-1||last==-1)**

**{**

**A[0]=el;**

**last=0;**

**if(first==-1)**

**first=0;**

**}**

**else**

**A[++last]=el;**

**}**

**template<class t>**

**t arrayqueue<t>::dequeue()**

**{**

**t temp;**

**if(isempty()==true)**

**{**

**cout<<"QUEUE IS EMPTY";**

**exit(0);**

**}**

**else**

**{**

**temp=A[first];**

**if(first==last)**

**last=first=-1;**

**else**

**if(first==size-1)**

**first=0;**

**else**

**first++;**

**return temp;**

**}**

**}**

**template<class t>**

**class stack**

**{**

**t \*A;**

**int top,size;**

**public:**

**stack(int s);**

**void push(t x);**

**void display();**

**t pop();**

**void reverse(int l);**

**template<class T>friend class arrayqueue;**

**};**

**template<class t>**

**stack<t>::stack(int s)**

**{**

**size=s;**

**A=new t[size];**

**top=-1;**

**}**

**template<class t>**

**void stack<t>::push(t x)**

**{**

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

**{**

**cout<<"\n\n\t\t Stack Overflowed . ";**

**exit(0);**

**}**

**else**

**{**

**top=top+1;**

**A[top]=x;**

**}**

**}**

**template<class t>**

**void stack<t>::display()**

**{**

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

**cout<<"STACK EMPTY";**

**else**

**{**

**cout<<"THE ELEMENTS ARE :- "<<endl;**

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

**cout<<A[i]<<" "<<endl;**

**}**

**cout<<endl;**

**}**

**template<class t>**

**t stack<t>::pop()**

**{**

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

**{**

**cout<<"\n\n\t\t Stack Underflowed ";**

**}**

**else**

**{**

**t x=A[top];**

**top=top-1;**

**return x;**

**}**

**}**

**template<class t>**

**void stack<t>::reverse(int l)**

**{**

**arrayqueue <int>k(l);**

**while(top!=-1)**

**{**

**t c=pop();**

**k.enqueue(c);**

**}**

**while(k.isempty()!=true)**

**{**

**t a=k.dequeue();**

**push(a);**

**}**

**}**

**int main()**

**{**

**int num;**

**int n;**

**cout<<"ENTER THE TOTAL NUMBER OF ELEMENTS YOU WANT IN STACK"<<endl;**

**cin>>num;**

**cout<<endl;**

**if(num!=0)**

**{**

**stack <int>s(num);**

**cout<<"ENTER THE ELEMENTS OF STACK"<<endl;**

**for(int i=0;i<num;i++)**

**{**

**cout<<"ENTER THE "<<i+1<<" ELEMENT :";**

**cin>>n;**

**s.push(n);**

**}**

**s.display();**

**cout<<"THE ELEMENTS OF STACK IN REVERSE ORDER "<<endl;**

**s.reverse(num);**

**s.display();**

**}**

**else**

**cout<<"INVALID CHOICE";**

**system("pause");**

**return 0**

**}**

**QS 12**

**using namespace std;**

**#include<iostream>**

**template<class t>**

**class node**

**{**

**t i;**

**node<t> \*next;**

**public: node()**

**{**

**next=NULL;**

**}**

**node(t x, node\*n=NULL)**

**{**

**i=x;**

**next=n;**

**}**

**template<class T>friend class list;**

**};**

**template<class t>**

**class list**

**{**

**node<t> \*start;**

**public:**

**list()**

**{**

**start=NULL;**

**}**

**void insertion(t x);**

**void deletion();**

**void search(t a);**

**void reverse();**

**void display();**

**};**

**template<class t>**

**void list<t>::insertion(t x)**

**{**

**if(start==NULL)**

**start=new node<t>(x);**

**else**

**{**

**node<t> \*temp=start;**

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

**{**

**temp=temp->next;**

**}**

**temp->next=new node<t>(x);**

**}**

**}**

**template<class t>**

**void list<t>::deletion()**

**{**

**if(start==NULL)**

**cout<<"\nEMPTY LIST";**

**else**

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

**start=NULL;**

**else**

**{**

**node<t> \*temp=start;**

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

**{**

**temp=temp->next;**

**}**

**node<t> \*p=temp->next;**

**temp->next=NULL;**

**delete p;**

**}**

**}**

**template<class t>**

**void list<t>::display()**

**{**

**node<t> \*temp=start;**

**while(temp!=NULL)**

**{**

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

**temp=temp->next;**

**}**

**cout<<"NULL"<<endl;**

**}**

**template<class t>**

**void list<t>::reverse()**

**{**

**if(start==NULL)**

**cout<<"LIST IS EMPTY";**

**else**

**{**

**node<t> \*temp;**

**node<t> \*prev=NULL;**

**while(start!=NULL)**

**{**

**temp=start->next;**

**start->next=prev;**

**prev=start;**

**start=temp;**

**}**

**start=prev;**

**}**

**}**

**template<class t>**

**void list<t>::search(t a)**

**{**

**node<t> \*temp=start;**

**bool flag=false;**

**while(temp!=NULL)**

**{**

**if(temp->i==a)**

**{**

**cout<<"NUMBER FOUND IN THE LIST";**

**flag=true;**

**break;**

**}**

**else**

**temp=temp->next;**

**}**

**if(flag==false)**

**cout<<"NUMBER NOT IN THE LIST";**

**}**

**int main()**

**{**

**list<int> l;**

**int op;**

**int len;**

**int a;**

**int sea;**

**char ch;**

**cout<<"ENTER THE LENGTH OF THE LIST"<<endl;**

**cin>>len;**

**if(len!=0)**

**{**

**cout<<"ENTER THE ELEMENTS"<<endl;**

**for(int i=0;i<len;i++)**

**{**

**cin>>a;**

**l.insertion(a);**

**}**

**cout<<"THE LIST IS "<<endl;**

**l.display();**

**cout<<endl;**

**cout<<"DO YOU WANT TO DELETE AN ELEMENT FROM THE LIST?"<<endl;**

**cout<<"ENTER Y FOR YES AND N FOR NO "<<endl;**

**cin>>ch;**

**if(ch=='Y' || ch=='y')**

**{**

**cout<<"LIST AFTER DELETION"<<endl;**

**l.deletion();**

**l.display();**

**}**

**cout<<"DO YOU WANT TO REVERSE THE ELEMENTS OF THE LIST?"<<endl;**

**cout<<"ENTER Y FOR YES AND N FOR NO "<<endl;**

**cin>>ch;**

**if(ch=='Y' || ch=='y')**

**{**

**cout<<"LIST AFTER REVERSAL"<<endl;**

**l.reverse();**

**l.display();**

**}**

**cout<<"DO YOU WANT TO SEARCH AN ELEMENT IN THE LIST?"<<endl;**

**cout<<"ENTER Y FOR YES AND N FOR NO "<<endl;**

**cin>>ch;**

**if(ch=='Y' || ch=='y')**

**{**

**cout<<"ENTER THE NUMBER TO BE SEARCHED IN THE LIST"<<endl;**

**cin>>sea;**

**l.search(sea);**

**}**

**}**

**else**

**cout<<"INVALID ENTERY";**

**system("pause");**

**return 0;**

**}**

**QS 13**

**using namespace std;**

**#include<iostream>**

**template<class t>**

**class node**

**{**

**t i;**

**node<t> \*next;**

**node<t> \*prev;**

**public:**

**node()**

**{**

**i=0;**

**next=NULL;**

**prev=NULL;**

**}**

**node(t x,node \*n=NULL,node \*p=NULL)**

**{**

**i=x;**

**next=n;**

**prev=p;**

**}**

**template<class T>friend class list;**

**};**

**template<class t>**

**class list**

**{**

**node<t> \*first;**

**node<t> \*last;**

**public:**

**list()**

**{**

**first=NULL;**

**last=NULL;**

**}**

**void insertion(t i);**

**void deletion();**

**void search(t a);**

**void reverse();**

**void display();**

**};**

**template<class t>**

**void list<t>::insertion(t i)**

**{**

**node<t> \*n=new node<t>(i);**

**if(first==NULL)**

**first=last=n;**

**else**

**{**

**last->next=n;**

**n->prev=last;**

**last=n;**

**}**

**}**

**template<class t>**

**void list<t>::deletion()**

**{**

**if(first==NULL)**

**cout<<"EMPTY LIST";**

**else**

**if(first==last)**

**{**

**first=NULL;**

**last=NULL;**

**}**

**else**

**{**

**node<t> \*temp=last;**

**last=last->prev;**

**last->next=NULL;**

**delete temp;**

**}**

**}**

**template<class t>**

**void list<t>::search(t a)**

**{**

**if(first==NULL)**

**cout<<"LIST IS EMPTY";**

**else**

**if(last->i==a)**

**cout<<"NUMBER FOUND AT LAST POSITION"<<endl;**

**else**

**{**

**node<t> \*temp=first;**

**int count=0;**

**while(temp!=last && temp->i!=a)**

**{**

**temp=temp->next;**

**++count;**

**}**

**if(temp!=last)**

**cout<<"NUMBER FOUND AT NODE POSITION "<<count+1;**

**else**

**cout<<"NUMBER NOT FOUND";**

**}**

**}**

**template<class t>**

**void list<t>::reverse()**

**{**

**int n=0;**

**node<t> \*fp=first;**

**node<t> \*lp=last;**

**t tem;**

**node<t> \*temp=first;**

**while(temp!=last)**

**{**

**temp=temp->next;**

**++n;**

**}**

**for(int a=0;a<=n/2;a++)**

**{**

**tem=fp->i;**

**fp->i=lp->i;**

**lp->i=tem;**

**fp=fp->next;**

**lp=lp->prev;**

**}**

**}**

**template<class t>**

**void list<t>::display()**

**{**

**if(first==NULL)**

**cout<<"EMPTY LIST NOTHING TO DISPLAY"<<endl;**

**else**

**{**

**node<t> \*temp=first;**

**while(temp!=last)**

**{**

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

**temp=temp->next;**

**}**

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

**cout<<"NULL"<<endl;**

**}**

**}**

**int main()**

**{**

**list<int> l;**

**int len;**

**int num;**

**int n;**

**char ch;**

**cout<<"ENTER THE LENGTH OF THE LIST"<<endl;**

**cin>>len;**

**if(len==0)**

**cout<<"INVALID ENTRY";**

**else**

**{**

**cout<<"ENTER THE ELEMENTS"<<endl;**

**for(int i=0;i<len;i++)**

**{**

**cin>>n;**

**l.insertion(n);**

**}**

**cout<<"THE LIST IS "<<endl;**

**l.display();**

**cout<<endl;**

**cout<<"DO YOU WANT TO DELETE AN ELEMENT FROM THE LIST?"<<endl;**

**cout<<"ENTER Y FOR YES AND N FOR NO "<<endl;**

**cin>>ch;**

**if(ch=='Y' || ch=='y')**

**{**

**cout<<"LIST AFTER DELETION"<<endl;**

**l.deletion();**

**l.display();**

**}**

**cout<<"DO YOU WANT TO REVERSE THE ELEMENTS OF THE LIST?"<<endl;**

**cout<<"ENTER Y FOR YES AND N FOR NO "<<endl;**

**cin>>ch;**

**if(ch=='Y' || ch=='y')**

**{**

**cout<<"LIST AFTER REVERSAL"<<endl;**

**l.reverse();**

**l.display();**

**}**

**cout<<"DO YOU WANT TO SEARCH AN ELEMENT IN THE LIST?"<<endl;**

**cout<<"ENTER Y FOR YES AND N FOR NO "<<endl;**

**cin>>ch;**

**if(ch=='Y' || ch=='y')**

**{**

**cout<<"ENTER THE NUMBER TO BE SEARCHED IN THE LIST"<<endl;**

**cin>>n;**

**l.search(n);**

**}**

**}**

**system("pause");**

**return 0;**

**}**

**QS 14**

**using namespace std;**

**#include<iostream>**

**template<class t>**

**class node**

**{**

**t i;**

**node<t> \*next;**

**public:**

**node()**

**{**

**i=0;**

**next=NULL;**

**}**

**node(t x,node<t> \*n=NULL)**

**{**

**i=x;**

**next=n;**

**}**

**template<class T>friend class list;**

**};**

**template<class t>**

**class list**

**{**

**node<t> \*last;**

**public:**

**list()**

**{**

**last=NULL;**

**}**

**void insertion(t i);**

**void deletion();**

**void search(t a);**

**void reverse();**

**void display();**

**};**

**template<class t>**

**void list<t>::insertion(t i)**

**{**

**node<t> \*n=new node<t>(i);**

**if(last==NULL)**

**{**

**last=n;**

**last->next=n;**

**}**

**else**

**{**

**n->next=last->next;**

**last->next=n;**

**last=n;**

**}**

**}**

**template<class t>**

**void list<t>::deletion()**

**{**

**if(last==NULL)**

**cout<<"EMPTY LIST";**

**else**

**if(last->next==last)**

**last=NULL;**

**else**

**{**

**node<t> \*temp=last->next;**

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

**temp=temp->next;**

**temp->next=last->next;**

**last=temp;**

**}**

**}**

**template<class t>**

**void list<t>::search(t a)**

**{**

**if(last==NULL)**

**cout<<"EMPTY LIST";**

**else**

**if(last->i==a)**

**cout<<"NUMBER FOUND AT LAST LOCATION";**

**else**

**{**

**int count=0;**

**node<t> \*temp=last->next;**

**while(temp!=last && temp->i!=a)**

**{**

**temp=temp->next;**

**++count;**

**}**

**if(temp!=last)**

**cout<<"ELEMENT FOUND AT POSITION "<<count+1;**

**else**

**cout<<"ELEMENT NOT IN THE LIST";**

**}**

**}**

**template<class t>**

**void list<t>::reverse()**

**{**

**node<t> \*l=last;**

**node<t> \*n=NULL;**

**node<t> \*temp=last->next;**

**while(last->next!=last)**

**{**

**n=last->next->next;**

**last->next->next=l;**

**l=last->next;**

**last->next=n;**

**}**

**last->next->next=l;**

**last=temp;**

**}**

**template<class t>**

**void list<t>::display()**

**{**

**if(last==NULL)**

**{**

**cout<<"EMPTY LIST"<<endl;**

**}**

**else**

**{**

**node<t> \*temp=last->next;**

**while(temp!=last)**

**{**

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

**temp=temp->next;**

**}**

**cout<<last->i<<endl;**

**}**

**}**

**int main()**

**{**

**list <int>l;**

**int len;**

**int num;**

**char ch;**

**cout<<"ENTER THE LENGTH OF THE LIST"<<endl;**

**cin>>len;**

**if(len==0)**

**cout<<"INVALID ENTRY"<<endl;**

**else**

**{**

**int n;**

**cout<<"ENTER THE ELEMENTS"<<endl;**

**for(int a=0;a<len;a++)**

**{**

**cin>>num;**

**l.insertion(num);**

**}**

**cout<<endl;**

**cout<<"THE LIST IS"<<endl;**

**l.display();**

**cout<<"DO YOU WANT TO DELETE AN ELEMENT FROM THE LIST?"<<endl;**

**cout<<"ENTER Y FOR YES AND N FOR NO "<<endl;**

**cin>>ch;**

**if(ch=='Y' || ch=='y')**

**{**

**cout<<"LIST AFTER FIRST DELETION"<<endl;**

**l.deletion();**

**l.display();**

**}**

**cout<<"DO YOU WANT TO REVERSE THE ELEMENTS OF THE LIST?"<<endl;**

**cout<<"ENTER Y FOR YES AND N FOR NO "<<endl;**

**cin>>ch;**

**if(ch=='Y' || ch=='y')**

**{**

**cout<<"LIST AFTER REVERSAL"<<endl;**

**l.reverse();**

**l.display();**

**}**

**cout<<"DO YOU WANT TO SEARCH AN ELEMENT IN THE LIST?"<<endl;**

**cout<<"ENTER Y FOR YES AND N FOR NO "<<endl;**

**cin>>ch;**

**if(ch=='Y' || ch=='y')**

**{**

**cout<<"ENTER THE NUMBER TO BE SEARCHED IN THE LIST"<<endl;**

**cin>>n;**

**l.search(n);**

**}**

**}**

**system("pause");**

**return 0;**

**}**

**QS 15**

**using namespace std;**

**#include<iostream>**

**class node**

**{**

**int deg;**

**int val;**

**node \*next;**

**public: node()**

**{ next=NULL; }**

**node(int x,int y, node\*n=NULL)**

**{ deg=x;**

**val=y;**

**next=n;**

**}**

**friend class list;**

**};**

**class list**

**{**

**node \*start;**

**public:**

**list()**

**{**

**start=NULL;**

**}**

**void insert\_at\_end(int x,int y);**

**void delete\_at\_beg();**

**void delete\_at\_end();**

**void display();**

**void add(list \*l1,list \*l2);**

**};**

**void list::insert\_at\_end(int x,int y)**

**{**

**node \*n=new node(x,y);**

**if(start==NULL)**

**start=n;**

**else**

**{**

**node \*temp=start;**

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

**{**

**temp=temp->next;**

**}**

**temp->next=n;**

**}**

**}**

**void list::add(list \*l1,list \*l2)**

**{**

**node \*t1,\*t2;**

**int res;**

**t1=l1->start;**

**t2=l2->start;**

**while(t1!=NULL || t2!=NULL)**

**{**

**if(t1->deg==t2->deg)**

**{**

**res=t1->val+t2->val;**

**insert\_at\_end(t1->deg,res);**

**t1=t1->next;**

**t2=t2->next;**

**}**

**else**

**if(t1->deg>t2->deg)**

**{**

**res=t1->val;**

**insert\_at\_end(t1->deg,res);**

**t1=t1->next;**

**}**

**else**

**{**

**res=t2->val;**

**insert\_at\_end(t2->deg,res);**

**t2=t2->next;**

**}**

**}**

**}**

**void list::display()**

**{**

**node \*temp=start->next;**

**cout<<start->val<<"x^"<<start->deg;**

**while(temp!=NULL)**

**{**

**if(temp->val!=0)**

**{**

**if(temp->val>0)**

**cout<<"+"<<temp->val<<"x^"<<temp->deg;**

**else**

**cout<<temp->val<<"x^"<<temp->deg;**

**}**

**temp=temp->next;**

**}**

**cout<<endl;**

**}**

**int main()**

**{**

**list l1;**

**list l2;**

**list l3;**

**int higdeg1, higdeg2;**

**int c;**

**cout<<"ENTER THE HIGHEST DEGREE IN THE FIRST POLYNOMIAL"<<endl;**

**cin>>higdeg1;**

**for(int i=higdeg1;i>=0;i--)**

**{**

**cout<<"ENTER THE COEFFFICIENT OF DEGREE "<<i<<endl;**

**cin>>c;**

**l1.insert\_at\_end(i,c);**

**}**

**cout<<"FIRST POLYNOMIAL IS"<<endl;**

**l1.display();**

**cout<<"ENTER THE HIGHEST DEGREE IN THE SECOND POLYNOMIAL"<<endl;**

**cin>>higdeg2;**

**for(int i=higdeg2;i>=0;i--)**

**{**

**cout<<"ENTER THE COEFFFICIENT OF DEGREE "<<i<<endl;**

**cin>>c;**

**l2.insert\_at\_end(i,c);**

**}**

**cout<<endl;**

**cout<<"SECOND POLYNOMIAL IS"<<endl;**

**l2.display();**

**l3.add(&l1,&l2);**

**cout<<endl;**

**cout<<"POLYNOMIAL AFTER THE ADDITION OF ABOVE TWO POLYNOMIALS"<<endl;**

**l3.display();**

**system("pause");**

**return 0;**

**};**

**QS 16**

**using namespace std;**

**#include<iostream>**

**class node**

**{**

**friend class list;**

**int i;**

**node \*next;**

**public:**

**node()**

**{**

**next=NULL;**

**}**

**node(int x,node \*n=NULL)**

**{**

**i=x;**

**next=n;**

**}**

**};**

**class list**

**{**

**node \*start;**

**public:**

**list()**

**{**

**start=NULL;**

**}**

**void insert\_at\_beg(int x);**

**void insert\_at\_end(int x);**

**void delete\_at\_beg();**

**void delete\_at\_end();**

**void merge(list \*l1,list \*l2);**

**void display();**

**};**

**void list::insert\_at\_beg(int x)**

**{**

**node \*n=new node(x);**

**if(start==NULL)**

**start=n;**

**else**

**{**

**n->next=start;**

**start=n;**

**}**

**}**

**void list::insert\_at\_end(int x)**

**{**

**node \*n=new node(x);**

**if(start==NULL)**

**start=n;**

**else**

**{**

**node \*temp=start;**

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

**{**

**temp=temp->next;**

**}**

**temp->next=n;**

**}**

**}**

**void list::delete\_at\_beg()**

**{**

**if(start==NULL)**

**cout**

**<<"EMPTY LIST";**

**else**

**{**

**node \*temp=start;**

**start=start->next;**

**delete temp;**

**}**

**}**

**void list::delete\_at\_end()**

**{**

**if(start==NULL)**

**cout<<"\nEMPTY LIST";**

**else**

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

**start=NULL;**

**else**

**{**

**node \*temp=start;**

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

**{**

**temp=temp->next;**

**}**

**node \*p=temp->next;**

**temp->next=NULL;**

**delete p;**

**}**

**}**

**void list::merge(list \*l1,list \*l2)**

**{**

**node \*t1,\*t2;**

**t1=l1->start;**

**t2=l2->start;**

**while(t1!=NULL && t2!=NULL)**

**{**

**if(t1->i == t2->i)**

**{insert\_at\_end(t1->i);**

**t1=t1->next;**

**t2=t2->next;**

**}**

**else**

**if(t1->i < t2->i)**

**{**

**insert\_at\_end(t1->i);**

**t1=t1->next;**

**}**

**else**

**{**

**insert\_at\_end(t2->i);**

**t2=t2->next;**

**}**

**}**

**while(t1!=NULL)**

**{**

**insert\_at\_end(t1->i);**

**t1=t1->next;**

**}**

**while(t2!=NULL)**

**{**

**insert\_at\_end(t2->i);**

**t2=t2->next;**

**}**

**}**

**void list::display()**

**{**

**node \*temp=start;**

**while(temp!=NULL)**

**{**

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

**temp=temp->next;**

**}**

**cout<<"NULL"<<endl;**

**}**

**int main()**

**{**

**list l;**

**int len;**

**int ch;**

**int op;**

**list l1,l2,l3;**

**int a,len2,z,len1;**

**cout<<"WHICH OPERARTION DO YOU WISH TO PERFORM"<<endl;**

**cout<<"1. INSERTION"<<endl;**

**cout<<"2. DELETION"<<endl;**

**cout<<"3. MERGING"<<endl;**

**cout<<"ENTER YOUR CHOICE"<<endl;**

**cin>>ch;**

**switch(ch)**

**{**

**case 1: cout<<"CHOOSE AN OPTION";**

**cout<<"\n1.INSERT AT BEG";**

**cout<<"\n2.INSERT AT END"<<endl;**

**cin>>op;**

**switch(op)**

**{**

**case 1:cout<<"ENTER THE LENGTH OF THE LIST"<<endl;**

**cin>>len;**

**if(len==0)**

**cout<<"INVALID CHOICE";**

**else**

**{**

**cout<<"ENTER THE ELEMENTS"<<endl;**

**for(int i=0;i<len;i++)**

**{**

**cin>>a;**

**l.insert\_at\_beg(a);**

**}**

**cout<<"THE LIST IS "<<endl;**

**l.display();**

**}**

**break;**

**case 2:cout<<"ENTER THE LENGTH OF THE LIST"<<endl;**

**cin>>len;**

**if(len==0)**

**cout<<"INVALID CHOICE";**

**else**

**{**

**cout<<"ENTER THE ELEMENTS"<<endl;**

**for(int i=0;i<len;i++)**

**{**

**cin>>a;**

**l.insert\_at\_end(a);**

**}**

**cout<<"THE LIST IS "<<endl;**

**l.display();**

**}**

**break;**

**default: cout<<"WRONG OPTION";**

**}**

**break;**

**case 2:cout<<"CHOOSE AN OPTION";**

**cout<<"\n 1.DELETE AT BEG";**

**cout<<"\n 2.DELETE AT END"<<endl;**

**cin>>op;**

**switch(op)**

**{**

**case 1:l.delete\_at\_beg();**

**cout<<"ELEMENTS AFTER FIRST DELETION"<<endl;**

**l.display();**

**break;**

**case 2:l.delete\_at\_end();**

**cout<<"ELEMENTS AFTER FIRST DELETION"<<endl;**

**l.display();**

**break;**

**default:**

**cout<<"WRONG OPTION";**

**}**

**break;**

**case 3:cout<<"ENTER LENGTH OF FIRST LIST TO BE MERGED"<<endl;**

**cin>>len1;**

**for(int i=0;i<len1;i++)**

**{**

**cout<<"enter element"<<i+1<<" :";**

**cin>>z;**

**l1.insert\_at\_end(z);**

**}**

**l1.display();**

**cout<<"\nENTER LENGTH OF SECOND LIST"<<endl;**

**cin>>len2;**

**for(int j=0;j<len2;j++)**

**{**

**cout<<"\nenter element"<<j+1<<" :";**

**cin>>z;**

**l2.insert\_at\_end(z);**

**}**

**l2.display();**

**l3.merge(&l1,&l2);**

**cout<<"\nAfter merging\n";**

**l3.display();**

**break;**

**default:**

**cout<<"INVALID ENTRY";**

**}**

**system("pause");**

**return 0;**

**}**

**QS 17**

**#include<iostream>**

**using namespace std;**

**class bstnode**

**{**

**friend class bstree;**

**bstnode \*left,\*right;**

**int key;**

**public:**

**bstnode()**

**{**

**key=0;**

**left=NULL;**

**right=NULL;**

**}**

**bstnode(int x,bstnode \*l=NULL,bstnode \*r=NULL)**

**{**

**key=x;**

**left=l;**

**right=r;**

**}**

**};**

**class bstree**

**{**

**bstnode \*root;**

**public:**

**bstree()**

**{**

**root=NULL;**

**}**

**void preorder(bstnode \*p);**

**void inorder(bstnode \*p);**

**void postorder(bstnode \*p);**

**void insert(int el);**

**void order();**

**void search(int el);**

**void deletebycopying(bstnode\* p);**

**void deletebymerging(bstnode\* p);**

**int getleafcount(bstnode \*p);**

**void getleafcount();**

**int height(bstnode \*p);**

**void height();**

**int nonleafcount(bstnode \*p);**

**void delete\_c(int el);**

**void delete\_m(int el);**

**void mirrorimage();**

**bstnode\* mirrorimage(bstnode \*temp);**

**void levelbylevel();**

**void levelbylevel(bstnode \*p,int level);**

**};**

**void bstree::levelbylevel()**

**{**

**int h= height(root);**

**int i;**

**cout<<"\n\n\t\t Level by level traversal :- ";**

**for(int i=1;i<=h;i++)**

**levelbylevel(root,i);**

**}**

**void bstree::levelbylevel(bstnode \*p,int level)**

**{**

**if(p==NULL)**

**return ;**

**else if(level==1)**

**cout<<p->key<<" ";**

**else if(level>1)**

**{**

**levelbylevel(p->left,level-1);**

**levelbylevel(p->right,level-1);**

**}**

**}**

**void bstree:: mirrorimage()**

**{**

**cout<<"\n\n\t\t Original traversal :- ";**

**inorder(root);**

**bstnode \* m=mirrorimage(root);**

**cout<<"\n\n\t\t Mirror image:- ";**

**inorder(m);**

**}**

**bstnode\* bstree:: mirrorimage(bstnode \*temp)**

**{**

**bstnode \*temp1;**

**if(temp==NULL)**

**return NULL;**

**else**

**{**

**temp1=temp->left;**

**temp->left=mirrorimage(temp->right);**

**temp->right=mirrorimage(temp1);**

**return temp;**

**}**

**}**

**int bstree::nonleafcount(bstnode \*p)**

**{**

**if(p==NULL || p->left==NULL && p->right==NULL)**

**return 0;**

**else return (1+ nonleafcount(p->left)+nonleafcount(p->right));**

**}**

**void bstree::delete\_c(int el)**

**{**

**bstnode \*p=root;**

**while(p!=NULL)**

**{**

**if(el==p->key)**

**{**

**cout<<"\n\n\t\t The element get deleted is :- "<<p->key;**

**deletebycopying(p);**

**break;**

**}**

**else if(el<p->key)**

**p=p->left;**

**else**

**p=p->right;**

**}**

**}**

**void bstree::deletebycopying(bstnode \* p)**

**{**

**bstnode \*prev;**

**bstnode \*tmp=p;**

**if(p->right==NULL)**

**p=p->left;**

**else if(p->left==NULL)**

**p=p->right;**

**else**

**{**

**tmp=p->left;**

**prev=p;**

**while(tmp->right!=NULL)**

**{**

**prev=tmp;**

**tmp=tmp->right;**

**}**

**p->key=tmp->key;**

**if(prev==p)**

**prev->left=tmp->left;**

**else**

**prev->right=tmp->left;**

**}**

**delete tmp;**

**cout<<"\n\n\t\t After deletion :- ";**

**inorder(root);**

**cout<<"NODE DELETED";**

**}**

**void bstree ::deletebymerging(bstnode \*p)**

**{**

**bstnode\*tmp=p;**

**if(p!=NULL)**

**{**

**if(p->right==NULL)**

**p=p->left;**

**else if(p->left==NULL)**

**p=p->right;**

**else**

**{**

**tmp=p->left;**

**while(tmp->right!=NULL)**

**tmp=tmp->right;**

**tmp->right=p->right;**

**tmp=p;**

**p=p->left;**

**}**

**delete tmp;}**

**}**

**void bstree::delete\_m(int el)**

**{**

**bstnode \*p=root;**

**while(p!=NULL)**

**{**

**if(el==p->key)**

**{**

**cout<<"\n\n\t\t The element get deleted is :- "<<p->key;**

**deletebymerging(p);**

**break;**

**}**

**else if(el<p->key)**

**p=p->left;**

**else**

**p=p->right;**

**}**

**}**

**void bstree ::getleafcount()**

**{**

**int c=getleafcount(root);**

**cout<<"\n\n\t\t Leaf Nodes :- "<<c;**

**int c1=nonleafcount(root);**

**cout<<"\n\n\t\t Non-leaf nodes :- "<<c1;**

**}**

**int bstree::getleafcount(bstnode \*p)**

**{**

**if(p==NULL)**

**return 0;**

**else**

**if(p->left==NULL&&p->right==NULL)**

**return 1;**

**else**

**return getleafcount(p->left)+getleafcount(p->right);**

**}**

**int bstree::height(bstnode \*p)**

**{**

**int h1=0,h2=0;**

**if(p==NULL)**

**return 0;**

**if(p->left==NULL &&p->right==NULL)**

**return 1;**

**else**

**h1= height(p->left);**

**h2= height(p->right);**

**if(h1>h2)**

**{**

**h1=h1+1;**

**return h1;**

**}**

**else**

**{**

**h2=h2+1;**

**return h2;**

**}**

**}**

**void bstree::height()**

**{**

**cout<<"\n\n\t\t Height :- ";**

**int h=height(root);**

**cout<<h;**

**}**

**void bstree::search (int el)**

**{**

**bstnode \*p=root;**

**while(p!=NULL)**

**{**

**if(el==p->key)**

**{**

**cout<<"\n\n\t\t FOUND !! ";**

**break;**

**}**

**else if(el<p->key)**

**p=p->left;**

**else**

**p=p->right;**

**}**

**}**

**void bstree::preorder(bstnode \*p=NULL )**

**{**

**if(p!=NULL)**

**{**

**cout<<p->key<<" ";**

**preorder(p->left);**

**preorder(p->right);**

**}**

**}**

**void bstree::inorder(bstnode \*p=NULL )**

**{**

**if(p!=NULL)**

**{**

**inorder(p->left);**

**cout<<p->key<<" ";**

**inorder(p->right);**

**}**

**}**

**void bstree::postorder(bstnode \*p=NULL )**

**{**

**if(p!=NULL)**

**{**

**postorder(p->left);**

**postorder(p->right);**

**cout<<p->key<<" ";**

**}**

**}**

**void bstree::insert(int el)**

**{**

**bstnode\*p=root;**

**bstnode\*prev=NULL;**

**while(p!=NULL)**

**{**

**prev=p;**

**if(el<p->key)**

**p=p->left;**

**else**

**p=p->right;**

**}**

**if(root==NULL)**

**root=new bstnode(el);**

**else if(el<prev->key)**

**prev->left=new bstnode(el);**

**else**

**prev->right=new bstnode(el);**

**}**

**void bstree::order()**

**{**

**cout<<"\n\n\t\t Preorder traversal :- ";**

**preorder(root);**

**cout<<"\n\n\t\t Inorder traversal :- ";**

**inorder(root);**

**cout<<"\n\n\t\t Postorder traversal :- ";**

**postorder(root);**

**}**

**int main()**

**{**

**bstree t;**

**cout<<"\n\n\t\t You can implement following operations :- ";**

**cout<<"\n\n\t\t 1.Insertion";**

**cout<<"\n\n\t\t 2.Deletion by copying";**

**cout<<"\n\n\t\t 3.Deletion by merging";**

**cout<<"\n\n\t\t 4.Search a no. in BST";**

**cout<<"\n\n\t\t 5.Display its preorder,postorder and inorder traversal";**

**cout<<"\n\n\t\t 6.Display its level by level traversal";**

**cout<<"\n\n\t\t 7.Count the leaf and non-leaf nodes";**

**cout<<"\n\n\t\t 8.Display height of the tree";**

**cout<<"\n\n\t\t 9.Create a mirror image of the tree";**

**char ch;**

**int n;**

**do**

**{**

**cout<<"\n\n\t\t Enter your choice :- ";**

**cin>>n;**

**switch(n)**

**{**

**case 1:**

**int size,x;**

**cout<<"\n\n\t\t Enter no. of nodes you want to insert :- ";**

**cin>>size;**

**cout<<"\n\n\t\t Enter the elements in new node :- ";**

**for(int i=0;i<size;i++)**

**{**

**cin>>x;**

**t.insert(x);**

**}**

**break;**

**case 2:**

**int d;**

**cout<<"\n\n\t\t Enter the element which you want to delete :- ";**

**cin>>d;**

**t.delete\_c(d);**

**break;**

**case 3:**

**int d1;**

**cout<<"\n\n\t\t Enter the element which you want to delete :- ";**

**cin>>d1;**

**t.delete\_m(d1);**

**break;**

**case 4:**

**int s;**

**cout<<"\n\n\t\t Enter the element you want to search in the tree :- ";**

**cin>>s;**

**t.search(s);**

**break;**

**case 5:**

**t.order();**

**break;**

**case 6:**

**t.levelbylevel();**

**break;**

**case 7:**

**t.getleafcount();**

**break;**

**case 8:**

**t.height();**

**break;**

**case 9:**

**t.mirrorimage();**

**break;**

**default:**

**cout<<"\n\n\t\t You entered a wrong choice !!";**

**}**

**cout<<"\n\n\t\t Do you want to continue(Y/N) ? ";**

**cin>>ch;**

**}**

**while(ch=='Y'||ch=='y');**

**cout<<"\n\n\t\t ";**

**system("pause");**

**return 0;**

**}**