Assignment: 03

**Course title:** Data Structure Laboratory

**Course Code:** CSE 212

**Submitted by,**

**Name:** Mohammad Fahim

**ID:** 242002112

**Section:** 6

**Department:** CSE

**Submitted to,**

**Name:** Mohammad Akbar Bin Shah

**Designation:** Lecturer, SoSET

**Date Of Submission:** 25-04-2025

**Binary Search Tree:**

**Code:**

#include <iostream>

#include <queue>

using namespace std;

struct node

{

int data;

node\* left = nullptr;

node\* right = nullptr;

};

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

{

if (root == nullptr)

{

node\* newnode = new node;

newnode->data = value;

return newnode;

}

if (value < root->data)

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

else

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

return root;

}

node\* findmin(node\* root)

{

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

root = root->left;

return root;

}

node\* deletet(node\* root, int key)

{

if (root == nullptr)

return nullptr;

if (key < root->data)

{

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

} else if (key > root->data)

{

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

} else

{

if (root->left == nullptr)

{

node\* temp = root->right;

delete root;

return temp;

}

if (root->right == nullptr)

{

node\* temp = root->left;

delete root;

return temp;

}

node\* successor = findmin(root->right);

root->data = successor->data;

root->right = deletet(root->right, successor->data);

}

return root;

}

void inorder(node\* root)

{

if (root == nullptr) return;

inorder(root->left);

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

inorder(root->right);

}

void preorder(node\* root)

{

if (root == nullptr) return;

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

preorder(root->left);

preorder(root->right);

}

void postorder(node\* root)

{

if (root == nullptr) return;

postorder(root->left);

postorder(root->right);

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

}

void levelorder(node\* root)

{

if (root == nullptr) return;

queue<node\*> q;

q.push(root);

while (!q.empty()) {

node\* temp = q.front();

q.pop();

cout << temp->data << " ";

if (temp->left) q.push(temp->left);

if (temp->right) q.push(temp->right);

}

}

node\* build() {

int n, val;

cout << "Enter number of elements: ";

cin >> n;

node\* root = nullptr;

for (int i = 0; i < n; ++i) {

cout << "Enter value #" << (i + 1) << ": ";

cin >> val;

root = insert(root, val);

}

return root;

}

int main() {

node\* root = build();

cout << "\nInorder Traversal: ";

inorder(root);

cout << "\nPreorder Traversal: ";

preorder(root);

cout << "\nPostorder Traversal: ";

postorder(root);

cout << "\nLevel Order Traversal: ";

levelorder(root);

cout << endl;

int deleteVal;

cout << "Enter value to delete: ";

cin >> deleteVal;

root = deletet(root, deleteVal);

cout << "After deletion, Inorder Traversal: ";

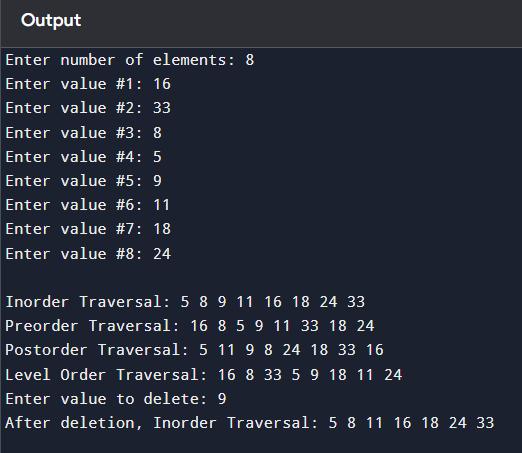
inorder(root);

cout << endl;

return 0;

}

**Input/Output:**



**Explanation:**

The approach solves the problem by using a Binary Search Tree (BST) to store and manage data efficiently.

* **Insertion:** The insert function places each new value in its correct position in the tree by comparing it to existing nodes, ensuring the BST property is maintained.
* **Deletion:** The deletet function removes a specified node while handling all cases (no child, one child, two children) and keeps the tree structure valid by using the in-order successor when needed.
* **Traversal:** The different traversal functions (inorder, preorder, postorder, levelorder) display the tree structure, helping verify that insertion and deletion were performed correctly.
* **User Input:** The build function allows the user to dynamically construct the tree and test its behavior.

This approach provides a complete solution to building, updating, and displaying a BST while preserving its core properties.