Assignment #3

#include <iostream>

using namespace std;

#define SPACE 10

class Node

{

public:

int data;

Node\* left;

Node\* right;

Node(int value)

{

data = value;

left = nullptr;

right = nullptr;

}

};

class BST

{

private:

Node\* root;

public:

BST()

{

root = nullptr;

}

void insert(int key)

{

root = insertbst(root, key);

}

// Recursive helper function for insert operation

Node\* insertbst(Node\* root, int val)

{

if (root == nullptr)

{

return new Node(val);

}

if (val < root->data) {

root->left = insertbst(root->left, val);

}

else if (val > root->data) {

root->right = insertbst(root->right, val);

}

return root;

}

// Function to delete a key from the BST

void remove(int key)

{

root = deletebst(root, key);

}

// Recursive helper function for delete operation

Node\* deletebst(Node\* root, int key)

{

if (root == nullptr)

{

return root;

}

if (key < root->data)

{

root->left = deletebst(root->left, key);

}

else if (key > root->data) {

root->right = deletebst(root->right, key);

}

else

{

if (root->left == nullptr)

{

Node\* temp = root->right;

delete root;

return temp;

}

else if (root->right == nullptr)

{

Node\* temp = root->left;

delete root;

return temp;

}

Node\* minValueNode = root->right;

while (minValueNode->left != nullptr) {

minValueNode = minValueNode->left;

}

root->data = minValueNode->data;

root->right = deletebst(root->right, minValueNode->data);

}

return root; // This will return the root

}

bool search(int key) // This function will search the value if exsist

{

return searchRecursive(root, key);

}

bool searchRecursive(Node\* root, int key)

{

if (root == nullptr)

{

return false;

}

if (key == root->data) {

return true;

}

else if (key < root->data) {

return searchRecursive(root->left, key);

}

else

{

return searchRecursive(root->right, key);

}

}

Node\* findmax(Node\* root)

{

Node\*temp = root;

while (temp->right!=NULL)

{

temp = temp->right;

}

return temp;

}

Node\* findmin(Node\* root)

{

Node\* temp = root;

while (temp->left!=NULL)

{

temp = temp->left;

}

return temp;

}

int Heightoftree(Node\* root)

{

if (root==NULL)

{

return -1;

}

int leftTree = Heightoftree(root->left);

int righttree = Heightoftree(root->right);

if (leftTree>righttree)

{

return leftTree + 1;

}

else

{

return righttree + 1;

}

}

int treenodescount(Node\* root)

{

if (root==NULL)

{

return 0;

}

int leftnodes = treenodescount(root->left);

int rightnodes = treenodescount(root->right);

return leftnodes + rightnodes + 1;//we will also count the root

}

int treleafcount(Node\* root)

{

if (root==NULL)

{

return 0;

}

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

{

return 1;

}

int leafnodes = treleafcount(root->left) + treleafcount(root->right);

return leafnodes;

}

void display()

{

preorder(root);

inorder(root);

postorder(root);

cout << endl;

}

int printlevelofnode(Node\* root,int cl)

{

if (root=NULL)

{

return 0;

}

else

{

cout << endl << "level of " << root->data << "is = " << cl << endl;

printlevelofnode(root->left, cl + 1);

printlevelofnode(root->right, cl + 1);

}

}

// Recursive function for display operation

void inorder(Node\* root)

{

if (root != nullptr)

{

inorder(root->left);

cout << "The inorder data of the root is " << root->data << " " << endl;

inorder(root->right);

}

}

void preorder(Node\* root)

{

if (root!=NULL)

{

cout << "The preoder data of the root is : " << root->data << " " << endl;

preorder(root->left);

preorder(root->right);

}

}

void postorder(Node\* root)

{

if (root!=NULL)

{

postorder(root->left);

postorder(root->right);

cout << "The postorder data of the root is : " << root->data << endl;

}

}

};

void displayMenu()

{

cout << "Binary Search Tree operations " << endl;

cout << "1-> Insert a key" << endl;

cout << "2-> Delete a key" << endl;

cout << "3-> Search for a key" << endl;

cout << "4-> Display the BST" << endl;

cout<<"5->Find Max"<<endl;

cout<<"6->Find Min"<<endl;

cout<<"7-> Height of Tree"<<endl;

cout<<"8-> Tree Nodes"<<endl;

cout<<"9-> Tree leaf counts"<<endl;

cout << "10->Node level " << endl;

cout << "11. Exit" << endl;

cout << "Enter your choice: ";

}

int main()

{

BST bst;

int choice, key;

Node\* root = NULL;

bst.display();

do

{

displayMenu();

cin >> choice;

switch (choice)

{

case 1:

cout << "Enter the value to insert: ";

cin >> key;

bst.insert(key);

cout << "value inserted." << endl;

break;

case 2:

cout << "Enter the key to delete: ";

cin >> key;

bst.remove(key);

cout << "value deleted." << endl;

break;

case 3:

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

cin >> key;

if (bst.search(key))

{

cout << "value found in the BST." << endl;

}

else

{

cout << "value not found in the BST." << endl;

}

break;

case 4:

cout << "BST elements";

bst.display();

break;

case 5:

cout << "The Max element is : "<<bst.findmax(root) << endl;

break;

case 6:

cout << "The Min element is : " << bst.findmin(root) << endl;

break;

case 7:

cout << "The Height of the Tree is : " << bst.Heightoftree(root) << endl;

break;

case 8:

cout << "The tree Nodes count = " << bst.treenodescount(root) << endl;

break;

case 9:

cout << "The Tree Leaf Counts = " << bst.treleafcount(root) << endl;

break;

case 10:

cout << "The Node Level is : " << bst.printlevelofnode(root,0)<<endl;

break;

case 11:

cout << "Exiting program " << endl;

break;

default:

cout << "Invalid choice. Please try again." << endl;

}

cout << endl;

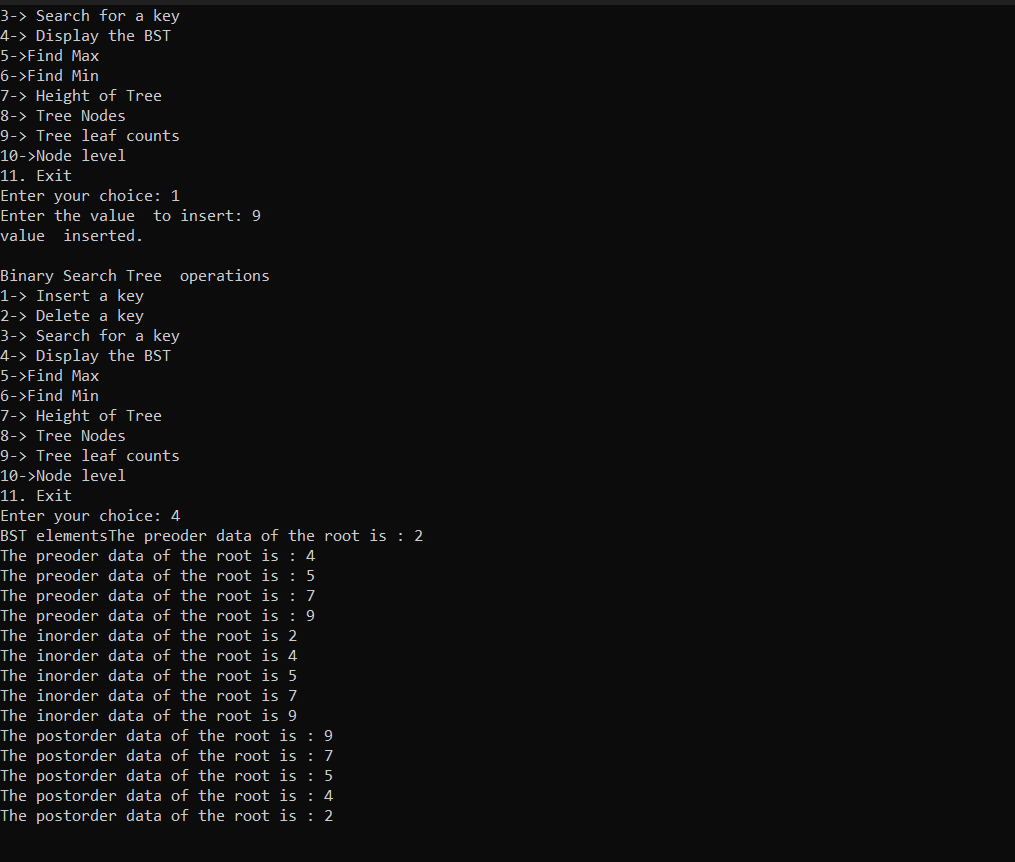
} while (choice != 11);

system("pause");

return 0;

}

**Output:**

****