#include <iostream>

using namespace std;

// Node structure for the binary search tree (BST)

struct Node {

    int data;

    Node\* left;

    Node\* right;

};

// Function to create a new node

Node\* createNode(int value) {

    Node\* newNode = new Node();

    newNode->data = value;

    newNode->left = newNode->right = nullptr;

    return newNode;

}

// Function to insert a value into the binary search tree (BST)

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

    if (root == nullptr)

        return createNode(value);

    if (value < root->data)

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

    else if (value > root->data)

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

    return root;

}

// Function to search for a value in the binary search tree (BST)

bool searchNode(Node\* root, int value) {

    if (root == nullptr)

        return false;

    if (root->data == value)

        return true;

    if (value < root->data)

        return searchNode(root->left, value);

    else

        return searchNode(root->right, value);

}

// Function to find the minimum value node in a BST

Node\* minValueNode(Node\* node) {

    Node\* current = node;

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

        current = current->left;

    return current;

}

// Function to delete a node with given value from BST

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 {

        // Node with only one child or no child

        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 with two children: Get the inorder successor (smallest in the right subtree)

        Node\* temp = minValueNode(root->right);

        root->data = temp->data;

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

    }

    return root;

}

// Function to traverse the binary search tree (inorder traversal)

void inorderTraversal(Node\* root) {

    if (root != nullptr) {

        inorderTraversal(root->left);

        cout << root->data << " ";

        inorderTraversal(root->right);

    }

}

int main() {

    Node\* root = nullptr;

    int choice, value;

    do {

        cout << "\nBinary Search Tree Operations:" << endl;

        cout << "1. Insert" << endl;

        cout << "2. Search" << endl;

        cout << "3. Delete" << endl;

        cout << "4. Print BST (Inorder Traversal)" << endl;

        cout << "5. Exit" << endl;

        cout << "Enter your choice: ";

        cin >> choice;

        switch (choice) {

            case 1:

                cout << "Enter value to insert: ";

                cin >> value;

                root = insertNode(root, value);

                break;

            case 2:

                cout << "Enter value to search: ";

                cin >> value;

                if (searchNode(root, value))

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

                else

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

                break;

            case 3:

                cout << "Enter value to delete: ";

                cin >> value;

                root = deleteNode(root, value);

                break;

            case 4:

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

                inorderTraversal(root);

                cout << endl;

                break;

            case 5:

                cout << "Exiting the program." << endl;

                break;

            default:

                cout << "Invalid choice. Please enter a valid choice." << endl;

        }

    } while (choice != 5);

    return 0;

}