Lab #10(QUIZ)

#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);

}

}

void display()

{

inorder(root);

cout << endl;

}

// Recursive function for display operation

void inorder(Node\* root)

{

if (root != nullptr)

{

inorder(root->left);

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

inorder(root->right);

}

}

};

void print2D(Node \*r, int space)

{

if (r == NULL) // Base case 1

return;

space += SPACE; // Increase distance between levels 2

print2D(r->right, space); // Process right child first 3

cout << endl;

for (int i = SPACE; i < space; i++) // 5

cout << " "; // 5.1

cout << r->data << "\n";

print2D(r->left, space); // Process left child 7

}

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. Exit" << endl;

cout << "Enter your choice: ";

}

int main()

{

BST bst;

int choice, key;

Node\*root = NULL;

print2D(root, 50);

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 << "Exiting program." << endl;

break;

default:

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

}

cout << endl;

} while (choice != 5);

print2D(root,50);

system("pause");

return 0;

}

Output:



