#include <iostream>

using namespace std;

// Node class to represent each node in the tree

class Node {

public:

int data;

Node\* left;

Node\* right;

// Constructor to initialize a new node

Node(int value) {

data = value;

left = right = nullptr;

}

};

// BinaryTree class to handle tree operations

class BinaryTree {

public:

Node\* root;

// Constructor to initialize an empty tree

BinaryTree() {

root = nullptr;

}

// Method to insert a node into the tree (simple binary search tree insert)

void insert(int value) {

root = insertHelper(root, value);

}

private:

// Helper method for inserting a node recursively

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

if (node == nullptr) {

return new Node(value); // Create a new node if the position is found

}

if (value < node->data) {

node->left = insertHelper(node->left, value); // Insert in the left subtree

} else {

node->right = insertHelper(node->right, value); // Insert in the right subtree

}

return node;

}

public:

// Pre-order traversal: root -> left -> right

void preOrder(Node\* node) {

if (node == nullptr) {

return;

}

cout << node->data << " "; // Visit the node

preOrder(node->left); // Traverse left subtree

preOrder(node->right); // Traverse right subtree

}

// In-order traversal: left -> root -> right

void inOrder(Node\* node) {

if (node == nullptr) {

return;

}

inOrder(node->left); // Traverse left subtree

cout << node->data << " "; // Visit the node

inOrder(node->right); // Traverse right subtree

}

// Post-order traversal: left -> right -> root

void postOrder(Node\* node) {

if (node == nullptr) {

return;

}

postOrder(node->left); // Traverse left subtree

postOrder(node->right); // Traverse right subtree

cout << node->data << " "; // Visit the node

}

// Wrapper methods to call traversal starting from the root

void preOrderTraversal() {

preOrder(root);

cout << endl;

}

void inOrderTraversal() {

inOrder(root);

cout << endl;

}

void postOrderTraversal() {

postOrder(root);

cout << endl;

}

};

int main() {

BinaryTree tree;

// Insert nodes into the tree

tree.insert(50);

tree.insert(30);

tree.insert(20);

tree.insert(40);

tree.insert(70);

tree.insert(60);

tree.insert(80);

cout << "Pre-order Traversal: ";

tree.preOrderTraversal(); // Pre-order: root, left, right

cout << "In-order Traversal: ";

tree.inOrderTraversal(); // In-order: left, root, right

cout << "Post-order Traversal: ";

tree.postOrderTraversal(); // Post-order: left, right, root

return 0;

}