**M. Ali. Arslan**

**19F-0348**

**Lab-05**

**Task # 01**

#include <iostream>

using namespace std;

struct Node

{

int data;

Node\* left, \* right;

};

class Tree

{

private:

Node\* root;

public:

Tree()

{

root = NULL;

}

bool isTreeEmpty() {

if (root == NULL) {

return true;

}

else {

return false;

}

}

Node\* getRoot()

{

return this->root;

}

Node\* insert(Node\* node, int key)

{

Node\* newNode = new Node;

newNode->left = NULL;

newNode->right = NULL;

newNode->data = key;

if (this->root == NULL)

{

this->root = newNode;

}

if (node == NULL)

{

node = newNode;

}

if (key < node->data)

node->left = insert(node->left, key);

else if (key > node->data)

node->right = insert(node->right, key);

return node;

}

bool Search(Node\* node, int key)

{

static int count = 0;

while (node != NULL)

{

if (node->data == key)

{

count++;

cout << "Found in " << count << " iterations." << endl;

return true;

}

if (node->data < key)

{

count++;

return Search(node->right, key);

}

count++;

return Search(root->left, key);

}

cout << "Not found after " << count << " iterations." << endl;

return false;

}

Node\* deleteNode(Node\* node, int key)

{

if (node == NULL)

{

return node;

}

if (key < node->data)

{

node->left = deleteNode(node->left, key);

}

else if (key > node->data)

{

node->right = deleteNode(node->right, key);

}

else

{

if (node->left == NULL)

{

Node\* temp = node->right;

delete node;

return temp;

}

else if (node->right == NULL)

{

Node\* temp = node->left;

delete node;

return temp;

}

Node\* temp = minimum(node->right);

node->data = temp->data;

node->right = deleteNode(node->right, temp->data);

}

return node;

}

Node\* minimum(Node\* temp)

{

if (temp == NULL)

{

return NULL;

}

while (temp->left != NULL)

{

temp = temp->left;

}

return temp;

}

};

int main()

{

Tree T;

int data;

int choice;

while (1)

{

cout << "1. Insert Node" << endl;

cout << "2. Search Node" << endl;

cout << "3. Delete Node" << endl;

cout << "4. IsEmpty" << endl;

cout << "5. Exit" << endl;

cout << " Option: ";

cin >> choice;

switch (choice)

{

case 1:

cout << "\nEnter the data:\n";

cin >> data;

T.insert(T.getRoot(), data);

break;

case 2:

cout << "\nEnter the data to search:\n";

cin >> data;

T.Search(T.getRoot(), data);

break;

case 3:

cout << "\nEnter the data to delete:\n";

cin >> data;

T.deleteNode(T.getRoot(), data);

break;

case 4:

if (T.isTreeEmpty() == 1)

{

cout << "Tree is Empty!" << endl;

}

else if (T.isTreeEmpty() == 0)

{

cout << "Tree is not empty!" << endl;

}

break;

case 5:

exit(0);

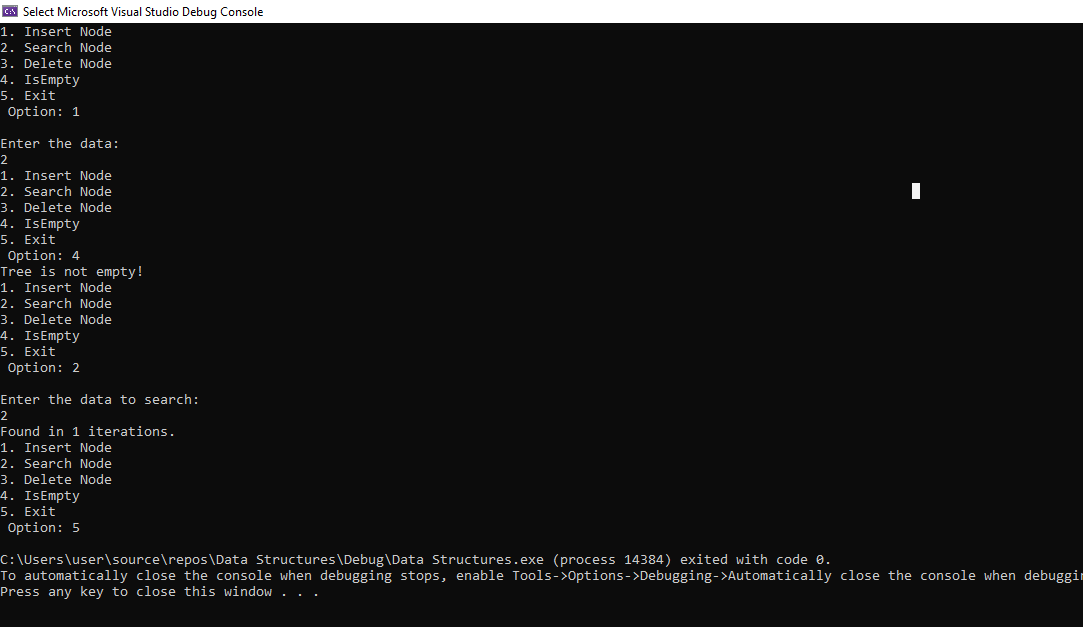
break;

}

}

return 0;

}



**Task # 02**

#include<iostream>

using namespace std;

class TreeNode {

public:

int value;

TreeNode\* left;

TreeNode\* right;

TreeNode() {

value = 0;

left = NULL;

right = NULL;

}

TreeNode(int v) {

value = v;

left = NULL;

right = NULL;

}

};

class BST {

public:

TreeNode\* root;

BST() {

root = NULL;

}

bool isTreeEmpty() {

if (root == NULL) {

return true;

}

else {

return false;

}

}

void insertNode(TreeNode\* new\_node) {

if (root == NULL) {

root = new\_node;

cout << "Value Inserted as root node!" << endl;

}

else {

TreeNode\* temp = root;

while (temp != NULL) {

if (new\_node->value == temp->value) {

cout << "Value Already exist," <<

"Insert another value!" << endl;

return;

}

else if ((new\_node->value < temp->value) && (temp->left == NULL)) {

temp->left = new\_node;

cout << "Value Inserted to the left!" << endl;

break;

}

else if (new\_node->value < temp->value) {

temp = temp->left;

}

else if ((new\_node->value > temp->value) && (temp->right == NULL)) {

temp->right = new\_node;

cout << "Value Inserted to the right!" << endl;

break;

}

else {

temp = temp->right;

}

}

}

}

TreeNode\* insertRecursively(TreeNode\* r, TreeNode\* new\_node)

{

if (r == NULL)

{

r = new\_node;

cout << "Insertion successful" << endl;

return r;

}

if (new\_node->value < r->value)

{

r->left = insertRecursively(r->left, new\_node);

}

else if (new\_node->value > r->value)

{

r->right = insertRecursively(r->right, new\_node);

}

else

{

cout << "No duplicate values allowed!" << endl;

return r;

}

return r;

}

TreeNode\* search(int v) {

if (root == NULL) {

return root;

}

else {

TreeNode\* temp = root;

while (temp != NULL) {

if (v == temp->value) {

return temp;

}

else if (v < temp->value) {

temp = temp->left;

}

else {

temp = temp->right;

}

}

return NULL;

}

}

TreeNode\* recursivelySearch(TreeNode\* r, int val) {

if (r == NULL || r->value == val)

return r;

else if (val < r->value)

return recursivelySearch(r->left, val);

else

return recursivelySearch(r->right, val);

}

TreeNode\* minValueNode(TreeNode\* node) {

TreeNode\* current = node;

while (current->left != NULL) {

current = current->left;

}

return current;

}

TreeNode\* deleteNode(TreeNode\* r, int v) {

if (r == NULL) {

return NULL;

}

else if (v < r->value) {

r->left = deleteNode(r->left, v);

}

else if (v > r->value) {

r->right = deleteNode(r->right, v);

}

else {

if (r->left == NULL) {

TreeNode\* temp = r->right;

delete r;

return temp;

}

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

TreeNode\* temp = r->left;

delete r;

return temp;

}

else {

TreeNode\* temp = minValueNode(r->right);

r->value = temp->value;

r->right = deleteNode(r->right, temp->value);

}

}

return r;

}

};

int main() {

BST obj;

int option, val;

do {

cout << "1. Insert Node" << endl;

cout << "2. Search Node" << endl;

cout << "3. Delete Node" << endl;

cout << "4. IsEmpty" << endl;

cout << "5. Exit" << endl;

cout << " Option: ";

cin >> option;

TreeNode\* new\_node = new TreeNode();

switch (option) {

case 0:

break;

case 1:

cout << "Enter VALUE of NODE: ";

cin >> val;

new\_node->value = val;

obj.root = obj.insertRecursively(obj.root, new\_node);

cout << endl;

break;

case 2:

cout << "Enter VALUE of NODE: ";

cin >> val;

new\_node = obj.recursivelySearch(obj.root, val);

if (new\_node != NULL) {

cout << "Value found!" << endl;

}

else {

cout << "Value NOT found!" << endl;

}

break;

case 3:

cout << "DELETE" << endl;

cout << "Enter VALUE of NODE: ";

cin >> val;

new\_node = obj.search(val);

if (new\_node != NULL) {

obj.deleteNode(obj.root, val);

cout << "Value Deleted!" << endl;

}

else {

cout << "Value NOT found!" << endl;

}

break;

case 4:

if (obj.isTreeEmpty() == 1)

{

cout << "Tree is Empty!" << endl;

}

else if(obj.isTreeEmpty()==0)

{

cout << "Tree is not empty!" << endl;

}

break;

case 5:

exit(0);

default:

cout << "Enter Proper Option number " << endl;

}

} while (option != 0);

return 0;

}