#ifndef NODE\_H

#define NODE\_H

class Node

{

public:

Node();

Node(int k);

virtual ~Node();

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

int getKey(){ return key;}

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

Node \*getRight(){ return right;}

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

Node \*getLeft(){ return left;}

void setParent(Node \*val){ parent = val;}

Node \*getParent(){ return parent;}

protected:

private:

int key;

Node \*right;

Node \*left;

Node \*parent;

};

#endif // NODE\_H

#include "Node.h"

Node::Node()

{

this->key = 0;

this->left = nullptr;

this->right = nullptr;

this->parent = nullptr;

}

Node::Node(int k)

{

this->key = k;

this->left = nullptr;

this->right = nullptr;

this->parent = nullptr;

}

Node::~Node()

{

}

#ifndef BST\_H

#define BST\_H

#include "Node.h"

#include<iostream>

using namespace std;

class BST

{

public:

BST();

virtual ~BST();

void setRoot(Node \*val){ root = val;}

Node\* getRoot(){ return root;}

bool insertNode(Node \*p);

void insertNodeRE(Node \*p);

void travelNLR();

void travelLNR();

void travelLRN();

Node\* search\_x(int k);

Node\* search\_xRE(int val);

void deleteNode(Node \*n);

void deleteNodeRE(Node \*p);

int sumNode(Node \*r);

int findMax(Node \*r);

int findMin(Node \*r);

int countNode(Node \*r);

int countNodeLeaf(Node \*r);

protected:

private:

Node \*root;

Node\* insertNodeRe(Node \*r, Node \*p);

void NLR(Node \*r);

void LNR(Node \*r);

void LRN(Node \*r);

Node\* search\_xRe(Node \*r, int val);

Node\* deleteRe(Node \*r, Node \*p);

Node\* findNodeMin(Node \*del, Node \*curr);

};

#endif // BST\_H

#include "BST.h"

BST::BST()

{

this->root = nullptr;

}

BST::~BST()

{

//dtor

}

bool BST::insertNode(Node \*n)

{

Node \*p = this->root;

Node \*T;

if(root == nullptr)

{

root = n;

return true;

}

while(p)

{

T = p;

if(p->getKey() == n->getKey()) return false;

else if(p->getKey() > n->getKey())

{

p = p->getLeft();

}

else

{

p = p->getRight();

}

}

if(T->getKey() > n->getKey())

{

T->setLeft(n);

}

else if(T->getKey() < n->getKey())

{

T->setRight(n);

}

return true;

}

Node\* BST::insertNodeRe(Node \*r, Node \*p)

{

if(r == nullptr)

{

return p;

}

if(r->getKey() > p->getKey())

{

r->setLeft(insertNodeRe(r->getLeft(), p));

// insertNodeRe(r->getLeft(),p);

// return insertNodeRe(r->getLeft(),p);

// khong return ve can set lai de tao lien ket

}

else if(r->getKey() < p->getKey())

{

r->setRight(insertNodeRe(r->getRight(), p));

// insertNodeRe(r->getRight(),p);

}

}

void BST::insertNodeRE(Node \*p)

{

setRoot(insertNodeRe(this->root, p));

}

void BST::NLR(Node \*r)

{

if(r)

{

cout << r->getKey() << " ";

NLR(r->getLeft());

NLR(r->getRight());

}

}

void BST::LNR(Node \*r)

{

if(r)

{

LNR(r->getLeft());

cout << r->getKey() << " ";

LNR(r->getRight());

}

}

void BST::LRN(Node \*r)

{

if(r)

{

LRN(r->getLeft());

LRN(r->getRight());

cout << r->getKey() << " ";

}

}

void BST::travelNLR()

{

NLR(this->root);

}

void BST::travelLNR()

{

LNR(this->root);

}

void BST::travelLRN()

{

LRN(this->root);

}

Node\* BST::search\_x(int k)

{

if(this->root == nullptr)

{

return nullptr;

}

else

{

Node \*p = this->root;

while(p != nullptr)

{

if(p->getKey() > k)

{

p = p->getLeft();

}

else if(p->getKey() < k)

{

p = p->getRight();

}

else

{

return p;

}

}

}

}

Node\* BST::search\_xRe(Node \*r, int val)

{

if(r == nullptr)

{

return nullptr;

}

else

{

if(r->getKey() > val)

{

// r->setLeft(search\_xRe(r->getLeft(), val));

return search\_xRe(r->getLeft(), val);

}

else if(r->getKey() < val)

{

// r->setRight(search\_xRe(r->getRight(), val));

return search\_xRe(r->getRight(), val);

}

else return r;

}

}

Node\* BST::search\_xRE(int val)

{

return search\_xRe(this->root, val);

}

//void BST::deleteNode(Node\* n){

// Node\* p=n;

// if(p->Getleft()==nullptr&&n->Getright()==nullptr)

// delete n;

// else{

// if(p->Getright()!=nullptr){

// while(p->Getleft()!=nullptr)//

// p=p->Getleft();

// n->Setkey(p->Getkey());

// //sinh vien code

// delete p;

// }else{

// while(p->Getright()!=nullptr)//

// p=p->Getright();

//

// n->Setkey(p->Getkey());

// //sinh vien code

// delete p;

// }

// }

//}

//Phần tử thay thế ptt :

//\* Cách 1: là phần tử lớn nhất trong cây con bên trái p

//\* Cách 2: là phần tử nhỏ nhất trong cây con bên phải p

//Node\* BST::findNodeMin(Node \*del, Node \*curr)

//{

// if(curr->getLeft() != nullptr)

// {

// curr->setLeft(findNodeMin(del, curr->getLeft()));

// return curr;

// }

// else{

// del->setKey(curr->getKey());

// del = curr;

// Node \*cur = curr->getLeft();

// delete del;

//

// }

//}

Node\* BST::deleteRe(Node \*r, Node \*p)

{

if(r == nullptr) return nullptr;

else

{

if(r->getKey() > p->getKey())

{

r->setLeft(deleteRe(r->getLeft(), p));

}

else if(r->getKey() < p->getKey())

{

r->setRight(deleteRe(r->getRight(), p));

}

else

{

Node \*del = r;

if(r->getLeft() == nullptr)

{

Node\* tmp = r->getRight();

delete r;

return tmp;

}

else if(r->getRight() == nullptr)

{

Node\* tmp = r->getLeft();

delete r;

return tmp;

}

else

{

Node \*parent = r;

Node \*tmp = r->getRight();

while(tmp->getLeft() != nullptr)

{

parent = tmp;

tmp = tmp->getLeft();

}

r->setKey( tmp->getKey());

if (parent != r)

parent->setLeft(tmp->getRight());

else

parent->setRight(tmp->getRight());

delete tmp;

return r;

}

}

}

}

void BST::deleteNodeRE(Node \*p)

{

setRoot(deleteRe(this->root, p));

}

int sum = 0;

int BST::sumNode(Node \*r)

{

if(r)

{

sum += r->getKey();

sumNode(r->getLeft());

sumNode(r->getRight());

}

return sum;

// if(r == nullptr )

// return 0;

// return r->getKey() + sumNode(r->getLeft()) + sumNode(r->getRight());

}

int BST::findMax(Node \*r)

{

// dk tro toi right, vi neu tro toi chinh no vi ko the tra ve value gi co y nghia

if(r->getRight() == nullptr)

return r->getKey();

return findMax(r->getRight());

}

int BST::findMin(Node \*r)

{

if(r->getLeft() == nullptr)

return r->getKey();

return findMin(r->getLeft());

}

int BST::countNode(Node \*r)

{

if(r == nullptr)

return 0;

return 1 + countNode(r->getLeft()) + countNode(r->getRight());

}

int BST::countNodeLeaf(Node \*r)

{

if(r == nullptr) return 0;

else{

if(r->getLeft() == nullptr && r->getRight() == nullptr)

return 1;

else return countNodeLeaf(r->getLeft()) + countNodeLeaf(r->getRight());

}

}

#include <iostream>

#include "BST.h"

using namespace std;

int main()

{

// BST tree;

// Node \*n;

// n = new Node(9);

// tree.insertNodeRe(tree.getRoot(),9);

//// n = new Node(20);

// tree.insertNodeRe(20);

//// n = new Node(17);

// tree.insertNodeRe(17);

//// n = new Node(25);

// tree.insertNodeRe(25);

//// n = new Node(12);

// tree.insertNodeRe(12);

//// n = new Node(11);

// tree.insertNodeRe(11);

//// n = new Node(13);

// tree.insertNodeRe(13);

// tree.travelNLR();

// cout << endl;

// tree.deleteNodeRE(9);

// tree.travelNLR();

BST \*tree = new BST();

Node \*n;

n = new Node(9);

tree->insertNodeRE(n);

n = new Node(2);

tree->insertNodeRE(n);

n = new Node(20);

tree->insertNodeRE(n);

n = new Node(17);

tree->insertNodeRE(n);

n = new Node(25);

tree->insertNodeRE(n);

n = new Node(12);

tree->insertNodeRE(n);

n = new Node(11);

tree->insertNodeRE(n);

n = new Node(13);

tree->insertNodeRE(n);

// n = new Node(15);

// tree->insertNodeRE(n);

tree->travelNLR();

cout << endl;

// tree->deleteNodeRE(new Node(9));

// tree->travelNLR();

cout << tree->sumNode(tree->getRoot());

cout << endl;

cout << tree->findMax(tree->getRoot()) << " " << tree->findMin(tree->getRoot());

cout << endl;

cout << tree->countNode(tree->getRoot());

cout << endl;

cout << tree->countNodeLeaf(tree->getRoot());

// BST \*tree = new BST();

// Node \*n;

// n = new Node(10);

// tree->insertNode(n);

// n = new Node(19);

// tree->insertNode(n);

// n = new Node(9);

// tree->insertNode(n);

// n = new Node(3);

// tree->insertNode(n);

// n = new Node(19);

// tree->insertNode(n);

// n = new Node(8);

// tree->insertNode(n);

// n = new Node(4);

// tree->insertNode(n);

// n = new Node(1);

// tree->insertNode(n);

// n = new Node(15);

// tree->insertNode(n);

// tree->travelNLR();

return 0;

}