// 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"

#include <boost/tokenizer.hpp>

#include <string>

#include <sstream>

#include <vector>

#include <string.h>

#include <stdio.h>

using namespace std;

void menu(){

cout << " --- Integer Binary Tree Menu -----" << endl;

cout << "B - to batch enter FeetInches " << endl;

cout << "C - to clear BST " << endl;

cout << "D - to display BST" << endl;

cout << "I - to Insert " << endl;

cout << "R - to Remove " << endl;

cout << "1 - to pre-order Traversal " << endl;

cout << "2 - to In-order Traversal " << endl;

cout << "3 - to Post - Order Traversal" << endl;

cout << "Q - to quite the program" << endl;

}

int main()

{

menu();

IntBinaryTree tree;

char command;

bool stay = true;

while (stay){

cout << " Enter your command: "<<endl;

cin >> command;

cin.ignore();

stringstream ss;

string tokens;

string input, token;

switch(command){

case 'B':

case 'b':{

cout << "Enter a list of Integers separated by comma(,): ";

getline (cin, input); // user input -> string

stringstream ss(input); // string -> stream

while ( getline(ss, token, ',') ) { // stream -> string token

stringstream sst(token); // string token -> stream token

int ff;

sst >> ff; // stream token -> double token

//tree.draw();

tree.insert(ff);;

}

tree.draw();

break;

}

case 'D':

case 'd':{

tree.draw();

break;

}

case 'q':

case 'Q':{

stay = false;

break;

}

case'i':

case'I':{

int num;

cout << " Enter a number to insert: ";

cin >> num;

tree.insert(num);;

break;

}

case'1':{

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

tree.displayPreOrder();

cout << "\n";

break;

}

case'2':{

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

tree.displayInOrder();

cout << "\n";

break;

}

case'3':{

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

tree.displayPostOrder();

cout << "\n";

break;

}

case'R':

case'r':{

int RemoveNum;

cout << "\nEnter the number you want to remove:";

cin >> RemoveNum;

tree.remove(RemoveNum);

break;

}

case'c':

case'C':{

tree.destroySubTree();

break;

}

}

}

}

#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 destroySubTree(){ 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 destroySubTree(TreeNode \*&){

if(node) {

destroySubTree(node->left);

node->left = nullptr;

destroySubTree(node->right);

node->right = nullptr;

delete node;

node = nullptr;

}

}

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

