National

University

of

Computer

and

Emerging

Sciences

Chiniot-Faisalabad

Campus



**CL2001 – Data Structures - Lab**

**Lab 9**

**“BST”**

**Fall 2024**

**Maximum Marks:** 100 **Due Date:** 20 October 2024

**Submitted By**

|  |  |
| --- | --- |
| Name: | Talha Abdullah |
| Student ID: | 22F-3194 |
| Section: | BAI-3A |

**Submitted To**

Muhammad Yousaf

Lecturer – AI & Data Science Department

**Submission Date**

20 October 2024

# **Problem 1**

## **Source Code**

#include <iostream>

using namespace std;

struct Node {

int data;

Node\* left;

Node\* right;

Node(int value) {

data = value;

left = nullptr;

right = nullptr;

}

};

class BST {

public:

Node\* insert(Node\* root, int value) {

if (root == nullptr) {

return new Node(value);

}

if (value < root->data) {

root->left = insert(root->left, value);

} else if (value > root->data) {

root->right = insert(root->right, value);

}

return root;

}

Node\* find(Node\* root, int value) {

if (root == nullptr || root->data == value) {

return root;

}

if (value > root->data) {

return find(root->right, value);

}

return find(root->left, value);

}

Node\* findMin(Node\* root) {

while (root && root->left != nullptr) {

root = root->left;

}

return root;

}

Node\* deleteNode(Node\* root, int value) {

if (root == nullptr) {

return root;

}

if (value < root->data) {

root->left = deleteNode(root->left, value);

} else if (value > root->data) {

root->right = deleteNode(root->right, value);

} 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\* temp = findMin(root->right);

root->data = temp->data;

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

}

return root;

}

void inorder(Node\* root) {

if (root != nullptr) {

inorder(root->left);

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

inorder(root->right);

}

}

};

int main() {

BST bst;

Node\* root = nullptr;

Node\* foundNode = nullptr;

int choice, value;

while (true) {

cout << "----------Menu----------" << endl;

cout << "1. Insert in Tree" << endl;

cout << "2. Search in Tree" << endl;

cout << "3. Delete in Tree" << endl;

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

cout << "0. Exit the code" << endl;

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1:

cout << "Enter value to insert: ";

cin >> value;

root = bst.insert(root, value);

break;

case 2:

cout << "Enter value to search: ";

cin >> value;

foundNode = bst.find(root, value);

if (foundNode) {

cout << "Node " << value << " found in the tree." << endl;

} else {

cout << "Node " << value << " not found in the tree." << endl;

}

break;

case 3:

cout << "Enter value to delete: ";

cin >> value;

root = bst.deleteNode(root, value);

cout << "Node " << value << " deleted from the tree." << endl;

break;

case 4:

cout << "Inorder traversal of the tree: ";

bst.inorder(root);

cout << endl;

break;

case 0:

cout << "Exiting..." << endl;

return 0;

default:

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

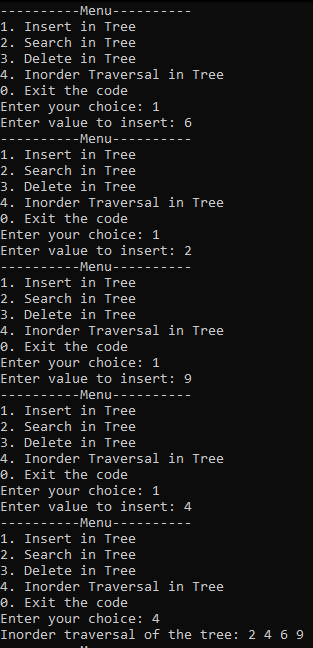
}

}

return 0;

}

## **Screenshot**



A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

# **Problem 2**

## **Source Code**

#include <iostream>

using namespace std;

struct Node {

int data;

Node\* left;

Node\* right;

Node(int value) {

data = value;

left = nullptr;

right = nullptr;

}

};

class BST {

public:

Node\* insert(Node\* root, int value) {

if (root == nullptr) {

return new Node(value);

}

if (value < root->data) {

root->left = insert(root->left, value);

}

else if (value > root->data) {

root->right = insert(root->right, value);

}

return root;

}

void inorder(Node\* root) {

if (root != nullptr) {

inorder(root->left);

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

inorder(root->right);

}

}

int sumOfLeafNodes(Node\* root) {

if (root == nullptr) {

return 0;

}

if (root->left == nullptr && root->right == nullptr) {

return root->data;

}

return sumOfLeafNodes(root->left) + sumOfLeafNodes(root->right);

}

};

int main() {

BST bst;

Node\* root = nullptr;

int choice, value;

while(true) {

cout << "----------Menu---------" << endl;

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

cout << "2. Inorder Traversal" << endl;

cout << "3. Sum of Leaf Nodes" << endl;

cout << "0. Exit" << endl;

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1:

cout << "Enter value to insert: ";

cin >> value;

root = bst.insert(root, value);

break;

case 2:

cout << "Inorder traversal of the tree: ";

bst.inorder(root);

cout << endl;

break;

case 3:

cout << "Sum of all leaf nodes: " << bst.sumOfLeafNodes(root) << endl;

break;

case 0:

cout << "Exiting..." << endl;

return 0;

default:

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

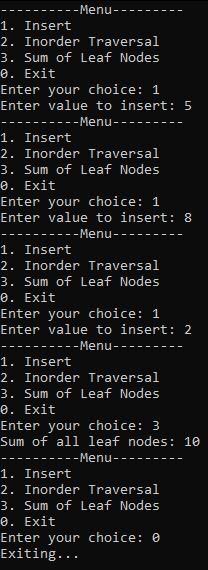
}

}

return 0;

}

## **Screenshot**



# **Problem 3**

## **Source Code**

#include <iostream>

using namespace std;

struct Node {

int data;

Node\* left;

Node\* right;

Node(int value) {

data = value;

left = nullptr;

right = nullptr;

}

};

class BST {

public:

Node\* insert(Node\* root, int value) {

if (root == nullptr) {

return new Node(value);

}

if (value < root->data) {

root->left = insert(root->left, value);

}

else if (value > root->data) {

root->right = insert(root->right, value);

}

return root;

}

void inorder(Node\* root) {

if (root != nullptr) {

inorder(root->left);

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

inorder(root->right);

}

}

void printLevel(Node\* root, int k) {

if (root == nullptr) {

return;

}

if (k == 0) {

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

return;

}

printLevel(root->left, k - 1);

printLevel(root->right, k - 1);

}

Node\* deleteLevel(Node\* root, int k) {

if (root == nullptr) {

return nullptr;

}

if (k == 0) {

delete root;

return nullptr;

}

root->left = deleteLevel(root->left, k - 1);

root->right = deleteLevel(root->right, k - 1);

return root;

}

};

int main() {

BST bst;

Node\* root = nullptr;

int choice, value, level;

while(true) {

cout << "----------Menu----------" << endl;

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

cout << "2. Inorder Traversal" << endl;

cout << "3. Print Nodes at Level K" << endl;

cout << "4. Delete Nodes at Level K" << endl;

cout << "0. Exit" << endl;

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1:

cout << "Enter value to insert: ";

cin >> value;

root = bst.insert(root, value);

break;

case 2:

cout << "Inorder traversal of the tree: ";

bst.inorder(root);

cout << endl;

break;

case 3:

cout << "Enter level to print: ";

cin >> level;

cout << "Nodes at level " << level << ": ";

bst.printLevel(root, level);

cout << endl;

break;

case 4:

cout << "Enter level to delete: ";

cin >> level;

cout << "Deleting nodes at level " << level << "..." << endl;

root = bst.deleteLevel(root, level);

cout << "Nodes at level " << level << " deleted." << endl;

break;

case 0:

cout << "Exiting..." << endl;

return 0;

default:

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

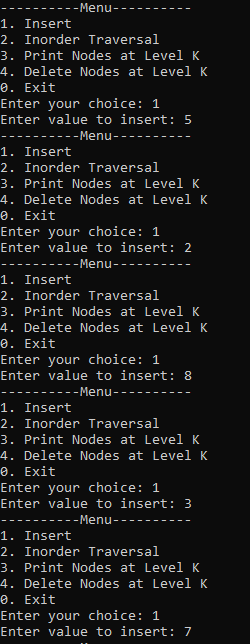
}

}

return 0;

}

## **Screenshot**



A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

# **Problem 4**

## **Source Code**

#include <iostream>

#include <cmath>

using namespace std;

struct Node {

int data;

Node\* left;

Node\* right;

Node(int value) {

data = value;

left = nullptr;

right = nullptr;

}

};

class BST {

public:

Node\* insert(Node\* root, int value) {

if (root == nullptr) {

return new Node(value);

}

if (value < root->data) {

root->left = insert(root->left, value);

}

else if (value > root->data) {

root->right = insert(root->right, value);

}

return root;

}

void inorder(Node\* root) {

if (root != nullptr) {

inorder(root->left);

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

inorder(root->right);

}

}

int closestValue(Node\* root, int key) {

int closest = root->data;

return findClosest(root, key, closest);

}

private:

int findClosest(Node\* root, int key, int closest) {

if (root == nullptr) {

return closest;

}

if (abs(root->data - key) < abs(closest - key)) {

closest = root->data;

}

if (key < root->data) {

return findClosest(root->left, key, closest);

}

else if (key > root->data) {

return findClosest(root->right, key, closest);

}

return closest;

}

};

int main() {

BST bst;

Node\* root = nullptr;

int choice, value, key, closest;

while(true) {

cout << "----------Menu----------" << endl;

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

cout << "2. Inorder Traversal" << endl;

cout << "3. Find Closest Value" << endl;

cout << "0. Exit" << endl;

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1:

cout << "Enter value to insert: ";

cin >> value;

root = bst.insert(root, value);

break;

case 2:

cout << "Inorder traversal of the tree: ";

bst.inorder(root);

cout << endl;

break;

case 3:

cout << "Enter key to find closest value: ";

cin >> key;

closest = bst.closestValue(root, key);

cout << "Closest value to " << key << " is: " << closest << endl;

break;

case 0:

cout << "Exiting..." << endl;

return 0;

default:

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

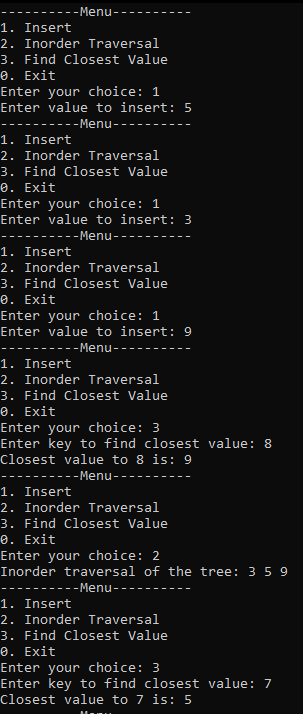
}

}

return 0;

}

## **Screenshot**



# **Problem 5**

## **Source Code**

#include <iostream>

using namespace std;

struct Node {

int data;

Node\* left;

Node\* right;

Node(int value) {

data = value;

left = nullptr;

right = nullptr;

}

};

class BST {

public:

Node\* insert(Node\* root, int value) {

if (root == nullptr) {

return new Node(value);

}

if (value < root->data) {

root->left = insert(root->left, value);

}

else if (value > root->data) {

root->right = insert(root->right, value);

}

return root;

}

void inorder(Node\* root) {

if (root != nullptr) {

inorder(root->left);

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

inorder(root->right);

}

}

bool isIdentical(Node\* root1, Node\* root2) {

if (root1 == nullptr && root2 == nullptr) {

return true;

}

if (root1 == nullptr || root2 == nullptr) {

return false;

}

return (root1->data == root2->data) &&

isIdentical(root1->left, root2->left) &&

isIdentical(root1->right, root2->right);

}

};

int main() {

BST bst;

Node\* root1 = nullptr;

Node\* root2 = nullptr;

int choice, value;

while (true) {

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

cout << "1. Insert into Tree 1" << endl;

cout << "2. Insert into Tree 2" << endl;

cout << "3. Inorder Traversal of Tree 1" << endl;

cout << "4. Inorder Traversal of Tree 2" << endl;

cout << "5. Check if Trees are Identical" << endl;

cout << "0. Exit" << endl;

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1:

cout << "Enter value to insert into Tree 1: ";

cin >> value;

root1 = bst.insert(root1, value);

break;

case 2:

cout << "Enter value to insert into Tree 2: ";

cin >> value;

root2 = bst.insert(root2, value);

break;

case 3:

cout << "Inorder traversal of Tree 1: ";

bst.inorder(root1);

cout << endl;

break;

case 4:

cout << "Inorder traversal of Tree 2: ";

bst.inorder(root2);

cout << endl;

break;

case 5:

if (bst.isIdentical(root1, root2)) {

cout << "The two trees are structurally identical." << endl;

}

else {

cout << "The two trees are NOT structurally identical." << endl;

}

break;

case 0:

cout << "Exiting..." << endl;

return 0;

default:

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

}

}

return 0;

}

## **Screenshot**