**DSA LAB – 10**

**Name:** Etcherla Sai Manoj **Mis. No:** 112015044 **Branch:** CSE

**Question 1:**

**Code:**

#include<bits/stdc++.h>

using namespace std;

struct Node{

int data;

Node\* left, \*right;

};

Node\* create\_new\_node(int data){

Node\* temp = new Node();

temp->data = data;

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

return temp;

}

Node\* insert\_node(Node\* ptr, int data){

if(ptr == NULL){

ptr = create\_new\_node(data);

return ptr;

}

else if(ptr->data >= data) ptr->left = insert\_node(ptr->left,data);

else ptr->right = insert\_node(ptr->right,data);

return ptr;

}

void search(Node\* ptr, int data){

if(ptr == NULL){

cout << "The element is not present in the Tree\n";

return;

}

if(ptr->data == data){

cout << "The element is present in the Tree\n";

return;

}

if(ptr->data > data){

search(ptr->left, data);

}

else{

search(ptr->right, data);

}

}

void depthfirst\_display(Node\* ptr){

if(ptr == NULL) return;

depthfirst\_display(ptr->left);

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

depthfirst\_display(ptr->right);

}

void breadthfirst\_display(Node\* ptr){

if(ptr == NULL){

cout << "Tree is Empty\n";

return;

}

queue<Node\*> q1;

q1.push(ptr);

while(!q1.empty()){

Node\* temp = q1.front();

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

if(temp->left != NULL){

q1.push(temp->left);

}

if(temp->right != NULL){

q1.push(temp->right);

}

q1.pop();

}

}

int minimum(Node\* ptr){

if(ptr == NULL) return -1;

while(ptr->left!=NULL){

ptr = ptr->left;

}

return ptr->data;

}

Node\* deletenode(Node\* ptr,int data){

if(ptr == NULL) return ptr;

if(ptr->data > data){

ptr->left = deletenode(ptr->left, data);

}

else if(ptr->data < data){

ptr->right = deletenode(ptr->right, data);

}

else{

if(ptr->left == NULL and ptr->right == NULL){

delete ptr;

ptr = NULL;

return ptr;

}

else if(ptr->right == NULL){

Node\* temp = ptr;

ptr = ptr->left;

delete temp;

return ptr;

}

else if(ptr->left == NULL){

Node\* temp = ptr;

ptr = ptr->right;

delete temp;

return ptr;

}

else{

int right\_minimum = minimum(ptr->right);

ptr->data = right\_minimum;

ptr->right = deletenode(ptr->right, right\_minimum);

}

}

return ptr;

}

int depth(Node\* ptr){

if(ptr == NULL) return -1;

return max(depth(ptr->right), depth(ptr->left)) + 1;

}

void mirror(Node\* ptr){

if (ptr == NULL) return;

else{

struct Node\* temp;

mirror(ptr->left);

mirror(ptr->right);

temp = ptr->left;

ptr->left = ptr->right;

ptr->right = temp;

}

}

int main(){

Node\* head = NULL;

int test\_node, insert, choice;

cout << "=================MENU=================\n";

cout << "1.Insert elements\n";

cout << "2.Delete a node\n";

cout << "3.Depth of tree\n";

cout << "4.search a node\n";

cout << "5.Display original tree\n";

cout << "6.Mirror image of tree\n";

cout << "7.Mirorr image of tree level-wise\n";

cout << "8.Exit\n";

cout << "======================================\n";

while(1){

cout << "\nEnter your choice : ";

cin >> choice;

switch(choice){

case 1:

cout << "Enter total number of elements in binary search tree : ";

cin >> test\_node;

cout << "\nEnter elements of binary search tree : ";

//inserting nodes to tree

for(int j=0; j < test\_node; j++){

cin >> insert;

head = insert\_node(head, insert);

}

depthfirst\_display(head);

cout << "\n";

break;

case 2:

cout<<"Enter element to be deleted : ";

cin >> test\_node;

head = deletenode(head, test\_node);

depthfirst\_display(head);

cout << "\n";

break;

case 3:

cout << "Depth of the tree : " << depth(head);

cout << "\n";

break;

case 4:

cout << "Enter element to be searched : ";

cin >> test\_node;

search(head, test\_node);

break;

case 5:

cout<<"Breadth-first Search of the tree(Display) : ";

breadthfirst\_display(head);

cout << "\n";

break;

case 6:

cout << "Mirror image of tree : ";

mirror(head);

depthfirst\_display(head);

cout << "\n";

break;

case 7:

cout << "Mirror image of tree level wise : ";

mirror(head);

breadthfirst\_display(head);

cout << "\n";

break;

case 8:

return 0;

default:

cout << "Enter a valid choice!!!\n";

break;

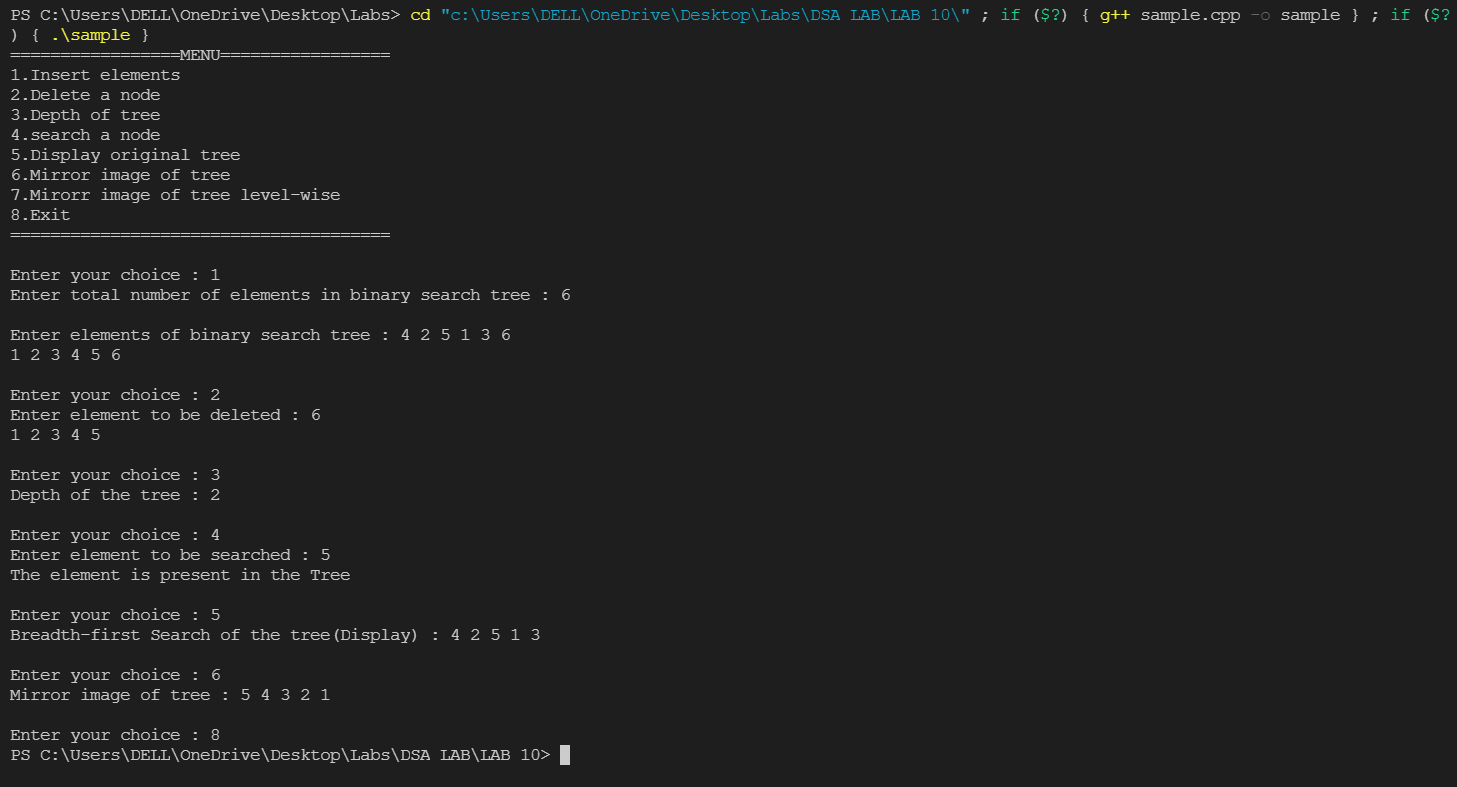
}

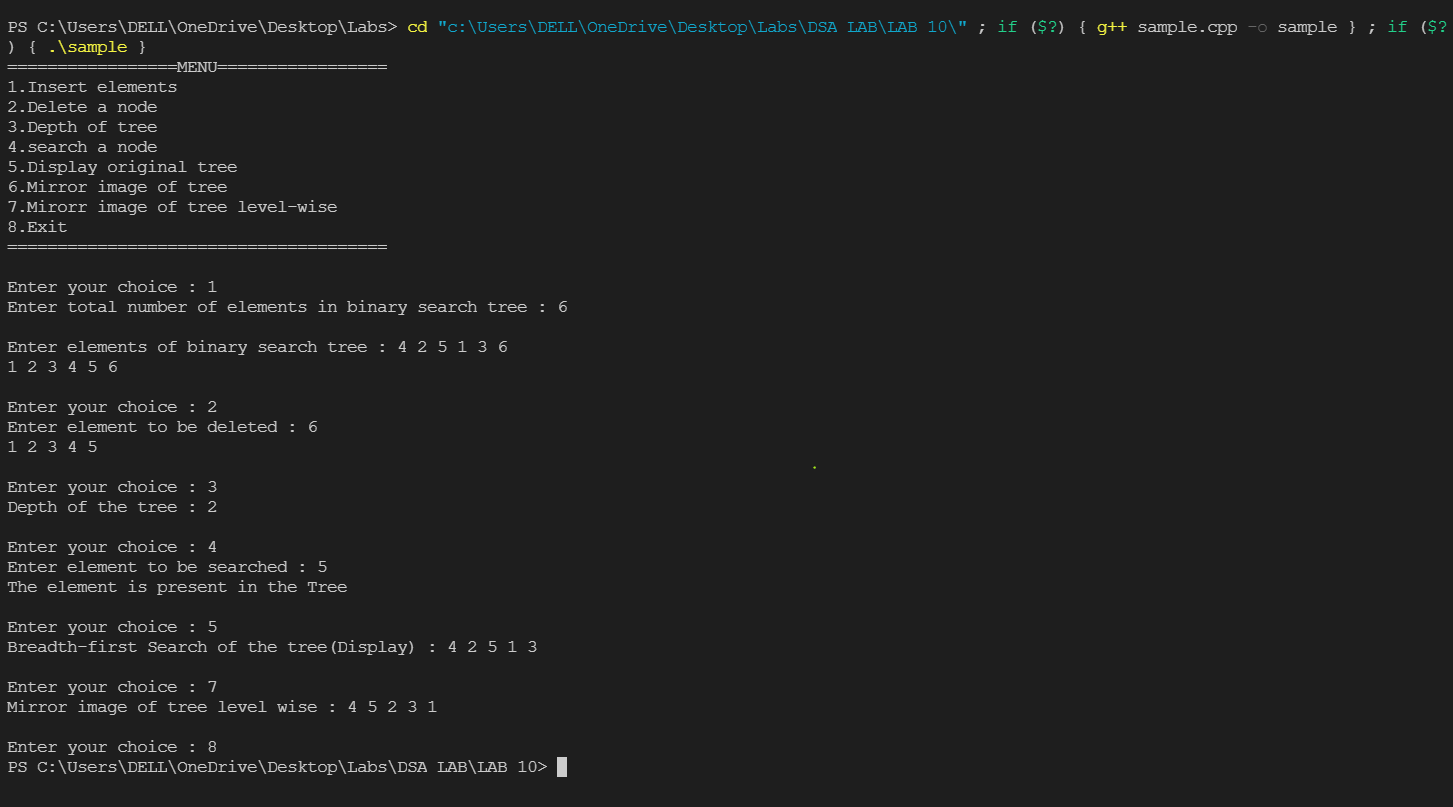
}

return 0;

}

**Input & Output:**

****

****