**/\*Project- Binary Search Tree implement all opertions (insertion, searching, deletion, inorder, preorder, postorder, traversing, inorder predecessor and successor, maximum and minimum value in BST. \*/**

# include <iostream>

# include <cstdlib>

using namespace std;

struct node //node declaration

{

int info;

struct node \*left;

struct node \*right;

}\*r;

class BST

{

public://functions declaration

void search(node \*, int);

void find(int, node \*\*, node \*\*);

void insert(node \*, node \*);

void del(int);

void casea(node \*,node \*);

void caseb(node \*,node \*);

void casec(node \*,node \*);

void preorder(node \*);

void inorder(node \*);

void postorder(node \*);

void show(node \*, int);

void findPreSuc(node\*, node\*&, node\*& , int);

int maxValue(node \*);

int minValue(node \*);

BST()

{

r = NULL;

}

};

void BST::find(int i, node \*\*par, node \*\*loc)//find the position of the item

{

node \*ptr, \*ptrsave;

if (r == NULL)

{

\*loc = NULL;

\*par = NULL;

return;

}

if (i == r->info)

{

\*loc = r;

\*par = NULL;

return;

}

if (i < r->info)

ptr = r->left;

else

ptr = r->right;

ptrsave = r;

while (ptr != NULL)

{

if (i == ptr->info)

{

\*loc = ptr;

\*par = ptrsave;

return;

}

ptrsave = ptr;

if (i < ptr->info)

ptr = ptr->left;

else

ptr = ptr->right;

}

\*loc = NULL;

\*par = ptrsave;

}

void BST::search(node \*root, int data) //searching

{

int depth = 0;

node \*temp = new node;

temp = root;

while(temp != NULL)

{

depth++;

if(temp->info == data)

{

cout<<"\nData found at depth: "<<depth<<endl;

return;

}

else if(temp->info > data)

temp = temp->left;

else

temp = temp->right;

}

cout<<"\n Data not found"<<endl;

return;

}

void BST::insert(node \*tree, node \*newnode)

{

if (r == NULL)

{

r = new node;

r->info = newnode->info;

r->left= NULL;

r->right= NULL;

cout<<"Root Node is Added"<<endl;

return;

}

if (tree->info == newnode->info)

{

cout<<"Element already in the tree"<<endl;

return;

}

if (tree->info > newnode->info)

{

if (tree->left != NULL)

{

insert(tree->left, newnode);

}

else

{

tree->left= newnode;

(tree->left)->left = NULL;

(tree->left)->right= NULL;

cout<<"Node Added To Left"<<endl;

return;

}

}

else

{

if (tree->right != NULL)

{

insert(tree->right, newnode);

}

else

{

tree->right = newnode;

(tree->right)->left= NULL;

(tree->right)->right = NULL;

cout<<"Node Added To Right"<<endl;

return;

}

}

}

void BST::del(int i)

{

node \*par, \*loc;

if (r == NULL)

{

cout<<"Tree empty"<<endl;

return;

}

find(i, &par, &loc);

if (loc == NULL)

{

cout<<"Item not present in tree"<<endl;

return;

}

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

{

casea(par, loc);

cout<<"item deleted"<<endl;

}

if (loc->left!= NULL && loc->right == NULL)

{

caseb(par, loc);

cout<<"item deleted"<<endl;

}

if (loc->left== NULL && loc->right != NULL)

{

caseb(par, loc);

cout<<"item deleted"<<endl;

}

if (loc->left != NULL && loc->right != NULL)

{

casec(par, loc);

cout<<"item deleted"<<endl;

}

free(loc);

}

void BST::casea(node \*par, node \*loc )

{

if (par == NULL)

{

r= NULL;

}

else

{

if (loc == par->left)

par->left = NULL;

else

par->right = NULL;

}

}

void BST::caseb(node \*par, node \*loc)

{

node \*child;

if (loc->left!= NULL)

child = loc->left;

else

child = loc->right;

if (par == NULL)

{

r = child;

}

else

{

if (loc == par->left)

par->left = child;

else

par->right = child;

}

}

void BST::casec(node \*par, node \*loc)

{

node \*ptr, \*ptrsave, \*suc, \*parsuc;

ptrsave = loc;

ptr = loc->right;

while (ptr->left!= NULL)

{

ptrsave = ptr;

ptr = ptr->left;

}

suc = ptr;

parsuc = ptrsave;

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

casea(parsuc, suc);

else

caseb(parsuc, suc);

if (par == NULL)

{

r = suc;

}

else

{

if (loc == par->left)

par->left = suc;

else

par->right= suc;

}

suc->left = loc->left;

suc->right= loc->right;

}

void BST::preorder(node \*ptr)

{

if (r == NULL)

{

cout<<"Tree is empty"<<endl;

return;

}

if (ptr != NULL)

{

cout<<ptr->info<<" ";

preorder(ptr->left);

preorder(ptr->right);

}

}

void BST::inorder(node \*ptr)//inorder traversal

{

if (r == NULL)

{

cout<<"Tree is empty"<<endl;

return;

}

if (ptr != NULL)

{

inorder(ptr->left);

cout<<ptr->info<<" ";

inorder(ptr->right);

}

}

void BST::postorder(node \*ptr)//postorder traversal

{

if (r == NULL)

{

cout<<"Tree is empty"<<endl;

return;

}

if (ptr != NULL)

{

postorder(ptr->left);

postorder(ptr->right);

cout<<ptr->info<<" ";

}

}

void BST::show(node \*ptr, int level)//print the tree

{

int i;

if (ptr != NULL)

{

show(ptr->right, level+1);

cout<<endl;

if (ptr == r)

cout<<"Root->: ";

else

{

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

cout<<" ";

}

cout<<ptr->info;

show(ptr->left, level+1);

}

}

void BST::findPreSuc(node\* r, node\*& pre, node\*& suc, int key) //find predecessor and successor

{

if (r == NULL)

return;

// Search for given key in BST.

while (r != NULL) {

// If root is given key.

if (r->info == key) {

// the minimum value in right subtree

// is successor.

if (r->right) {

suc = r->right;

while (suc->left)

suc = suc->left;

}

// the maximum value in left subtree

// is predecessor.

if (r->left) {

pre = r->left;

while (pre->right)

pre = pre->right;

}

return;

}

/\* If key is greater than root, then key lies in right subtree. Root could be predecessor if left subtree of key is null.\*/

else if (r->info < key) {

pre = r;

r = r->right;

}

// If key is smaller than root, then key lies in left subtree. Root could be successor if right subtree of key is null. \*/

else {

suc = r;

r = r->left;

}

}

}

int BST::maxValue(node \*node)

{

/\* loop down to find the rightmost leaf \*/

struct node\* current = node;

while (current->right != NULL)

current = current->right;

return (current->info);

}

int BST::minValue(node \*node)

{

/\* loop down to find the rightmost leaf \*/

struct node\* current2 = node;

while (current2->left != NULL)

current2 = current2->left;

return (current2->info);

}

int main()

{

int c, n,item;

BST bst;

node \*t;

node \*pre = NULL, \*suc = NULL;

int key;

while (1)

{

cout<<"-----------------------------------------------------------";

cout<<"\n1.Insert Element "<<endl;

cout<<"2.Delete Element "<<endl;

cout<<"3.Search Element"<<endl;

cout<<"4.Inorder Traversal"<<endl;

cout<<"5.Preorder Traversal"<<endl;

cout<<"6.Postorder Traversal"<<endl;

cout<<"7.Display the tree"<<endl;

cout<<"8.Find in inorder predecessor and successor "<<endl;

cout<<"9.Maximum and Minimum value in BST:"<<endl;

cout<<"10.Quit"<<endl;

cout<<"Enter your choice : ";

cin>>c;

switch(c)

{

case 1:

t = new node;

cout<<"Enter the number to be inserted : ";

cin>>t->info;

bst.insert(r, t);

break;

case 2:

if (r == NULL)

{

cout<<"Tree is empty, nothing to delete"<<endl;

continue;

}

cout<<"Enter the number to be deleted : ";

cin>>n;

bst.del(n);

break;

case 3:

cout<<"Search:"<<endl;

cin>>item;

bst.search(r,item);

break;

case 4:

cout<<"Inorder Traversal of BST:"<<endl;

bst.inorder(r);

cout<<endl;

break;

case 5:

cout<<"Preorder Traversal of BST:"<<endl;

bst.preorder(r);

cout<<endl;

break;

case 6:

cout<<"Postorder Traversal of BST:"<<endl;

bst.postorder(r);

cout<<endl;

break;

case 7:

cout<<"Display BST:"<<endl;

bst.show(r,1);

cout<<endl;

break;

case 8:

cout<<"Enter Key:";

cin>>key;

bst.findPreSuc(r, pre, suc, key);

if (pre != NULL)

cout << "Inorder Predecessor is " << pre->info << endl;

else

cout << "-1";

if (suc != NULL)

cout << "Inorder Successor is " << suc->info;

else

cout << "-1";

cout<<endl;

break;

case 9:

cout << "Maximum value in BST is " << bst.maxValue(r);

cout << "\nMinimum value in BST is " << bst.minValue(r);

cout<<endl;

break;

case 10:

exit(1);

default:

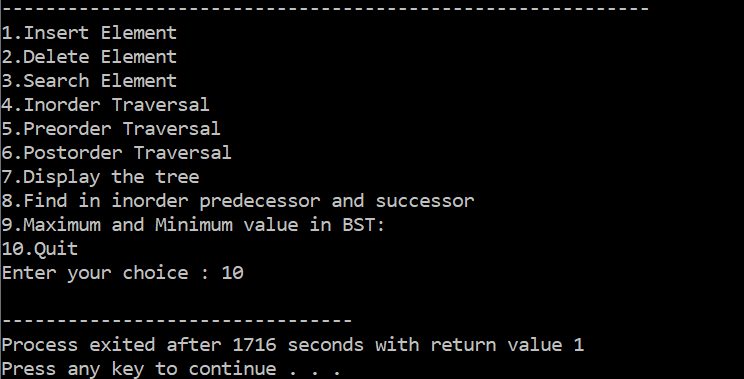
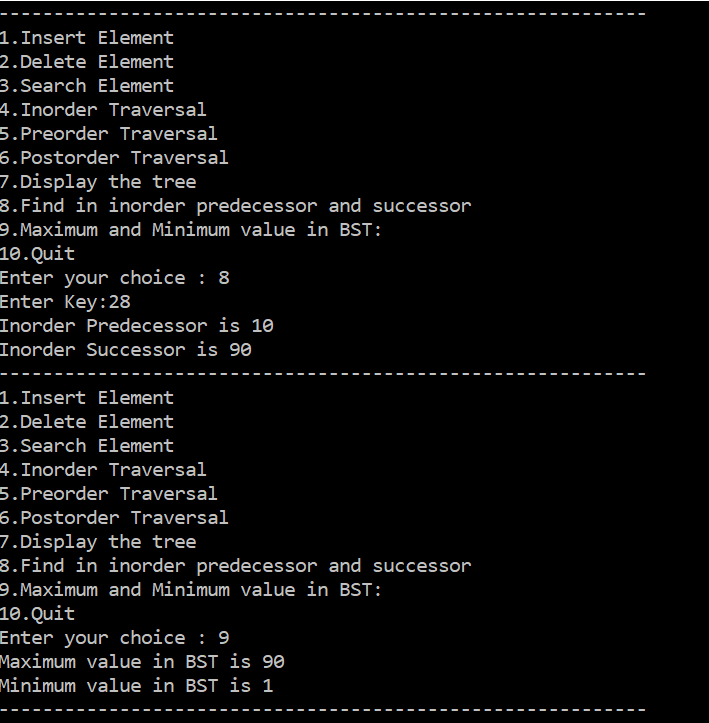
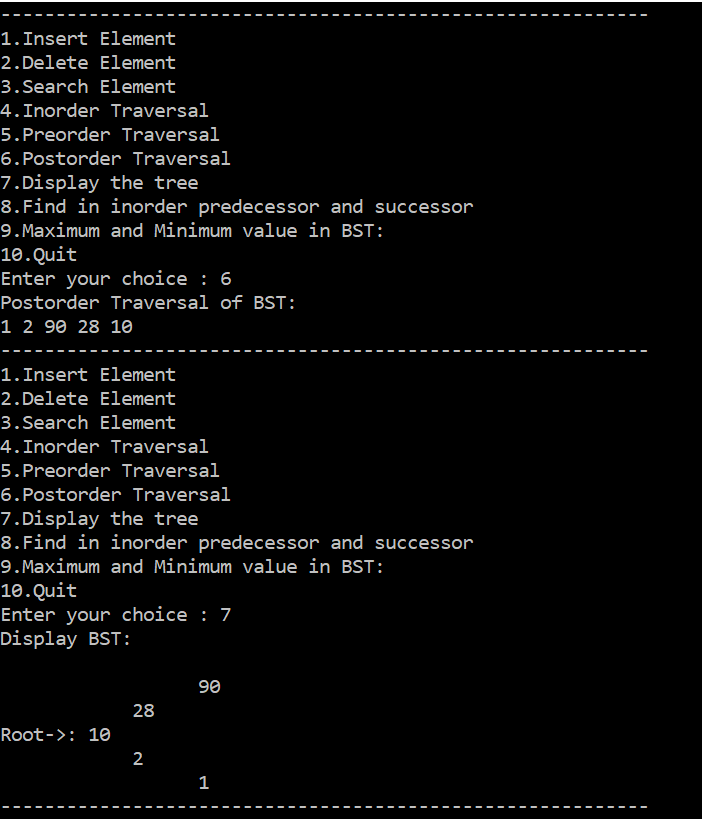
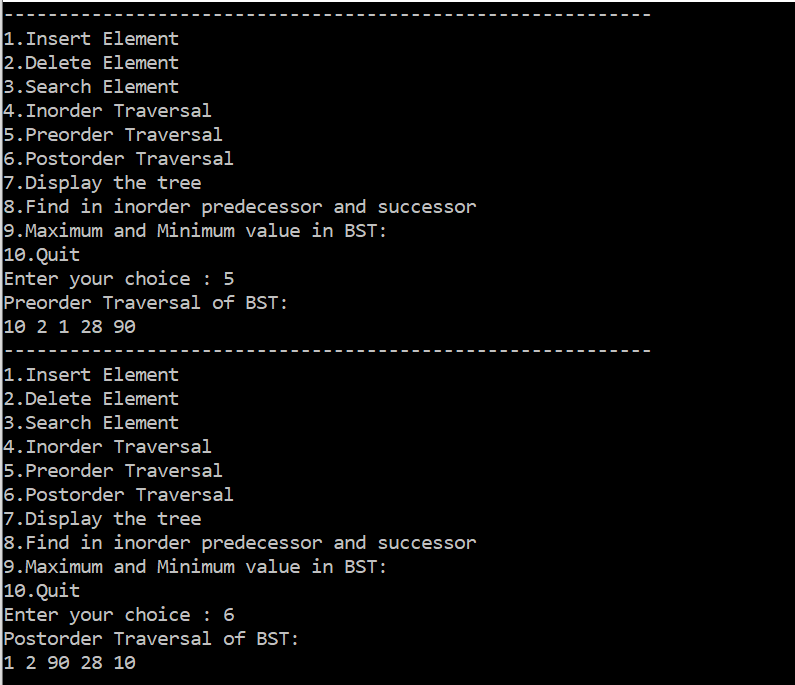
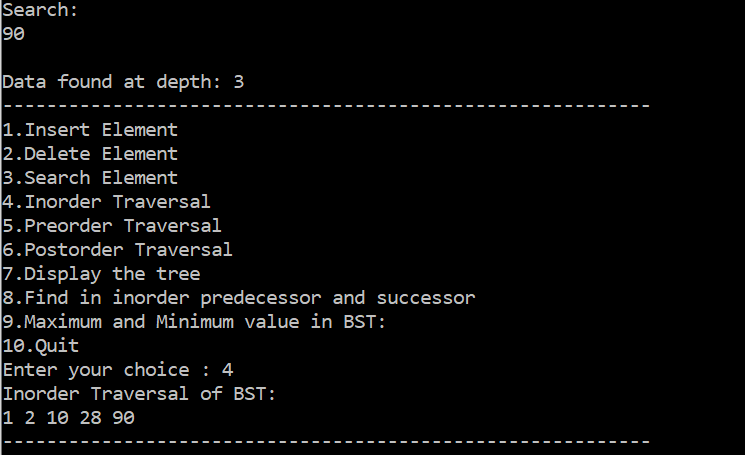
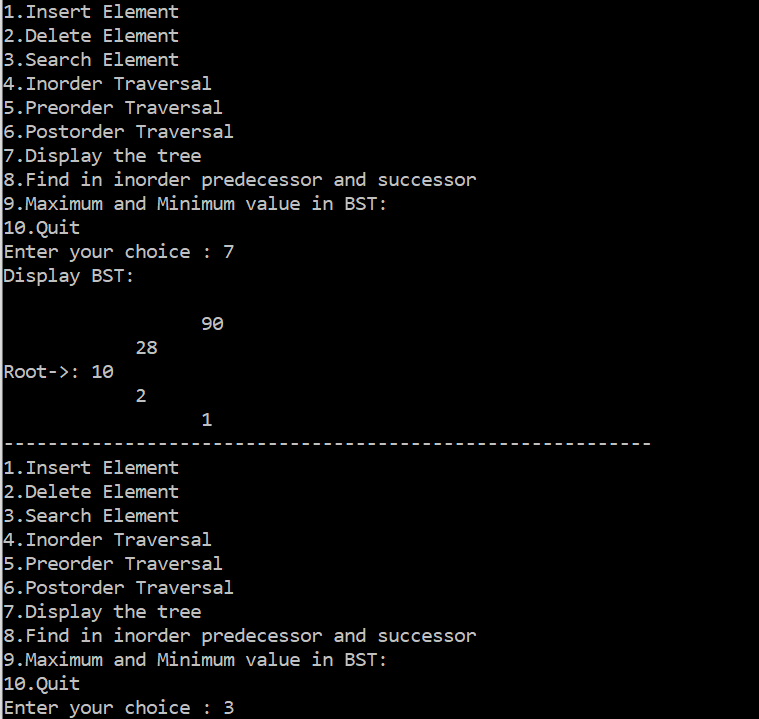
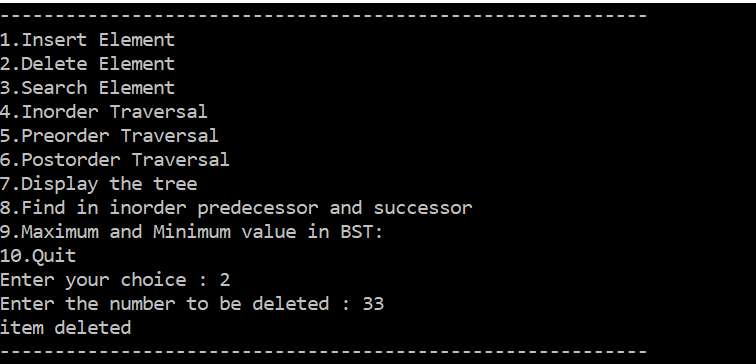
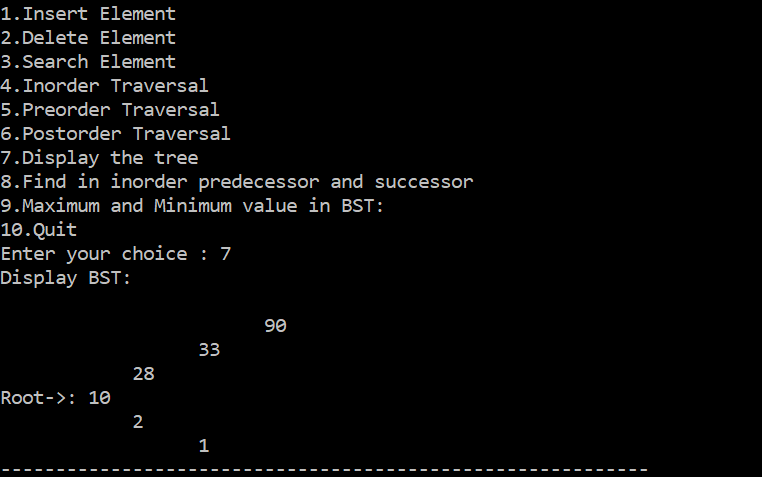
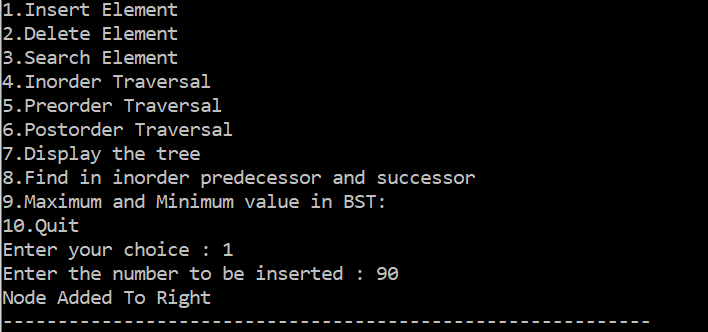
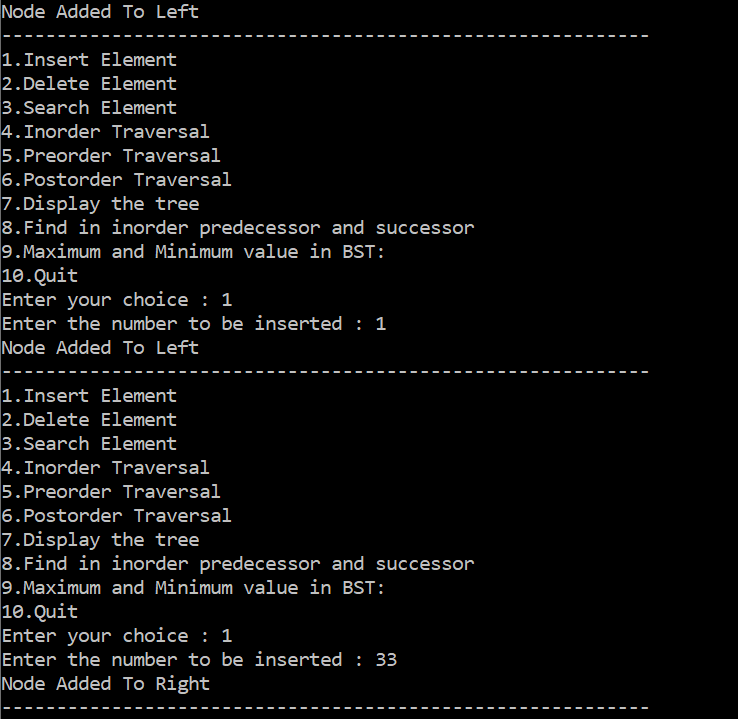
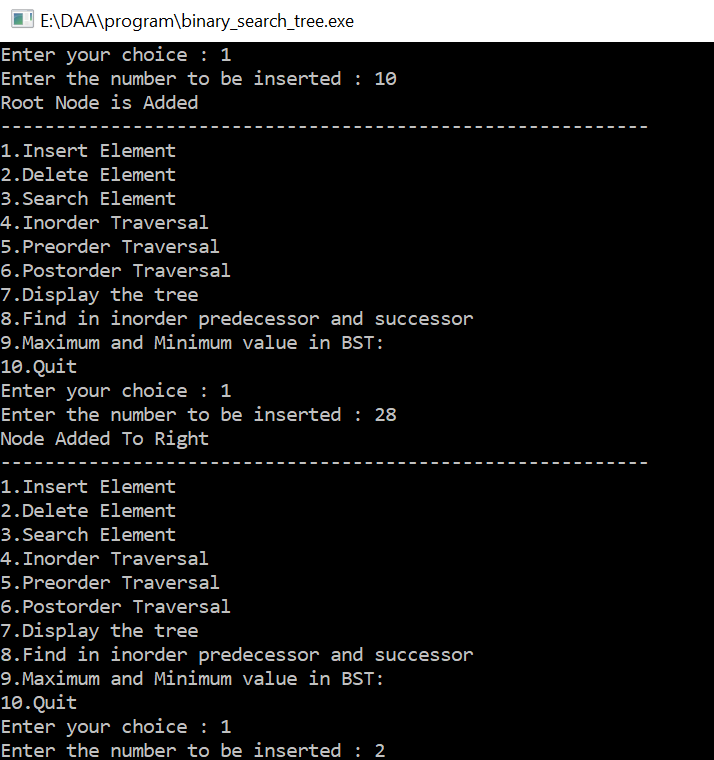
cout<<"Wrong choice"<<endl;

}

}

}

**OUTPUT**

****

**/\*Project- Breadth First Search(BFS) and Depth First Search(DFS)\*/**

#include<iostream>

#include<vector>

#include<ctime>

#include<stdlib.h>

#include<stack>

#include<queue>

using namespace std;

/\* run this program using the console pauser or add your own getch, system("pause") or input loop \*/

class RandomNumber

{

int low,high;

public:

RandomNumber(int h=10,int l=0):low(l),high(h)

{

srand(time(0));

}

int operator()(){

return rand()%(high-low)+low;

}

};

const int MAX=40;

class Graph

{

int nv;

vector <vector<int> > adjM;

vector<bool> visited;

public:

Graph(int v=10):nv(v),adjM(v),visited(v) {}

void genGraph()

{

RandomNumber r(2,0);

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

{

visited[i]=false;

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

{

adjM[i].push\_back(r());

}

}

}

void printGraph()

{

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

{

cout<<i<<" -> ";

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

{

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

}

cout<<"\n";

}

}

void DFS(int s=3)

{

stack<int> stk;

stk.push(s);

visited[s]=true;

cout<<"\nDFS sequence::.............\n";

while(!stk.empty())

{

int t=stk.top();

stk.pop();

cout<<t<<" ";

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

{

if(adjM[t][i])

{

if(!(visited[i]))

{

visited[i]=true;

stk.push(i);

}

}

}

}

}

void BFS(int s=3)

{

queue<int> q;

q.push(s);

visited[s]=true;

cout<<"\nBFS sequence::........\n";

while(!q.empty())

{

int s=q.front();

cout<<s<<" ";

q.pop();

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

{

if(adjM[s][i])

{

if(!(visited[i]))

{

visited[i]=true;

q.push(i);

}

}

}

}

}

};

int main()

{

Graph g(7);

//g.genGraph();

//g.printGraph();

int choice;

while(1)

{

cout<<"\nOPERATIONS:\n1.DFS\n2.BFS\n3.Exit\nENTER CHOICE:";

cin>>choice;

switch(choice)

{

case 1:

//Graph g(7);

g.genGraph();

g.printGraph();

g.DFS(4);

break;

case 2:

//Graph g(7);

g.genGraph();

g.printGraph();

g.BFS(5);

break;

case 3:

exit(1);

default:

cout<<"\nWrong option entered\n";

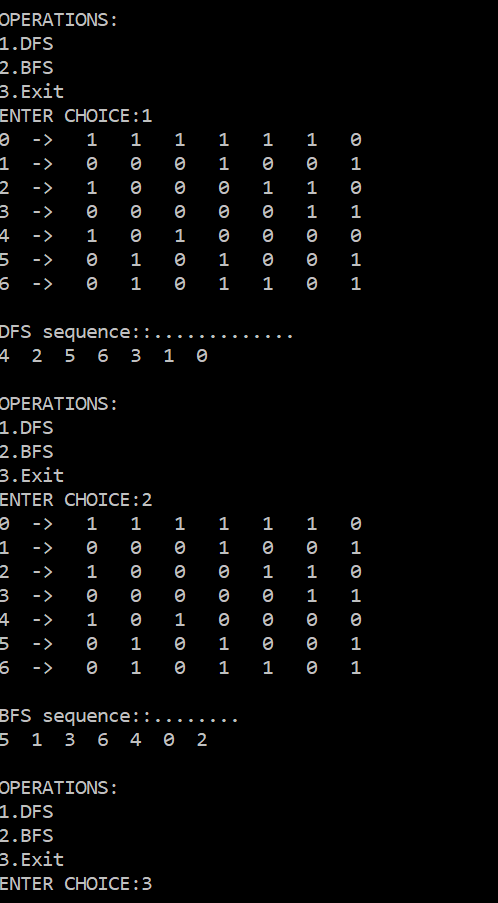
}

}

return 0;

}

**OUTPUT**

****