BRACT’s

Vishwakarma Institute of Information Technology, Pune

**Practical Implementation Sheet**

| **Department:** IT | **Semester:** IV | **Academic Year:** 2024-25 | **Practical No: 1** |
| --- | --- | --- | --- |
| **Class/ Division/ Batch: SY (B)- B3** | | **Roll no: 70** | |
| **Course:** Data Structures and Analysis of Algorithms | | **Name of Student**: Anushka Kadam | |

**Aim:** Create a binary search tree (BST) of and perform following operations: i) Insert ii)Display inorder iii)Search a node iv) Find height of the tree v) level wise display iv)

Delete v) Mirror

**Code:**

#include <iostream>

using namespace std;

struct Node

{

int data;

Node\* left;

Node\* right;

Node(int val) : data(val), left(nullptr), right(nullptr) {}

};

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

{

if (root == nullptr)

{

return new Node(value);

}

if (value < root->data)

{

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

}

else

{

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

}

return root;

}

void inorderTraversal(Node\* root)

{

if (root != nullptr)

{

inorderTraversal(root->left);

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

inorderTraversal(root->right);

}

}

Node\* search(Node\* root, int value)

{

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

{

return root;

}

if (value < root->data)

{

return search(root->left, value);

}

return search(root->right, value);

}

int height(Node\* root)

{

if (root == nullptr)

{

return 0;

}

int leftHeight = height(root->left);

int rightHeight = height(root->right);

return max(leftHeight, rightHeight) + 1;

}

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 = root->right;

while (temp && temp->left != nullptr)

{

temp = temp->left;

}

root->data = temp->data;

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

}

return root;

}

void mirrorImage(Node\* root)

{

if (root == nullptr)

{

return;

}

Node\* temp = root->left;

root->left = root->right;

root->right = temp;

mirrorImage(root->left);

mirrorImage(root->right);

}

int main()

{

Node\* root = nullptr;

root = insert(root, 50);

root = insert(root, 30);

root = insert(root, 20);

root = insert(root, 40);

root = insert(root, 70);

root = insert(root, 60);

root = insert(root, 80);

cout << "Inorder Traversal: ";

inorderTraversal(root);

cout << endl;

int searchVal = 40;

Node\* searchResult = search(root, searchVal);

if (searchResult)

{

cout << "Node " << searchVal << " found!" << endl;

} else

{

cout << "Node " << searchVal << " not found!" << endl;

}

cout << "Height of tree: " << height(root) << endl;

cout << "Tree (Inorder Traversal): ";

inorderTraversal(root);

cout << endl;

int deleteVal = 20;

root = deleteNode(root, deleteVal);

cout << "Tree after deleting node " << deleteVal << ": ";

inorderTraversal(root);

cout << endl;

mirrorImage(root);

cout << "Tree after creating mirror image (Inorder Traversal): ";

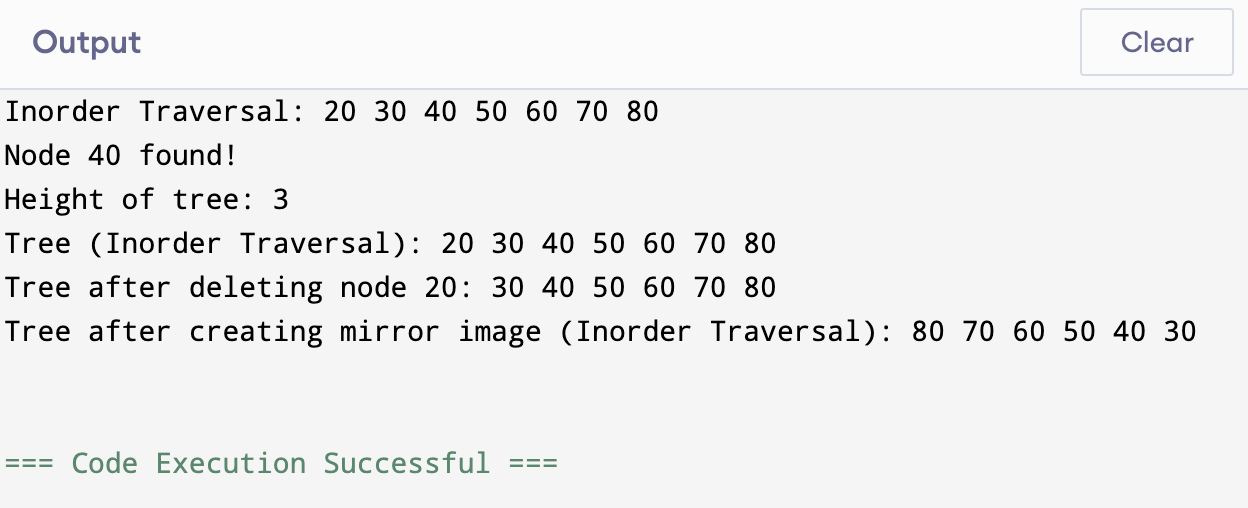
inorderTraversal(root);

cout << endl;

return 0;

}

**Output:**

****