#include <iostream>

using namespace std;

// Node class

class Node {

int key;

Node\* left;

Node\* right;

public:

Node() { key=-1; left=NULL; right=NULL; };

void setKey(int aKey) { key = aKey; };

void setLeft(Node\* aLeft) { left = aLeft; };

void setRight(Node\* aRight) { right = aRight; };

int Key() { return key; };

Node\* Left() { return left; };

Node\* Right() { return right; };

};

// Tree class

class Tree {

Node\* root;

public:

Tree();

~Tree();

Node\* Root() { return root; };

void addNode(int key);

void inOrder(Node\* n);

void preOrder(Node\* n);

void postOrder(Node\* n);

private:

void addNode(int key, Node\* leaf);

void freeNode(Node\* leaf);

};

// Constructor

Tree::Tree() {

root = NULL;

}

// Destructor

Tree::~Tree() {

freeNode(root);

}

// Free the node

void Tree::freeNode(Node\* leaf)

{

if ( leaf != NULL )

{

freeNode(leaf->Left());

freeNode(leaf->Right());

delete leaf;

}

}

// Add a node

void Tree::addNode(int key) {

// No elements. Add the root

if ( root == NULL ) {

cout << "add root node ... " << key << endl;

Node\* n = new Node();

n->setKey(key);

root = n;

}

else {

cout << "add other node ... " << key << endl;

addNode(key, root);

}

}

// Add a node (private)

void Tree::addNode(int key, Node\* leaf) {

if ( key <= leaf->Key() ) {

if ( leaf->Left() != NULL )

addNode(key, leaf->Left());

else {

Node\* n = new Node();

n->setKey(key);

leaf->setLeft(n);

}

}

else {

if ( leaf->Right() != NULL )

addNode(key, leaf->Right());

else {

Node\* n = new Node();

n->setKey(key);

leaf->setRight(n);

}

}

}

// Print the tree in-order

// Traverse the left sub-tree, root, right sub-tree

void Tree::inOrder(Node\* n) {

if ( n ) {

inOrder(n->Left());

cout << n->Key() << " ";

inOrder(n->Right());

}

}

// Print the tree pre-order

// Traverse the root, left sub-tree, right sub-tree

void Tree::preOrder(Node\* n) {

if ( n ) {

cout << n->Key() << " ";

preOrder(n->Left());

preOrder(n->Right());

}

}

// Print the tree post-order

// Traverse left sub-tree, right sub-tree, root

void Tree::postOrder(Node\* n) {

if ( n ) {

postOrder(n->Left());

postOrder(n->Right());

cout << n->Key() << " ";

}

}

// Test main program

int main() {

Tree\* tree = new Tree();

tree->addNode(30);

tree->addNode(10);

tree->addNode(20);

tree->addNode(40);

tree->addNode(50);

cout << "In order traversal" << endl;

tree->inOrder(tree->Root());

cout << endl;

cout << "Pre order traversal" << endl;

tree->preOrder(tree->Root());

cout << endl;

cout << "Post order traversal" << endl;

tree->postOrder(tree->Root());

cout << endl;

delete tree;

return 0;

}

.

OUTPUT

add root node ... 30

add other node ... 10

add other node ... 20

add other node ... 40

add other node ... 50

In order traversal

10 20 30 40 50

Pre order traversal

30 10 20 40 50

Post order traversal

20 10 50 40 30