BST.h start here:

#ifndef BST\_H

#define BST\_H

#include <iostream>

using namespace std;

class BST {

private:

class binNode {

public:

int value;

binNode\* lchild;

binNode\* rchild;

binNode() :rchild(0), lchild(0) {}

binNode(int item) : value(item), rchild(0), lchild(0) {}

};

binNode\* myRoot;

//helper functions

void helpDestroy(binNode\*);

binNode\* helpCopy(binNode\*);

void helpDisplay(ostream&, int, binNode\*);

int helpcount(binNode\*);

void helpInorder(binNode\*);

int helpLevel(binNode\*, int, int);

int helpHeight(binNode\*);

int helpLeafCount(binNode\*);

bool helpIsBST(binNode\*, int, int);

public:

BST();

~BST();

BST(const BST&);

//data structure methods

void insert(int);

void erase(int);

bool search(int);

int Size();

int count();

void displayInorder();

int level(int item);

int height();

int leafCount();

binNode\* Mirror(binNode\*);

BST mirrorImage();

bool isBST();

friend ostream& operator<<(ostream&, BST&);

};

#endif

BST.cpp starts here:

#include"BST.h"

#include <iomanip>

#include <climits>

//all helper functions here

void BST::helpDestroy(binNode\* node) {

if (node != 0) {

helpDestroy(node->lchild);

helpDestroy(node->rchild);

delete node;

}

}

BST::binNode\* BST::helpCopy(binNode\* node) {

if (node != 0) {

binNode\* temp = new binNode(node->value);

temp->lchild = helpCopy(node->lchild);

temp->rchild = helpCopy(node->rchild);

return temp;

}

return 0;

}

void BST::helpDisplay(ostream& out, int indent, binNode\* node) {

if (node != 0) {

helpDisplay(out, indent + 8, node->rchild);

out << setw(indent) << node->value << endl;

helpDisplay(out, indent + 8, node->lchild);

}

}

void BST::helpInorder(binNode\* node) {

if (node != 0) {

helpInorder(node->lchild);

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

helpInorder(node->rchild);

}

}

int BST::helpcount(binNode\* node) {

if (node != 0)

return (1 + helpcount(node->lchild) + helpcount(node->rchild));

else

return 0;

}

//All my class functions here

BST::BST() :myRoot(0) {}

BST::~BST() { helpDestroy(myRoot); myRoot = 0; }//Destructor

BST::BST(const BST& B) { myRoot = helpCopy(B.myRoot); }//Copy Constructor

void BST::insert(int item) { //Assuming all items are unique

bool found = false;

binNode\* child = myRoot;

binNode\* parent = myRoot;

//first search for the item and keep track of parent.

while (found != true && child != 0) {

if (item > child->value) { parent = child; child = child->rchild; }

else if (item < child->value) { parent = child; child = child->lchild; }

else found = true;

}

if (!found) {

if (myRoot == 0) myRoot = new binNode(item);//the first node in the tree

else if (item < parent->value)parent->lchild = new binNode(item);

else parent->rchild = new binNode(item);

}

else cout << "Item found, nothing to add \n";

}

void BST::erase(int item) {

binNode\* parent = 0;

binNode\* n2del = myRoot;

bool found = false;

//find the node

while (found == false && n2del != 0) {

if (item < n2del->value) { parent = n2del; n2del = n2del->lchild; }

else if (item > n2del->value) { parent = n2del; n2del = n2del->rchild; }

else found = true;

}

// check if node has both childred

if (found == false) { cout << "nothing to delete \n"; return; }

binNode\* temp = n2del;

if (temp->lchild != 0 && temp->rchild != 0) {

parent = temp; temp = temp->rchild;

while (temp->lchild != 0) {

parent = temp; temp = temp->lchild;

}

n2del->value = temp->value;

n2del = temp;

}

//now delete

if (parent == 0) {//root is deleted

myRoot = myRoot->lchild != 0 ? myRoot->lchild : myRoot->rchild;

}

else {

if (parent->lchild == n2del) {

parent->lchild = n2del->lchild != 0 ? n2del->lchild : n2del->rchild;

}

else {

parent->rchild = n2del->lchild != 0 ? n2del->lchild : parent->rchild = n2del->rchild;

}

}

delete n2del;

}

bool BST::search(int)

{

return false;

}

ostream& operator<<(ostream& out, BST& B) {

B.helpDisplay(out, 0, B.myRoot);

return out;

}

int BST::Size() { return helpcount(myRoot); }

int BST::count() { return helpcount(myRoot); }

void BST::displayInorder() { helpInorder(myRoot); }

int BST::helpHeight(binNode\* node)

{

if (node == nullptr) {

return -1; // an empty tree has height -1

}

int leftHeight = helpHeight(node->lchild);

int rightHeight = helpHeight(node->rchild);

return 1 + max(leftHeight, rightHeight);

}

int BST::helpLeafCount(binNode\* node)

{

if (node == nullptr)

{

return 0;

}

else if (node->lchild == nullptr && node->rchild == nullptr)

{

return 1;

}

else {

return helpLeafCount(node->lchild) + helpLeafCount(node->rchild);

}

}

int BST::helpLevel(binNode\* node, int item, int level) {

if (node == nullptr) {

// item not found in tree

return -1;

}

else if (node->value == item) {

// item found at current level

return level;

}

else if (item < node->value) {

// item is in left subtree

return helpLevel(node->lchild, item, level + 1);

}

else if (item > node->value) {

// item is in right subtree

return helpLevel(node->rchild, item, level + 1);

}

else {

// item is equal to node value (found at current level)

return level;

}

}

int BST::leafCount() {

return helpLeafCount(myRoot);

}

int BST::level(int item)

{

return helpLevel(myRoot, item, 0);

}

int BST::height()

{

return helpHeight(myRoot);

}

BST::binNode\* BST::Mirror(binNode\* node) {

if (node == nullptr) {

return node;

}

else {

binNode\* temp;

Mirror(node->lchild);

Mirror(node->rchild);

temp = node->lchild;

node->lchild = node->rchild;

node->rchild = temp;

}

return node;

}

BST BST::mirrorImage() {

BST newBST;

newBST.myRoot = Mirror(myRoot);

return newBST;

}

bool BST::isBST() {

return helpIsBST(myRoot, INT\_MIN, INT\_MAX);

}

bool BST::helpIsBST(binNode\* node, int minVal, int maxVal) {

if (node == nullptr) {

return true;

}

if (node->value < minVal || node->value > maxVal) {

return false;

}

return helpIsBST(node->lchild, minVal, node->value - 1) &&

helpIsBST(node->rchild, node->value + 1, maxVal);

}

Main.cpp starts here:

#include<iostream>

#include<time.h>

#include"BST.h"

using namespace std;

int main() {

BST B;

B.insert(19);

B.insert(3);

B.insert(17);

B.insert(2);

B.insert(20);

B.insert(30);

B.insert(8);

cout << "Original B: " << endl;

cout << B << endl;

cout << "Is B a BST? " << B.isBST() << endl;

cout << "The level of 8 is: " << B.level(8) << endl;

cout << "Inorder traversal: ";

B.displayInorder();

cout << endl;

cout << "Deleting node 3: " << endl;

B.erase(3);

cout << "BST after deleting node 3: " << endl;

cout << B << endl;

cout << "Number of nodes in BST: " << B.Size() << endl;

cout << "Size of BST: " << B.Size() << endl;

cout << "The height of the BST is: " << B.height() << endl;

cout << "The number of leaves in the BST is: " << B.leafCount() << endl;

BST MB = B.mirrorImage();

cout << "Mirror image of the BST: " << endl;

cout << MB << endl;

system("pause");

return 0;

}

A screenshot of a computer

Description automatically generated