**DATA STRUCTURE PRACTICALS**

**SUBJECT CODE-32341301**

**Made By**

**Simran Pandey**

**ROLL NO-15/1296**

**B.SC (Hons) COMPUTER SCIENCE**

**THIRD SEMESTER**

**Index**

**1. Write a menu driven program to implement the following sparse matrices :**

**a) Diagonal Matrix**

**b) Lower Triangular Matrix**

**c) Upper Triangular Matrix**

**d) Symmetric Matrix**

**2. WAP to compute b r using recursion where b represents base and r represents power.**

**3. WAP to reverse a user entered string using recursion.**

**4. Perform the following Stack operations using Linked List implementation.**

**a) Push**

**b) Pop**

**c) Clear**

**5. Perform the following Stack operations using Array implementation using Templates:**

**a) Push**

**b) Pop**

**c) Clear**

**6. Perform the following Queue operations using Circular Array implementation:**

**a) Enqueue**

**b) Dequeue**

**7. Create and perform the following operations on Queues using Linked List implementation:**

**a) Enqueue**

**b) Dequeue**

**8. WAP to add two large integers using stack.**

**9. WAP to evaluate postfix expression using stack.**

**10. Implementation Linked List using templates. Include functions for insertion, deletion and search of a number, reverse the list .**

**11. Implement Doubly Linked List using templates. Include functions for insertion, deletion and search of a number, reverse the list.**

**12. Implement Circular Linked List using templates. Include functions for insertion, deletion and search of a number, reverse the list.**

**13. WAP to add two polynomials using linked list representation.**

**14. Write a menu driven program to implement the following operations in an ordered linked list:**

**a) Insertion**

**b) Deletion**

**c) Merging**

**15. Write a menu driven program to implement the following operations in a Binary Search Tree:**

**a) Insertion**

**b) Deletion by copying**

**c) Deletion by Merging**

**d) Search a no. Merging**

**e) Search a no. in BST**

**f) Display its preorder, postorder and inorder traversals**

**g) Display its level-by- level traversals**

**h) Count the non-leaf nodes and leaf nodes**

**i) Display height of tree**

**16. Write a menu driven program to implement the following sorting and searching algorithms:**

**a) Bubble Sort**

**b) Insertion**

**c) Selection**

**d) Merge Sort**

**e) Quick Sort**

**f) Linear Search**

**g) Binary Search**

**Q1 SPARSE MATRIX**

**#include<iostream>**

**#include<math.h>**

**using namespace std;**

**class sparseMatrix**

**{**

**int A[10][10];**

**int row;**

**public:**

**void Diagonal();**

**void Lower\_Triangular();**

**void Upper\_Triangular();**

**void Symmetric();**

**void display();**

**};**

**void sparseMatrix :: Diagonal()**

**{**

**cout<<"input number of rows :";**

**cin>>row;**

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

**{**

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

**{**

**if(i!=j)**

**A[i][j]=0;**

**else**

**{**

**cout<<"Input element at "<<i+1<<"th row and"<<j+1<<"th column";**

**cin>>A[i][j];**

**}**

**}**

**}**

**cout<<"diagonal matrix is"<<endl;**

**display();**

**}**

**void sparseMatrix ::Lower\_Triangular()**

**{**

**cout<<"input number of rows :";**

**cin>>row;**

**cout<<"lower triangulat matrix"<<endl;**

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

**{**

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

**{**

**if(i<j)**

**A[i][j]=0;**

**else**

**{**

**cout<<"Input element at "<<i+1<<"th row and"<<j+1<<"th column";**

**cin>>A[i][j];**

**}**

**}**

**}**

**cout<<"lower triangular matrix is"<<endl;**

**display();**

**}**

**void sparseMatrix :: Upper\_Triangular()**

**{**

**cout<<"input number of rows :";**

**cin>>row;**

**cout<<"upper triangulat matrix"<<endl;**

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

**{**

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

**{**

**if(i>j)**

**A[i][j]=0;**

**else**

**{**

**cout<<"Input element at "<<i+1<<"th row and "<<j+1<<"th column";**

**cin>>A[i][j];**

**}**

**}**

**}**

**cout<<"upper triangular matrix is"<<endl;**

**display();**

**}**

**void sparseMatrix :: Symmetric()**

**{**

**cout<<"input number of rows :";**

**cin>>row;**

**cout<<"symmetric matrix"<<endl;**

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

**{**

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

**{**

**if(i<=j)**

**{cout<<"Input element at "<<i+1<<"th row and "<<j+1<<"th column";**

**cin>>A[i][j];}**

**}**

**}**

**cout<<"symmetric matrix is"<<endl;**

**display();**

**}**

**void sparseMatrix :: display()**

**{**

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

**{**

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

**{**

**if(i>j)**

**A[i][j]=A[j][i];**

**cout<<A[i][j]<<"\t";**

**}**

**cout<<endl;**

**}**

**}**

**int main()**

**{**

**int ch;**

**char m;**

**sparseMatrix S;**

**do**

**{**

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

**cout<<"\n\n\t\t\t1. DIAGONAL MATRIX"<<endl;**

**cout<<"\n\n\t\t\t2. LOWER TRIANGULAR MATRIX"<<endl;**

**cout<<"\n\n\t\t\t3. UPPER TRIANGULAR MATRIX"<<endl;**

**cout<<"\n\n\t\t\t4. SYMMETRIX MATRIX"<<endl;**

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

**cin>>ch;**

**switch(ch)**

**{**

**case 1:**

**S.Diagonal();**

**break;**

**case 2:**

**S.Lower\_Triangular();**

**break;**

**case 3:**

**S.Upper\_Triangular();**

**break;**

**case 4:**

**S.Symmetric();**

**break;**

**}**

**cout<<"do you want to continue:(y/n)";**

**cin>>m;**

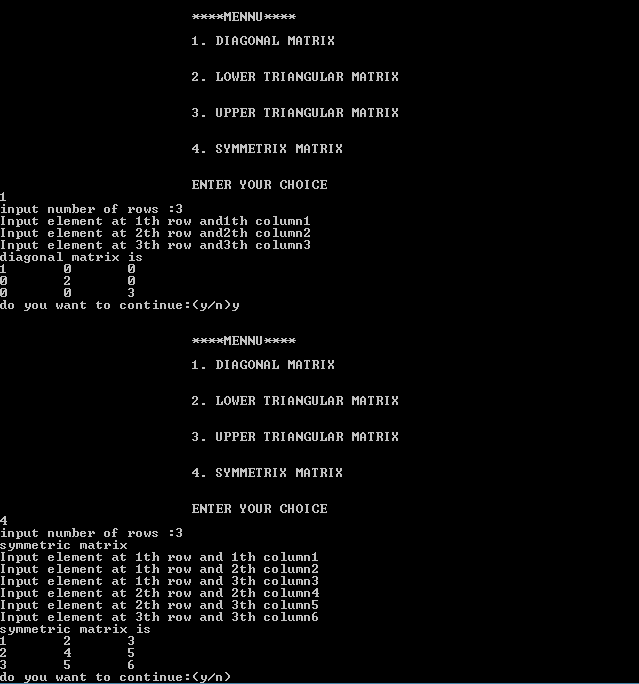
**}while(m!='n');**

**system("pause");**

**return 0;**

**}**

**OUTPUT:**

****

**Q2 B^R USING RECUESION**

**#include<iostream>**

**using namespace std;**

**int power(int b,int r){**

**if(r==1)**

**return b;**

**return b\*power(b,r-1);**

**}**

**int main(){**

**int b,r;**

**cout<<"\n\nENTER VALUE OF B-";**

**cin>>b;**

**cout<<"\n\nENTER THE VALUE OF R-";**

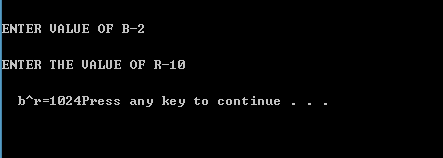
**cin>>r;**

**cout<<"\n\n b^r="<<power(b,r);**

**system("pause");**

**return 0;}**

**OUTPUT-**

****

**Q3 REVERSING A STRING**

**#include<iostream>**

**using namespace std;**

**#include<string.h>**

**void printReverse(string s,int k)**

**{ if(k==s.length())**

**return;**

**printReverse(s,k+1);**

**cout<<s[k];}**

**int main(){**

**string s;**

**cout<<"ENTER A STRING:";**

**getline(cin,s);**

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

**cout<<"\n\nREVERSED STRING IS:";**

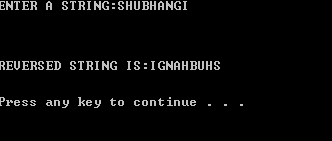
**printReverse(s,0);**

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

**system("pause");**

**return 0;}**

**OUTPUT-**

****

**Q4 STACK USING LINKED LIST**

**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();**

**}**

**}**

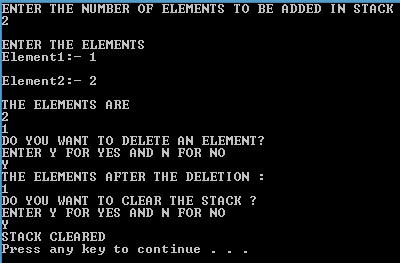
**else**

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

**system("pause");**

**return 0;**

**}**

****

**Q5 STACK USING TEMPLATE**

**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";**

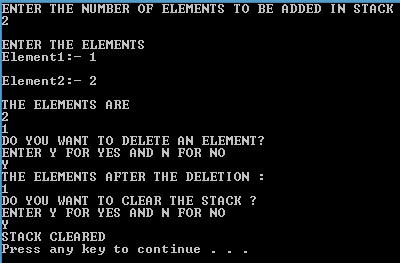
**cout<<"\n\n\t\t";**

**system("pause");**

**return 0;**

**}**

**OUTPUT-**

****

**Q6 CIRCULAR QUEUE BY ARRAY IMPLEMENTATION**

**#include<iostream>**

**using namespace std;**

**#define Maxsize 20**

**template <class T>**

**class CQueue**

**{**

**private:**

**T queue[Maxsize];**

**int rear, front;**

**public:**

**CQueue()**

**{**

**rear = -1;**

**front = 0;**

**}**

**void Enqueue(T n);**

**int CQsize();**

**int FullQ();**

**int EmptyQ();**

**T Dequeue();**

**void show();**

**};**

**template <class T>**

**void CQueue<T> :: Enqueue(T n)**

**{**

**if(FullQ())**

**cout<<"ERROR!! QUEUE IS ALREADY FULL";**

**else**

**{**

**rear = (++rear)%Maxsize;**

**queue[rear] = n;**

**}**

**}**

**template <class T>**

**T CQueue<T> :: Dequeue()**

**{**

**int n;**

**if(EmptyQ())**

**{**

**cout<<"\n\nERROR!!QUEUE IS EMPTY";**

**return NULL;**

**}**

**else**

**{**

**n = queue[front];**

**front = (++front)%Maxsize;**

**return n;**

**}**

**}**

**template <class T>**

**int CQueue<T> :: CQsize()**

**{**

**int count = 0;**

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

**return count;**

**for(int i= front;i!=rear;(++i)%Maxsize)**

**count++;**

**return ++count;**

**}**

**template <class T>**

**int CQueue<T> :: FullQ()**

**{**

**return (CQsize() == Maxsize);**

**}**

**template <class T>**

**int CQueue<T> :: EmptyQ()**

**{**

**return (CQsize() == 0);**

**}**

**template <class T>**

**void CQueue<T> :: show()**

**{**

**cout<<"\n\nQUEUE IS=>\n\n";**

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

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

**}**

**int main()**

**{**

**char ch='y';**

**int choice;**

**CQueue<int> CQ;**

**int n;**

**do**

**{**

**cout<<"\n\n\*\*\*\*\*\*MENU\*\*\*\*\*\*";**

**cout<<"\n\n1.INSERT";**

**cout<<"\n\n2.DELETE";**

**cout<<"\n\n3.SHOW";**

**cout<<"\n\nENTER YOUR CHOICE:\n";**

**cin>>choice;**

**switch(choice)**

**{**

**case 1:**

**{**

**cout<<"\n\nENTER THE ELEMENT YOU WANT TO INSERT";**

**cin>>n;**

**CQ.Enqueue(n);**

**break;**

**}**

**case 2:**

**{**

**n = CQ.Dequeue();**

**cout<<"\n\n"<<n<<" DELETED FROM QUEUE!!";**

**break;**

**}**

**case 3:**

**{**

**CQ.show();**

**break;**

**}**

**default:**

**cout<<"\n\nOOPS!!WRONG CHOICE";**

**}**

**cout<<"\n\nDO YOU WANT TO ENTER MORE(Y/y):";**

**cin>>ch;**

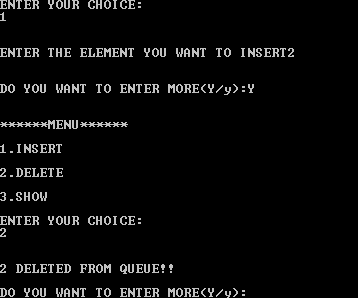
**}while(ch=='y'||ch=='Y');**

**system("pause");**

**return 0;**

**}**

**OUTPUT-**

****

**Q7 CIRCULAR QUEUE USING LINKED LIST**

**#include<iostream> //HEADER FILES**

**using namespace std;**

**typedef struct node**

**{**

**int info;**

**struct node \* next; //SELF-REFERENCING STRUCTURE**

**}Node;**

**typedef Node \* NPTR;**

**class Queue**

**{**

**private:**

**NPTR rear; //END OF THE QUEUE WHERE ELEMENTS WILL BE INSERTED**

**NPTR front; //END OF THE QUEUE FROM WHERE ELEMENTS WILL BE DELETED**

**public:**

**Queue() //CONSTRUCTOR**

**{**

**rear = NULL;**

**front = NULL;**

**}**

**void Enqueue(int x); //FUNCTIONS OF CLASS QUEUE**

**int Dequeue();**

**void show();**

**};**

**void Queue :: Enqueue(int x)**

**{**

**NPTR p = new node;**

**p->info = x;**

**p->next = NULL;**

**if(front == NULL)**

**{**

**front = p;**

**}**

**else**

**rear->next = p;**

**rear = p;**

**}**

**int Queue :: Dequeue()**

**{**

**int x;**

**if(front == NULL)**

**{**

**cout<<"\n EMPTY QUEUE\n";**

**return 0;**

**}**

**else**

**{**

**NPTR p = front;**

**x = p->info;**

**front = front->next;**

**delete p;**

**return x;**

**}**

**}**

**void Queue :: show()**

**{**

**NPTR q = new node;**

**q = front;**

**if(front == NULL)**

**{**

**cout<<"\nNOTHING TO DISPLAY\n";**

**}**

**else**

**{**

**while(q!=NULL)**

**{**

**cout<<endl<<q->info;**

**q = q->next;**

**}**

**}**

**}**

**int main()**

**{**

**char ch='y';**

**int choice;**

**Queue Q;**

**int n;**

**do**

**{**

**cout<<"\n\n\*\*\*\*\*\*MENU\*\*\*\*\*\*";**

**cout<<"\n\n1.INSERT";**

**cout<<"\n\n2.DELETE";**

**cout<<"\n\n3.SHOW";**

**cout<<"\n\nENTER YOUR CHOICE:\n";**

**cin>>choice;**

**switch(choice)**

**{**

**case 1:**

**{**

**cout<<"\n\nENTER THE ELEMENT YOU WANT TO INSERT";**

**cin>>n;**

**Q.Enqueue(n);**

**break;**

**}**

**case 2:**

**{**

**n = Q.Dequeue();**

**cout<<"\n\n"<<n<<" DELETED FROM QUEUE!!";**

**break;**

**}**

**case 3:**

**{**

**Q.show();**

**break;**

**}**

**default:**

**cout<<"\n\nOOPS!!WRONG CHOICE";**

**}**

**cout<<"\n\nDO YOU WANT TO ENTER MORE(Y/y):";**

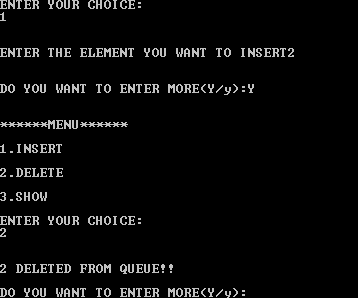
**cin>>ch;**

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

**system("pause");**

**return 0;**

**}**

****

**Q8 ADD TWO INTEGERS USING STACK**

**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;**

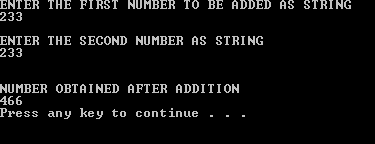
**s3.display();**

**system("pause");**

**return 0;**

**}**

**Output-**

****

**Q9 POSTFIX EXPRESSION**

**#include<iostream>**

**using namespace std;**

**#include<stdio.h>**

**#include<process.h>**

**#include<ctype.h> //FOR isdigit()**

**#define Maxsize 20**

**class Stack**

**{**

**private:**

**int stack[Maxsize];**

**int top,elt;**

**char \*s; //FOR STORING THE BASE ADDRESS OF CHARACTER ARRAY CONTAINING POSTFIX EXPRESSION**

**public:**

**Stack()**

**{**

**top = -1;**

**}**

**void expression(char \*str);**

**void push(int element);**

**int pop();**

**void evaluate();**

**void show();**

**};**

**void Stack :: expression(char \*str)**

**{**

**s=str;**

**}**

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

**{**

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

**{**

**cout<<"\n ERROR, STACK IS FULL!!";**

**}**

**else**

**{**

**top++;**

**stack[top]=element;**

**}**

**}**

**int Stack :: pop()**

**{**

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

**{**

**cout<<"\n ERROR, STACK IS EMPTY";**

**return NULL;**

**}**

**else**

**{**

**return stack[top--];**

**}**

**}**

**void Stack :: evaluate()**

**{**

**int elt1,elt2,elt3;**

**while(\*s)**

**{**

**if(\*s ==' ' || \*s == '\t')**

**{**

**s++;**

**continue;**

**}**

**else if(isdigit(\*s))**

**{**

**elt = \*s-'0';**

**push(elt);**

**}**

**else**

**{**

**elt1=pop();**

**elt2=pop();**

**switch(\*s)**

**{**

**case '+':**

**elt3 = elt1 + elt2;**

**break;**

**case '-':**

**elt3 = elt1 - elt2;**

**break;**

**case '\*':**

**elt3 = elt1 \* elt2;**

**break;**

**case '/':**

**elt3 = elt1 / elt2;**

**break;**

**default:**

**cout<<"OOPS!! UNKNOWN OPERATOR...";**

**exit(-1);**

**}**

**push(elt3);**

**}**

**s++;**

**}**

**}**

**void Stack :: show()**

**{**

**cout<<"\n\nRESULT IS =>"<<pop();**

**}**

**int main()**

**{**

**char expr[Maxsize];**

**Stack st;**

**cout<<"\n\nENTER THE POSTFIX EXPRESSION=>";**

**gets(expr);**

**st.expression(expr);**

**st.evaluate();**

**st.show();**

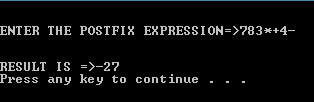
**cout<<endl;**

**system("pause");**

**return 0;**

**}**

**OUTPUT-**

****

**Q10 IMPLEMENTING LINKED LIST USING TEMPLATES**

**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**

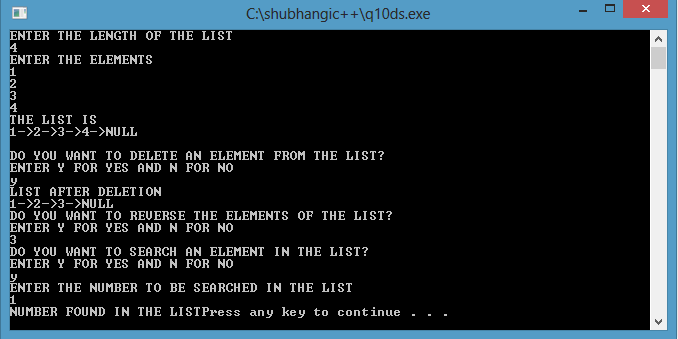
**cout<<"INVALID ENTERY";**

**system("pause");**

**return 0;**

**}**

**OUTPUT-**

****

**Q11 IMPLEMENT DOUBLY LINKED LIST**

**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);**

**}**

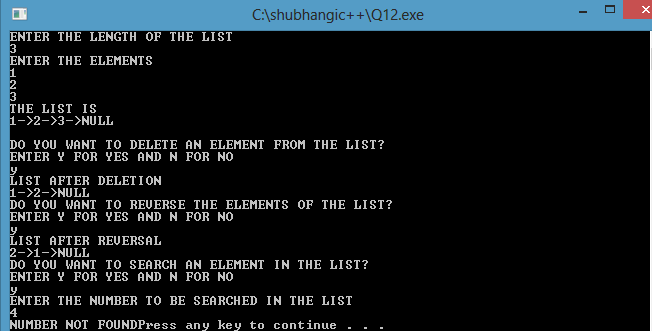
**}**

**system("pause");**

**return 0;**

**}**

**OUTPUT-**

****

**Q12 IMPLEMENT CIRCULAR LINKED LIST**

**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);**

**}**

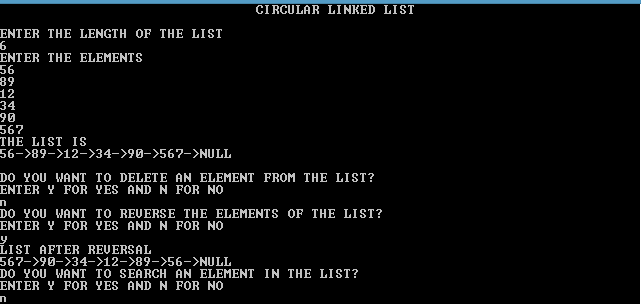
**}**

**system("pause");**

**return 0;**

**}**

**OUTPUT-**

****

**Q13 ADDING TWO POLYNOMIALS**

**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;**

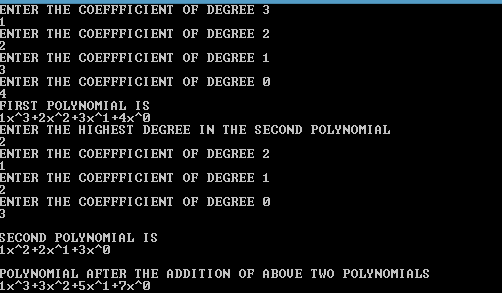
**l3.display();**

**system("pause");**

**return 0;**

**}**

**OUTPUT-**

****

**Q14 ORDERED LINKED LIST**

**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";**

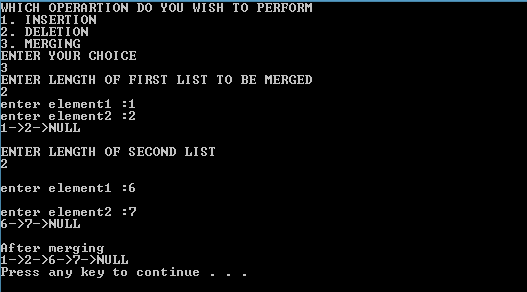
**}**

**system("pause");**

**return 0;**

**}**

**OUTPUT-**

****

**Q15 BINARY SEARCH TREE**

**#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;**

**}**

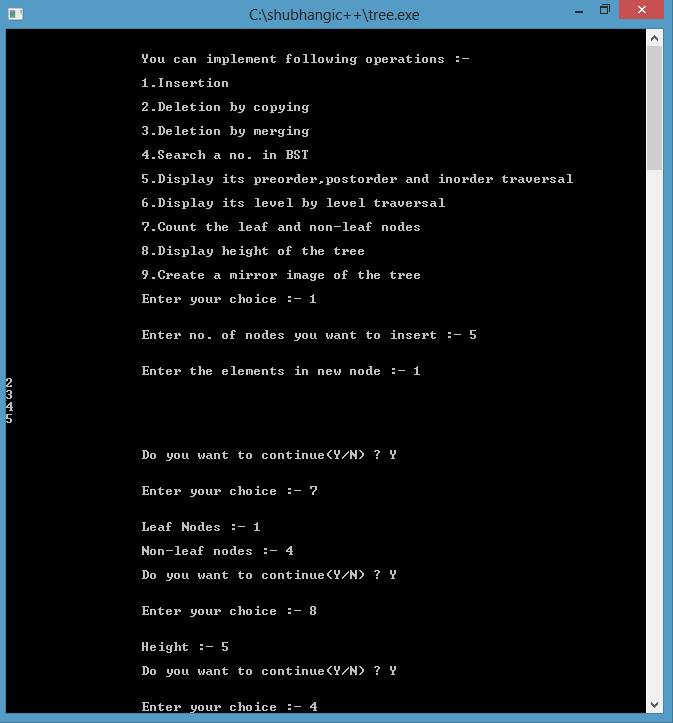
**while(ch=='Y'||ch=='y');**

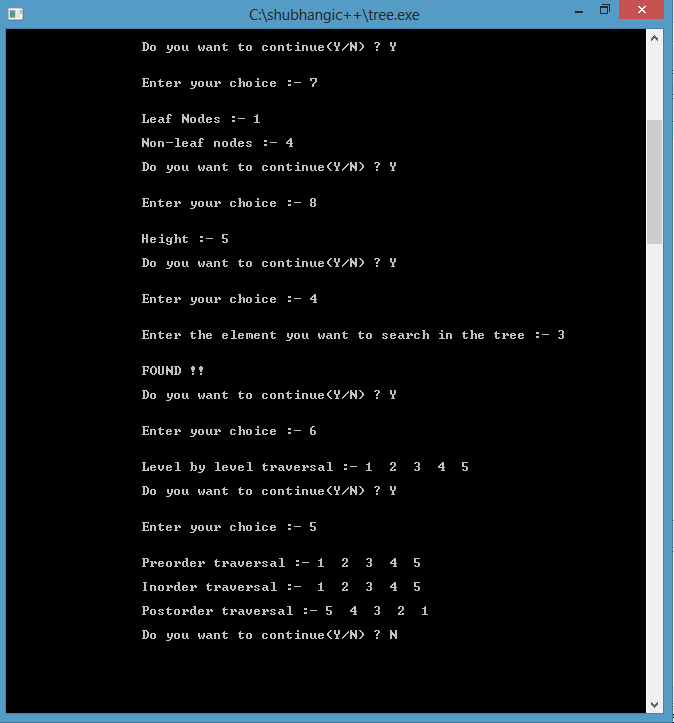
**cout<<"\n\n\t\t ";**

**system("pause");**

**return 0;**

**}**

****

****

**Q16 SORTING AND SEARCHING**

**#include<iostream>**

**using namespace std;**

**int partition(int a[], int p, int r);**

**void quicksort(int a[],int p,int r);**

**class Array**

**{**

**int array[10];**

**int size;**

**public:**

**Array()**

**{**

**size=0;**

**}**

**void Bubblesort();**

**void Insertionsort();**

**void Selectionsort();**

**void Quicksort();**

**void Mergesort();**

**void Linearsearch();**

**void Binarysearch();**

**};**

**void Array ::Bubblesort()**

**{**

**int temp;**

**cout<<"\nENTER THE SIZE OF ARRAY";**

**cin>>size;**

**cout<<"\nENTER THE ELEMENTS OF ARRAY";**

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

**{**

**cin>>array[i];**

**}**

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

**{**

**for(int j=0;j<size-i-1;j++)**

**{**

**if(array[j]>array[j+1])**

**{**

**temp=array[j];**

**array[j]=array[j+1];**

**array[j+1]=temp;**

**}**

**}**

**}**

**cout<<"\nARRAY IN SORTED ORDER"<<endl;**

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

**{**

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

**}**

**}**

**void Array ::Insertionsort()**

**{**

**cout<<"\nENTER THE SIZE OF ARRAY";**

**cin>>size;**

**cout<<"\nENTER THE ELEMENTS OF ARRAY";**

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

**{**

**cin>>array[i];**

**}**

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

**{**

**int key = array[i];**

**int j = i-1;**

**while(j>=0 && key < array[j])**

**{**

**array[j+1] = array[j];**

**j--;**

**}**

**array[j+1] = key;**

**}**

**cout<<"\nARRAY IN SORTED ORDER"<<endl;**

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

**{**

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

**}**

**}**

**void Array :: Selectionsort()**

**{**

**cout<<"\nENTER THE SIZE OF ARRAY";**

**cin>>size;**

**cout<<"\nENTER THE ELEMENTS OF ARRAY";**

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

**{**

**cin>>array[i];**

**}**

**int i, j, min, temp;**

**for(i=0; i < size-1**

**; i++ )**

**{**

**min = i; //setting min as i**

**for(j=i+1; j < size; j++)**

**{**

**if(array[j] < array[min]) //if element at j is less than element at min position**

**{**

**min = j; //then set min as j**

**}**

**}**

**temp = array[i];**

**array[i] = array[min];**

**array[min] = temp;**

**}**

**cout<<"\nARRAY IN SORTED ORDER"<<endl;**

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

**{**

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

**}**

**}**

**void Array ::Quicksort()**

**{**

**cout<<"\nENTER THE SIZE OF ARRAY";**

**cin>>size;**

**cout<<"\nENTER THE ELEMENTS OF ARRAY";**

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

**{**

**cin>>array[i];**

**}**

**int p = 0;**

**int r = size-1;**

**quicksort(array,p,r);**

**cout<<"\nARRAY IN SORTED ORDER"<<endl;**

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

**{**

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

**}**

**}**

**void quicksort(int a[],int p,int r)**

**{**

**if(p < r)**

**{**

**int q;**

**q = partition(a, p, r);**

**quicksort(a, p, q);**

**quicksort(a, q+1, r);**

**}**

**}**

**int partition(int a[], int p, int r)**

**{**

**int i, j, pivot, temp;**

**pivot = a[p];**

**i = p;**

**j = r;**

**while(1)**

**{**

**while(a[i] < pivot && a[i] != pivot)**

**i++;**

**while(a[j] > pivot && a[j] != pivot)**

**j--;**

**if(i < j)**

**{**

**temp = a[i];**

**a[i] = a[j];**

**a[j] = temp;**

**}**

**else**

**{**

**return j;**

**}**

**}**

**}**

**void Array :: Mergesort()**

**{**

**int a[30];**

**int size1;**

**cout<<"\nENTER THE SIZE OF ARRAY 1";**

**cin>>size;**

**cout<<"\nENTER THE ELEMENTS OF ARRAY 1(IN ASCENDING ORDER)";**

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

**{**

**cin>>array[i];**

**}**

**cout<<"\nENTER THE SIZE OF ARRAY 2";**

**cin>>size1;**

**cout<<"\nENTER THE ELEMENTS OF ARRAY 2(IN ASCENDING ORDER)";**

**for(int i=0;i<size1;i++)**

**{**

**cin>>a[i];**

**}**

**int n=size+size1;**

**int m=0,k=0,l=0;**

**int answer[n];**

**while ( m < size && l < size1)**

**{**

**if (array[m] < a[l])**

**{**

**answer[k] = array[m];**

**m++;**

**}**

**else**

**{**

**answer[k] = a[l];**

**l++;**

**}**

**k++;**

**}**

**while (m < size)**

**{**

**answer[k] = array[m];**

**m++;**

**k++;**

**}**

**while (l < size1)**

**{**

**answer[k] = a[l];**

**l++;**

**k++;**

**}**

**cout<<"\nARRAY AFTER MERGING IN SORTED ORDER"<<endl;**

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

**{**

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

**}**

**}**

**void Array :: Linearsearch()**

**{**

**int element;**

**int flag=0;**

**cout<<"\nENTER THE SIZE OF ARRAY";**

**cin>>size;**

**cout<<"\nENTER THE ELEMENTS OF ARRAY";**

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

**{**

**cin>>array[i];**

**}**

**cout<<"\n ARRAY IS:\n";**

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

**{**

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

**}**

**cout<<"\nENTER THE ELEMENT YOU WANT TO SEARCH:";**

**cin>>element;**

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

**{**

**if(array[i] == element)**

**{**

**cout<<"\n ELEMENT FOUND AT POSITION: "<<i+1;**

**flag = 1;**

**break;**

**}**

**}**

**if(flag==0)**

**cout<<"\n\nOOPS!ELEMENT NOT FOUND";**

**}**

**void Array ::Binarysearch()**

**{**

**int found=0;**

**int beg=0;**

**int last=size-1;**

**int val;**

**cout<<"\n\nENTER NO. OF ELEMENTS YOU WANT TO ENTER=>";**

**cin>>size;**

**cout<<"\n\nENTER THE ELEMENTS IN THE ASCENDING ORDER=>";**

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

**{**

**cout<<"\n\nENTER ELEMENT "<<i+1<<":";**

**cin>>array[i];**

**}**

**cout<<"\n\nENTER THE NUMBER YOU WANT TO SEARCH:";**

**cin>>val;**

**while(beg<=last)**

**{**

**int mid=(beg+last)/2;**

**if(array[mid]==val)**

**{**

**cout<<"\n\nNUMBER FOUND AT POSITION "<<mid+1;**

**found =1;**

**break;**

**}**

**else if(val>array[mid])**

**beg=mid+1;**

**else if(val<array[mid])**

**last=mid-1;**

**}**

**if(found==0)**

**cout<<"\n\nNUMBER NOT FOUND";**

**}**

**int main()**

**{**

**Array a;**

**char ch='y';**

**int choice;**

**do**

**{**

**cout<<"\n\n\n\*\*\*\*\*\*MENU\*\*\*\*\*\*";**

**cout<<"\n\n1.BUBBLE SORT";**

**cout<<"\n\n2.INSERTION SORT";**

**cout<<"\n\n3.SELECTION SORT";**

**cout<<"\n\n4.QUICK SORT";**

**cout<<"\n\n5.MERGE SORT";**

**cout<<"\n\n6.LINEAR SEARCH";**

**cout<<"\n\n7.BINARY SEARCH";**

**cout<<"\n\nENTER YOUR CHOICE";**

**cin>>choice;**

**switch (choice)**

**{**

**case 1:**

**a.Bubblesort();**

**break;**

**case 2:**

**a.Insertionsort();**

**break;**

**case 3:**

**a.Selectionsort();**

**break;**

**case 4:**

**a.Quicksort();**

**break;**

**case 5:**

**a.Mergesort();**

**break;**

**case 6:**

**a.Linearsearch();**

**break;**

**case 7:**

**a.Binarysearch();**

**break; ;**

**default:**

**cout<<"OOPS!!WRONG CHOICE";**

**}**

**cout<<"\n\nDO YOU WANT TO ENTER MORE";**

**cin>>ch;**

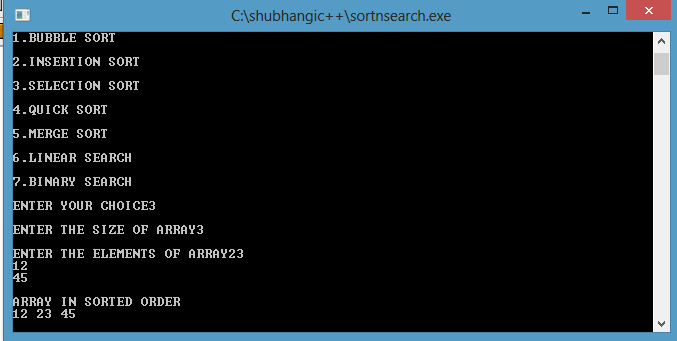
**}while(ch=='y'||ch=='Y');**

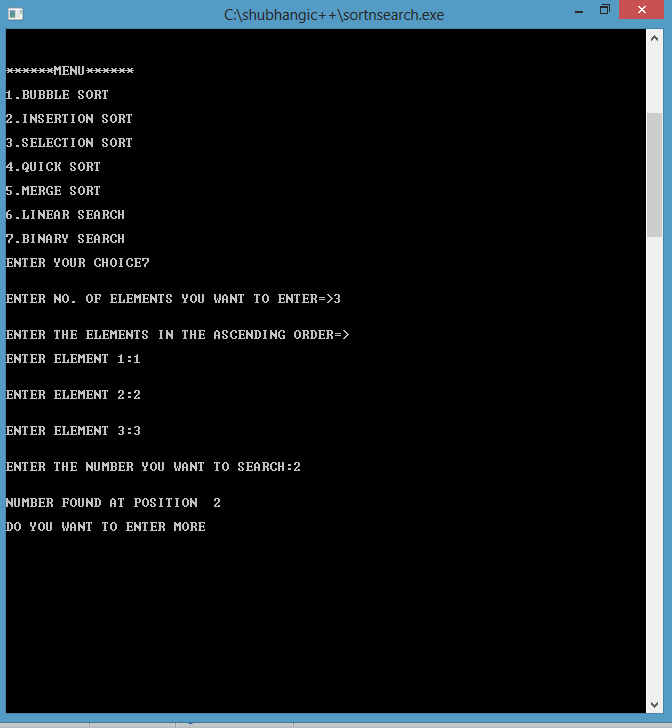
**system("pause");**

**return 0;**

**}**

**OUTPUT-**

****

****