// Lab13 Integer Binary Search Tree (BST)

// This program builds an integer binary search tree of 5 nodes.

// First to use the enclosed draw() helper to show the BST upon each node insertion.

// Then display the BST content with inorder, preorder, and postorder traversal.

//

#include <iostream>

#include <iomanip>

#include "IntBinaryTree.h"

using namespace std;

int main()

{

IntBinaryTree tree;

// Insert some nodes.

cout << "Inserting nodes:\n";

tree.insert(8);

tree.draw();

tree.insert(4);

tree.draw();

tree.insert(12);

tree.draw();

tree.insert(2);

tree.draw();

tree.insert(6);

tree.draw();

tree.insert(1);

tree.draw();

tree.insert(3);

tree.draw();

tree.insert(5);

tree.draw();

tree.insert(7);;

tree.draw();

tree.insert(14);;

tree.draw();

tree.insert(10);;

tree.draw();

tree.insert(9);;

tree.draw();

tree.insert(11);;

tree.draw();

tree.insert(13);;

tree.draw();

tree.insert(15);;

tree.draw();

tree.remove(8);

cout << "remove 8\n";

tree.draw();

tree.remove(12);

cout << "remove 12\n";

tree.draw();

tree.remove(5);

cout << "remove 5\n";

tree.draw();

cout << "Inorder traversal:\n";

tree.displayInOrder();

cout << "\nPreorder traversal:\n";

tree.displayPreOrder();

cout << "\nPostorder traversal:\n";

tree.displayPostOrder();

return 0;

}

#ifndef INTBINARYTREE\_H

#define INTBINARYTREE\_H

#include <iostream>

#include <string>

#include <iomanip>

using namespace std;

class IntBinaryTree

{

private:

struct TreeNode {

int value;

TreeNode \*left;

TreeNode \*right;

};

TreeNode \*root;

// Private helpers - use recursion

void insert(TreeNode \*&, TreeNode \*&);

void remove(TreeNode \*&, int);

void makeDeletion(TreeNode \*&);

// void destroySubTree(TreeNode \*&);

void displayInOrder(TreeNode \*) const;

void displayPreOrder(TreeNode \*) const;

void displayPostOrder(TreeNode \*) const;

public:

IntBinaryTree() { root = nullptr; }

~IntBinaryTree() { } //destroySubTree(root); }

void insert(int);

void remove(int value) { remove(root, value); }

void displayInOrder() const { displayInOrder(root); }

void displayPreOrder() const { displayPreOrder(root); }

void displayPostOrder() const { displayPostOrder(root); }

void draw(TreeNode\* tree, std::string lpad, std::string rpad) const {

std::string pad = lpad.substr(0, lpad.size() - 1);

if (tree == nullptr) return;

draw(tree->right, rpad + " |", rpad + " ");

std::cout << pad << "+--" << std::setw(3) << tree->value << std::endl;

draw(tree->left, lpad + " ", lpad + " |");

}

void draw() const {

std::cout << std::endl;

this->draw(root, " ", " ");

// std::cout << std::endl;

}

};

////////////////////

// private Helpers

void IntBinaryTree::insert(TreeNode \*&node, TreeNode \*&newNode) {

if(node == nullptr) node = newNode;

else if(newNode->value < node->value ){

insert(node->left,newNode);

}

else if (newNode->value > node->value ){

insert(node->right,newNode);

}

}

void IntBinaryTree::remove(TreeNode \*&node, int n) {

if(!node)

cout << " node " << n << "doesn't exist!\n";

else if (n < node->value )

remove ( node -> left, n);

else if (node->value < n )

remove (node -> right,n);

else

makeDeletion( node );

}

void IntBinaryTree::makeDeletion(TreeNode \*&node) {

TreeNode \*tempNode;

if(!node){

}

else if(!node->right) {

tempNode = node;

node = node -> left;

delete tempNode;

}

else if( !node -> left){

tempNode = node;

node = node ->right;

delete tempNode;

}

else{

tempNode= node->right;

TreeNode \*leftmost = tempNode;

while(leftmost->left){

leftmost = leftmost -> left;

if(!leftmost->left) break;

tempNode = leftmost;

}

node ->value = leftmost->value;

if(leftmost == tempNode)

node->right = tempNode->right;

else

tempNode->left = leftmost ->right;

delete leftmost;

}

}

// void IntBinaryTree::destroySubTree(TreeNode \*&node) {}

void IntBinaryTree::displayInOrder(TreeNode \*node) const {

if(node) {

displayInOrder(node->left);

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

displayInOrder(node->right);

}

}

void IntBinaryTree::displayPreOrder(TreeNode \*node) const {

if(node) {

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

displayPreOrder(node->left);

displayPreOrder(node->right);

}

}

void IntBinaryTree::displayPostOrder(TreeNode \*node) const {

if(node) {

displayPostOrder(node->left);

displayPostOrder(node->right);

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

}

}

////////////////////

// Public Method

void IntBinaryTree::insert(int value) {

TreeNode \*newNode = new TreeNode;

newNode->value = value;

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

insert(root, newNode);

}

#endif

