**EXPERIMENT 1**

#include<iostream>

using namespace std;

int main()

{

int a[20],n,i,m,pos,ele,ch,flag=0;

cout<<"Enter the size of array: ";

cin>>n;

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

{

cin>>a[i];

}

cout<<"\n1.insertion\n2.deletion\n3.searching\n4.display\n";

cin>>ch;

while(ch!=0)

{

switch(ch)

{

case 1:

cout<<"Enter the position and element you want to insert\n";

cin>>pos>>ele;

m=n-pos-1;

for(i=n;i>=m;i--)

{

a[i+1]=a[i];

}

a[pos]=ele;

n=n+1;

break;

case 2:

cout<<"Enter the position of element you want to delete\n";

cin>>pos;

for(i=pos;i<n;i++)

{

a[i]=a[i+1];

}

n=n-1;

break;

case 3:

flag=0;

cout<<"\nEnter the element to search: ";

cin>>ele;

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

{

if(a[i]==ele)

{

cout<<"\nPosition is: "<<i+1;

flag=1;

break;

}

}

if(flag==0)

cout<<"\nElement not found";

break;

case 4:

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

{

cout<<a[i]<<" ";

}

cout<<endl;

break;

cout<<"\nEnter choice again: ";

cin>>ch;

}

}

return 0;

}

**EXPERIMENT 2**

#include<iostream>

using namespace std;

int main()

{

int a[20],n,ele,flag=0;

cout<<"Enter the size of array: ";

cin>>n;

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

{

cin>>a[i];

}

cout<<"Enter the element to search: ";

cin>>ele;

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

{

if(a[i]==ele)

{

cout<<"Position of "<<ele<<" is: "<<i+1;

flag=1;

break;

}

}

if(flag==0)

cout<<"Element not found\n";

return 0;

}

**EXPERIMENT 3**

#include<iostream>

using namespace std;

int main()

{

int a[20],n,ele,flag=0,temp;

int mid,min,max;

cout<<"Enter the size of array: ";

cin>>n;

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

{

cin>>a[i];

}

cout<<"Array is:\n";

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

{

cout<<a[i]<<" ";

}

cout<<endl;

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

{

for(int j=0;j<n-1;j++)

{

if(a[j]>a[j+1])

{

temp=a[j];

a[j]=a[j+1];

a[j+1]=temp;

}

}

}

cout<<"\nAfter sorting the array: \n";

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

{

cout<<a[i]<<" ";

}

cout<<endl;

cout<<"\nEnter the element to search: ";

cin>>ele;

min=0;

max=n-1;

mid=(min+max)/2;

if(min<max)

{

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

{

if(a[mid]==ele)

{

cout<<"Position of "<<ele<<" is: "<<mid+1;

flag=1;

break;

}

else if(a[mid]>ele)

{

max=mid-1;

}

else if(a[mid]<ele)

{

min=mid+1;

}

mid=(min+max)/2;

}

}

if(flag==0)

{

cout<<"Element not found ";

}

return 0;

}

**EXPERIMENT 4**

#include<iostream>

using namespace std;

struct node

{

int data;

struct node \*prev;

struct node \*next;

};

struct linked\_list

{

node \*head,\*tail;

linked\_list()

{

head=NULL;

tail=NULL;

}

void insert\_beg();

void display();

void insert2();

void deletion();

void searching();

};

node\* createnode(int x)

{

node \*temp=new node;

temp->data=x;

temp->prev=NULL;

temp->next=NULL;

return temp;

}

void linked\_list::insert\_beg()

{

int n;

cout<<"Enter the no. of elements you want to enter: ";

cin>>n;

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

{

int ele;

cin>>ele;

node \*p,\*ptr;

p=createnode(ele);

if(head==NULL)

{

head=p;

tail=p;

}

else

{

p->next=head;

head->prev=p;

head=p;

}

}

tail->next=head;

head->prev=tail;

}

void linked\_list::insert2()

{

int ch,ele,x;

cout<<"\n1.insertion in beginning\n2.insertion at end\n3.insertion in between\n";

cin>>ch;

node \*p=NULL,\*temp,\*ptr;

while(ch!=0)

{

switch(ch)

{

case 1:

cout<<"Enter the element to add: ";

cin>>ele;

p=createnode(ele);

p->next=head;

head->prev=p;

head=p;

tail->next=head;

head->prev=tail;

break;

case 2:

cout<<"Enter the element to add: ";

cin>>ele;

p=createnode(ele);

tail->next=p;

p->prev=tail;

tail=p;

tail->next=head;

head->prev=tail;

break;

case 3:

cout<<"\nEnter an element you want to add: ";

cin>>ele;

cout<<"Enter the element after you want to add: ";

cin>>x;

p=createnode(ele);

ptr=head;

while(ptr->data!=x)

{

ptr=ptr->next;

}

temp=ptr->next;

ptr->next=p;

p->prev=ptr;

temp->prev=p;

p->next=temp;

break;

default:

cout<<"Wrong choice\nPlease enter 1-3 else 0 to exit.";

break;

}

cout<<"\nEnter choice to insert: ";

cin>>ch;

}

cout<<"\nElement Inserted";

}

void linked\_list::deletion()

{

int ele;

cout<<"\nEnter an element you want to delete: ";

cin>>ele;

node \*temp,\*p;

p=head;

if(head->data==ele) //Deletion in beginning

{

head=head->next;

tail->next=head;

head->prev=tail;

}

else if(tail->data==ele) //Deletion at end

{

tail=tail->prev;

tail->next=head;

head->prev=tail;

}

else //Deletion in between

{

while(p->data!=ele)

{

p=p->next;

}

temp=p->prev;

temp->next=p->next;

p->next->prev=temp;

}

cout<<"\nElement deleted";

}

void linked\_list::searching()

{

int ele;

cout<<"Enter element you want to search: ";

cin>>ele;

node \*p;

p=head;

while(p->data!=ele && p->next!=head)

{

p=p->next;

}

if(p->data==ele)

cout<<"\nNode exists";

else

cout<<"\nNode does not exist";

}

void linked\_list::display()

{

node \*ptr=head;

if(head==NULL)

cout<<"Linked List is empty";

else

while(ptr->next!=head)

{

cout<<ptr->data<<"->";

ptr=ptr->next;

}

cout<<ptr->data;

}

int main()

{

int ch;

linked\_list l1;

l1.insert\_beg();

cout<<"\nEnter choice:\n1 for insertion\n2 for deletion\n3 for searching\n4 for display\n";

cin>>ch;

while(ch!=0)

{

switch(ch)

{

case 1:

l1.insert2();

break;

case 2:

l1.deletion();

break;

case 3:

l1.searching();

break;

case 4:

l1.display();

break;

default:

cout<<"\nWrong choice\n";

break;

}

cout<<"\nEnter choice again:\n1 for insertion\n2 for deletion\n3 for searching\n4 for display\n";

cin>>ch;

}

}

**EXPERIMENT 5**

#include<iostream>

using namespace std;

#define MAX 6

int Stack[MAX];

int top = -1;

void push(int x);

void pop();

void display();

int main ()

{

int ch,val;

ch=-1;

while(ch!=4)

{

cin>> ch;

switch(ch)

{

case 1:// cout<<”Enter the element to push”;

cin>>val;

push(val);

break;

case 2: // cout<<”Enter the element to pop”;

pop();

break;

case 3:// cout<<”Display the stack elements”;

display();

break;

default:

break;

}

cout<<"enter choice"<<endl;

}

return 0;

}

void push(int x)

{

if(top==MAX)

cout<<"overflow "<<endl;

else

{

Stack[top+1]=x;

top++;

}

}

void pop()

{

if(top==-1)

cout<<"underflow"<<endl;

else

{

cout<<"deleted element is"<<" "<<Stack[top--]<<endl;

}

}

void display()

{

for(int i=top;i>-1;i--)

{

cout<<Stack[i]<<" ";

}

cout<<endl;

}

**EXPERIMENT 6**

#include<iostream>

using namespace std;

class convert

{

char a[20];

int top;

public:

convert()

{

top=-1;

}

void push(int x)

{

top++;

a[top]=x;

}

char pop()

{

return a[top--];

}

int priority(char p)

{

if(p=='(' )

return 0;

else if(p=='+'||p=='-')

return 1;

else if(p=='\*'|| p=='/')

return 2;

else if(p=='^')

return 3;

}

void in\_po(char exp[20])

{

char \*e;

char x;

e=exp;

while(\*e!='\0')

{

if(isalnum(\*e))

{

cout<<\*e;

}

else if(\*e=='(')

{

push(\*e);

}

else if(\*e==')')

{

while((x=pop())!='(')

cout<<x;

}

else

{

while(priority(a[top])>=priority(\*e))

{

cout<<pop();

}

push(\*e);

}

e++;

}

while(top!=-1)

{

x=pop();

cout<<x;

}

}

};

class s1

{

public:

char a[20];

int top;

s1()

{

top=-1;

}

void push(int x)

{

a[++top]=x;

}

int pop()

{

return a[top--];

}

void in\_po(char exp[20])

{

char \*e;

char x;

int num;

e=exp;

while(\*e!='\0')

{

if(isalnum(\*e))

{

num=\*e-48;

push(num);

}

else

{

int a=pop();

int b=pop();

int c;

switch(\*e)

{

case '+':

c=a+b;

push(c);

break;

case '-':

c=a-b;

push(c);

break;

case '/':

c=a/b;

push(c);

break;

case '\*':

c=a\*b;

push(c);

break;

case '^':

c=a^b;

push(c);

break;

}

}

e++;

}

while(top!=-1)

{

cout<<pop();

}

}

};

int main()

{

int ch;

cout<<"n1 for Infix to postfix conversion\n2 for Postfix Evaluation\n";

cin>>ch;

if(ch==1)

{

convert r;

char ss[20];

cin>>ss;

r.in\_po(ss);

}

else if(ch==2)

{

s1 r;

char ss[20];

cin>>ss;

r.in\_po(ss);

}

return 0;

}

**EXPERIMENT 7**

#include<iostream>

using namespace std;

struct node

{

int data;

struct node \*next;

};

struct linked\_list

{

node \*front,\*rear;

linked\_list()

{

front=NULL;

rear=NULL;

}

void insert\_beg();

void display();

node\* createnode(int);

void del();

};

node\* linked\_list::createnode(int x)

{

node \*temp=new node;

temp->data=x;

temp->next=NULL;

return temp;

}

void linked\_list::insert\_beg()

{

int ele;

cin>>ele;

node \*p,\*ptr;

p=createnode(ele);

if(front==NULL)

{

front=p;

rear=p;

}

else

{

rear->next=p;

rear=p;

}

}

void linked\_list::display()

{

rear->next=front;

node \*temp=front;

if(front==NULL)

{

cout<<"Underflow";

cout<<"\nQueue is empty";

}

else

{

while(temp->next!=front)

{

cout<<temp->data<<"->";

temp=temp->next;

}

cout<<temp->data;

}

}

void linked\_list::del()

{

if(front==rear)

{

cout<<"Deleted element is: "<<front->data<<endl;

front=NULL;

}

else if(front==NULL)

{

cout<<"Underflow";

cout<<"\nQueue is empty";

}

else

{

cout<<"Deleted element is: "<<front->data<<endl;

node\* temp=front;

front=front->next;

delete temp;

}

}

int main()

{

int ch,n,count=0;

linked\_list l1;

cout<<"Enter the size of queue ";

cin>>n;

cout<<"\n1.Insert\n2.Display\n3.Delete\n";

cin>>ch;

while(ch)

{

if(ch==1 )

{

if(count==n)

{

cout<<"Overflow";

}

else

{

l1.insert\_beg();

count++;

}

}

else if(ch==2)

{

l1.display();

}

else if(ch==3)

{

l1.del();

count--;

}

cout<<"Enter choice again: ";

cin>>ch;

}

}

**EXPERIMENT 8**

#include<iostream>

using namespace std;

void insertionsort(int a[],int n)

{

int i,j,k,temp,m;

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

{

m=a[i];

j=i;

while(a[j-1]>m && j>0)

{

a[j]=a[j-1];

j--;

}

a[j]=m;

}

}

int main()

{

int i,a[20],n;

cout<<"Enter no of elements in an array: ";

cin>>n;

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

{

cin>>a[i];

}

insertionsort(a,n);

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

{

cout<<a[i]<<" ";

}

int item;

cout<<"\nEnter ITEM to be inserted : ";

cin>>item;

i = n-1;

while(item<a[i] && i>=0)

{

a[i+1] = a[i];

i--;

}

a[i+1] = item;

n++;

cout<<"\nAfter insertion array is :\n";

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

{

cout<<a[i]<<" ";

}

return 0;

}

**EXPERIMENT 9**

#include<iostream>

using namespace std;

void swap(int\* a, int\* b)

{

int temp=\*a;

\*a=\*b;

\*b=temp;

}

int partition(int arr[],int low,int high)

{

int pivot=arr[high];

int i=low - 1;

for(int j=low;j<=high-1;j++)

{

if(arr[j]<=pivot)

{

i++;

swap(&arr[i],&arr[j]);

}

}

swap(&arr[i+1],&arr[high]);

return (i+1);

}

void quickSort(int arr[],int low,int high)

{

if (low<high)

{

int pi=partition(arr,low,high);

quickSort(arr,low,pi-1);

quickSort(arr,pi+1,high);

}

}

int main()

{

int a[100],i,j,m,s;

cout<<"Enter no of elements to be entered in array: ";

cin>>s;

for(i=0;i<s;i++)

{

cin>>a[i];

}

quickSort(a,0,s-1);

for (i=0;i<s;i++)

{

cout<<a[i]<<" ";

}

}

**EXPERIMENT 10**

#include<iostream>

using namespace std;

int merge(int \*A, int M, int \*B, int N, int \*C)

{

int p = 0;

int q = 0;

int k = 0;

int c=0;

while ( p < M && q < N)

{

if (A[p] < B[q])

{

C[k++] = A[p++];

c++;

}

else

{

C[k++] = B[q++];

c++;

}

}

while ( p < M)

{

C[k++] = A[p++];

}

while ( q < N)

{

C[k++] = B[q++];

}

return c;

}

int main()

{

int i,M,N;

cin >> M >> N;

int A[M],B[N],C[M+N];

int X;

for(i=0;i<=M-1;i++)

cin >> A[i];

for(i=0;i<=N-1;i++)

cin >> B[i];

X = merge(A,M,B,N,C);

for(i=0;i<=M+N-1;i++)

cout << C[i] << " ";

cout << endl << X;

return 0;}

**EXPERIMENT 11**

#include<iostream>

using namespace std;

struct node

{

int data;

node \*left;

node \*right;

};

node \*createnode(int x)

{

node \*temp=new node;

temp->data=x;

temp->left=temp->right=NULL;

return temp;

}

node\* insertion(node \*root,int value)

{

if(root==NULL)

{

root=createnode(value);

}

else if(value < root->data)

root->left=insertion(root->left,value);

else if(value > root->data)

root->right=insertion(root->right,value);

return root;

}

void inorder(struct node \*root)

{

if (root != NULL)

{

inorder(root->left);

cout<<root->data<<" ";

inorder(root->right);

}

}

void preorder(struct node \*root)

{

if (root != NULL)

{

cout<<root->data<<" ";

preorder(root->left);

preorder(root->right);

}

}

void postorder(struct node \*root)

{

if (root != NULL)

{

postorder(root->left);

postorder(root->right);

cout<<root->data<<" ";

}

}

node\* searchnode(node \*root,int x)

{

if(root==NULL){

cout<<"Node not found";

return NULL;

}

else if(root->data==x)

{

cout<<"\nNode exists\n";

return NULL;

}

else if(x < root->data)

root->left=searchnode(root->left,x);

else

root->right=searchnode(root->right,x);

}

node\* delnode(node \*root,int x)

{

if(root==NULL)

cout<<"Node not found";

else if(root->data==x && root->right==NULL && root->left==NULL)

{

delete root;

root=NULL;

return root;

}

else if(x < root->data)

root->left=delnode(root->left,x);

else

root->right=delnode(root->right,x);

}

int findheight(node \*root)

{

int lh,rh;

if(root==NULL)

return 0;

else

{

lh=findheight(root->left);

rh=findheight(root->right);

if(lh > rh)

return (lh+1);

else

return (rh+1);

}

}

int findsize(node \*root)

{

if(root==NULL)

return 0;

else

return (findsize(root->left)+1+findsize(root->right));

}

int checkbst(node \*root)

{

if(root==NULL)

return 1;

if(root->left!=NULL && root->left->data > root->data)

return 0;

if(root->right!=NULL && root->right->data < root->data)

return 0;

if(!checkbst(root->left) || !checkbst(root->right))

return 0;

return 1;

}

int count=0;

int childnodes(node \*root)

{

if(root==NULL)

return 0;

if(root->left==NULL && root->right==NULL)

count++;

else{

childnodes(root->left);

childnodes(root->right);

}

return count;

}

int internalnodes(node \*root)

{

if(root==NULL)

return 0;

if(root->left!=NULL || root->right!=NULL)

count++;

childnodes(root->left);

childnodes(root->right);

return count;

}

node\* heightofnode(node \*root,int x)

{

if(root==NULL){

cout<<"Node not found";

return NULL;

}

else if(root->data==x)

{

int x=findheight(root);

cout<<"\nHeight of the node is: "<<x-1<<endl;

return NULL;

}

else if(x < root->data)

root->left=heightofnode(root->left,x);

else

root->right=heightofnode(root->right,x);

}

int main()

{

node \*root = NULL;

node \*ptr;

int x,n,ch;

cout<<"Enter choice\n1.Insert\n2.Search\n3.Traverse\n4.Delete\n5.Height or Depth of tree\n6.Size of the tree\n7.BST or not\n8.No of child nodes\n9.No of internal nodes\n10.Height of node\n";

cin>>ch;

while(ch)

{

switch(ch)

{

case 1:

cout<<"Enter the no of nodes: ";

cin>>n;

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

{

cin>>x;

root=insertion(root,x);

}

break;

case 2:

cout<<"\nEnter an element you want to search: ";

cin>>x;

ptr=root;

ptr=searchnode(ptr,x);

break;

case 3:

ptr=root;

cout<<"Inorder: ";

inorder(ptr);

cout<<"\nPreorder: ";

preorder(ptr);

cout<<"\nPostorder: ";

postorder(ptr);

break;

case 4:

cout<<"\nEnter an element you want to delete: ";

cin>>x;

ptr=root;

delnode(ptr,x);

break;

case 5:

ptr=root;

int h;

h=findheight(ptr);

cout<<"Height of the tree is: "<<h-1<<endl;

break;

case 6:

ptr=root;

x=findsize(ptr);

cout<<"Size of the tree: "<<x<<endl;

break;

case 7:

ptr=root;

x=checkbst(ptr);

if(x==0)

cout<<"\nTree is not a BST\n";

else

cout<<"\nIt is a BST\n";

break;

case 8:

ptr=root;

x=childnodes(ptr);

if(x==0)

cout<<"\nNo child nodes present\n";

else

cout<<"Child nodes are: "<<x<<endl;

count=0;

break;

case 9:

ptr=root;

x=internalnodes(ptr);

if(x==0)

cout<<"\nEmpty tree\n";

else

cout<<"\nInternal nodes are: "<<x+1<<endl;

count=0;

break;

case 10:

cout<<"\nEnter the node you want to find the path for: ";

cin>>x;

ptr=root;

heightofnode(ptr,x);

break;

}

cout<<"\nEnter choice: ";

cin>>ch;

}

return 0;}

**EXPERIMENT 12**

#include<iostream>

using namespace std;

void max\_heapify(int a[],int i,int n)

{

int l=2\*i+1;

int r=2\*i+2;

int largest;

if(l<n&&a[l]>a[i])

largest=l;

else

largest=i;

if(r<n&&a[r]>a[largest])

largest=r;

if(largest!=i)

{

swap(a[i],a[largest]);

max\_heapify(a,largest,n);

}

}

void build(int a[],int n)

{

for(int i=n/2-1;i>=0;i--)

max\_heapify(a,i,n);

}

void heapsort(int a[],int n)

{

int s=n;

build(a,n);

for(int i=n-1;i>=0;i--)

{

swap(a[0],a[i]);

s=s-1;

max\_heapify(a,0,s);

}

}

int main()

{

int n,i;

int a[20];

cin>>n;

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

cin>>a[i];

heapsort(a,n);

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

cout<<a[i]<<" ";

return 0;

}

**EXPERIMENT 13**

#include<iostream>

#include<stack>

#include<queue>

using namespace std;

class graph

{

int \*\*AM;

int vertices;

public:

graph(int vert)

{

vertices=vert;

AM=new int \*[vertices];

for(int i=0;i<vertices;++i)

AM[i]=new int[vertices];

for(int i=0;i<vertices;++i)

for(int j=0;j<vertices;++j)

AM[i][j]=0;

}

void create\_graph();

void in\_degree(int);

void out\_degree(int);

void depth\_first\_traversal();

void display\_graph();

void get\_neighbours(int \*,int);

void depth\_first\_traversal1();

void breadth\_first\_traversal();

};

void graph::create\_graph()

{

char op;

do

{

cout<<"Enter vertices you want to connect:"<<endl;

cout<<"Enter first vertex: ";

int vert1;

cin>>vert1;

cout<<"Enter second vertex: ";

int vert2;

cin>>vert2;

AM[vert1][vert2]=1;

AM[vert2][vert1]=1;

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

cin>>op;

}

while(op=='y' || op=='Y');

}

void graph::in\_degree(int in)

{

int count=0;

for(int i=0;i<vertices;++i)

if(AM[in][i]==1)

++count;

cout<<"In degree of "<<in<<" is "<<count;

}

void graph::out\_degree(int out)

{

int count=0;

for(int i=0;i<vertices;++i)

if(AM[i][out]==1)

++count;

cout<<"In degree of "<<out<<" is "<<count;

}

void graph::get\_neighbours(int \*arr,int vert)

{

int j=0;

for(int i=0;i<vertices;++i)

if(AM[vert][i]==1)

{

arr[j]=i;

++j;

}

}

void graph::depth\_first\_traversal()

{

cout<<"Enter source vertex: ";

int vert;

cin>>vert;

stack<int>s;

int track[vertices]={0};

s.push(vert);

cout<<"Depth first traversal is:"<<endl;

while(!s.empty())

{

vert=s.top();

if(track[vert]==0)

{

cout<<vert<<" ";

track[vert]=1;

}

int arr[vertices];

for(int i=0;i<vertices;++i)

arr[i]=-1;

get\_neighbours(arr,vert);

int i;

for(i=0;track[arr[i]]==1;++i);

int min\_neighbor=arr[i];

if(min\_neighbor==-1)

s.pop();

else

s.push(min\_neighbor);

}

}

void graph::depth\_first\_traversal1()

{

cout<<"Enter source vertex: ";

int vert;

cin>>vert;

stack<int>s;

int track[vertices]={0};

s.push(vert);

track[vert]=1;

cout<<"Depth first traversal is:"<<endl;

cout<<vert<<" ";

while(!s.empty())

{

int flag=0;

int store=s.top();

for(int i=0;i<vertices;++i)

{

if(track[i]==0)

{

if(AM[store][i]==1)

{

flag=1;

track[i]=1;

s.push(i);

cout<<i<<" ";

break;

}

}

}

if(flag==0)

s.pop();

}

}

void graph::breadth\_first\_traversal()

{

cout<<"Enter source vertex: ";

int vert;

cin>>vert;

queue<int>q;

int track[vertices]={0};

q.push(vert);

track[vert]=1;

cout<<"Breadth first traversal is:"<<endl;

while(!q.empty())

{

int store=q.front();

q.pop();

cout<<store<<" ";

for(int i=0;i<vertices;++i)

{

if(track[i]==0)

{

if(AM[store][i]==1)

{

track[i]=1;

q.push(i);

}

}

}

}

}

void graph::display\_graph()

{

cout<<"Graph:"<<endl;

for(int i=0;i<vertices;++i)

{

for(int j=0;j<vertices;++j)

cout<<AM[i][j]<<" ";

cout<<endl;

}

cout<<endl;

}

int main()

{

cout<<"Enter vertices of graph: ";

int vert;

cin>>vert;

graph obj(vert);

int ch;

do

{

cout<<"\nEnter\n1 to create graph\n2 for depth first traversal\n3 for breadth first traversal\n4 to calculate in degree\n5 to calculate out degree\n6 to display graph\n0 to exit:";

cin>>ch;

switch(ch)

{

case 1:

obj.create\_graph();

cout<<"Graph created"<<endl;

break;

case 2:

obj.depth\_first\_traversal1();

break;

case 3:

obj.breadth\_first\_traversal();

break;

case 4:

cout<<"Enter element whose in degree you want to calculate: ";

int in;

cin>>in;

obj.in\_degree(in);

break;

case 5:

cout<<"Enter element whose in degree you want to calculate: ";

int out;

cin>>out;

obj.out\_degree(out);

break;

case 6:

obj.display\_graph();

break;

case 0:

ch=0;

break;

default:

cout<<"Invalid input,try aagin"<<endl;

}

}

while(ch!=0);

return 0;

}