#include<iostream>

using namespace std;

class Node

{

public:

int value;

Node\* left;

Node\* right;

Node()

{

value = -1;

left = NULL;

right = NULL;

}

Node(int val)

{

value = val;

left = NULL;

right = NULL;

}

};

class BinaryTree

{

public:

Node\* root;

BinaryTree() : root(NULL) {}

void insert(int value) {

root = insertHelper(root, value);

}

Node\* insertHelper(Node\* node, int value)

{

if (node == NULL)

{

Node\* ptr = new Node(value);

return ptr;

}

if (value < node->value) {

node->left = insertHelper(node->left, value);

}

else {

node->right = insertHelper(node->right, value);

}

return node;

}

void insertLeft(int value) {

if (root == NULL) {

root = new Node(value);

}

else {

Node\* current = root;

while (current->left != NULL) {

current = current->left;

}

current->left = new Node(value);

}

}

void insertRight(int value) {

if (root == NULL) {

root = new Node(value);

}

else {

Node\* current = root;

while (current->right != NULL) {

current = current->right;

}

current->right = new Node(value);

}

}

Node\* findMin(Node\* node)

{

while (node->left != NULL) {

node = node->left;

}

return node;

}

void deleteNode(int value) {

root = deleteNodeHelper(root, value);

}

void deleteLeft(int value) {

Node\* current = root;

Node\* parent = NULL;

while (current != NULL && current->value != value) {

parent = current;

if (value < current->value) {

current = current->left;

}

else {

current = current->right;

}

}

if (current != NULL && current->left != NULL) {

deleteNode(current->left->value);

}

}

void deleteRight(int value) {

Node\* current = root;

Node\* parent = NULL;

while (current != NULL && current->value != value) {

parent = current;

if (value < current->value) {

current = current->left;

}

else {

current = current->right;

}

}

if (current != NULL && current->right != NULL) {

deleteNode(current->right->value);

}

}

Node\* deleteNodeHelper(Node\* node, int value)

{

if (node == NULL) {

return NULL;

}

if (value < node->value) {

node->left = deleteNodeHelper(node->left, value);

}

else if (value > node->value) {

node->right = deleteNodeHelper(node->right, value);

}

else {

if (node->left == NULL && node->right == NULL) {

delete node;

return NULL;

}

else if (node->left == NULL) {

Node\* temp = node->right;

delete node;

return temp;

}

else if (node->right == NULL) {

Node\* temp = node->left;

delete node;

return temp;

}

else {

Node\* temp = findMin(node->right);

node->value = temp->value;

node->right = deleteNodeHelper(node->right, temp->value);

}

}

return node;

}

void inOrder(Node\* node) {

if (node == NULL)

return;

inOrder(node->left);

cout << node->value << " ";

inOrder(node->right);

}

void preOrder(Node\* node) {

if (node == NULL)

return;

cout << node->value << " ";

preOrder(node->left);

preOrder(node->right);

}

void postOrder(Node\* node) {

if (node == NULL)

return;

postOrder(node->left);

postOrder(node->right);

cout << node->value << " ";

}

int find(Node\* node, int value, int& count) {

if (node == NULL)

return -1;

count++;

if (node->value == value)

return node->value;

if (value < node->value)

return find(node->left, value, count);

return find(node->right, value, count);

}

//void display(Node\* node, int space) {

// if (node == NULL) return;

// space += 5;

// display(node->right, space);

// cout << endl;

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

// cout << node->value << "\n";

// display(node->left, space);

//}

//void display(Node\* node, int depth) {

// if (node == nullptr) return;

// display(node->right, depth + 1);

// for (int i = 0; i < depth; ++i) cout << " ";

// cout << node->value << endl;

// display(node->left, depth + 1);

//}

void traverseInOrder() {

inOrder(root);

cout << endl;

}

void traversePreOrder() {

preOrder(root);

cout << endl;

}

void traversePostOrder() {

postOrder(root);

cout << endl;

}

int find(int value) {

int count = 0;

find(root, value, count);

return count;

}

//void display() {

// display(root, 0);

//}

};

int main() {

BinaryTree tree;

tree.insert(10);

tree.insert(5);

tree.insert(20);

tree.insert(3);

tree.insert(7);

tree.insertLeft(1);

tree.insertRight(25);

cout << "In-order traversal: ";

tree.traverseInOrder();

cout << "Pre-order traversal: ";

tree.traversePreOrder();

cout << "Post-order traversal: ";

tree.traversePostOrder();

cout << "Number of iterations to find 7: " << tree.find(7) << endl;

cout << "Tree display:\n";

tree.traverseInOrder();

tree.deleteNode(10);

cout << "Tree display after deleting 10:\n";

tree.traverseInOrder();

tree.deleteLeft(5);

cout << "Tree display after deleting left child of 5:\n";

tree.traverseInOrder();

tree.deleteRight(20);

cout << "Tree display after deleting right child of 20:\n";

tree.traverseInOrder();

return 0;

}

