#include <iostream>

#include <algorithm>

using namespace std;

class TreeNode{

public :

string key ;

string value ;

int height;

TreeNode\* left;

TreeNode\* right;

Tree(string k , string v){

key = k ;

value = v;

height = 1;

left = NULL;

right = NULL;

}

};

class AVLTree{

private :

TreeNode\* root ;

int height(TreeNode\* node ){

if (node != NULL) {

return node->height;

} else {

return 0;

}

}

int balanceFactor(TreeNode\* node){

if ( node != NULL)

return height(node->left) - height(node->right);

else

return 0;

}

TreeNode\* rotateRight(TreeNode\* y){

TreeNode\* x = y -> left;

TreeNode\* T2 = x->right;

x->right = y;

y->left = T2;

y->height = 1 + max(height(y->left), height(y->right));

x->height = 1 + max(height(x->left), height(x->right));

return x;

}

TreeNode\* rotateLeft(TreeNode\* x) {

TreeNode\* y = x->right;

TreeNode\* T2 = y->left;

y->left = x;

x->right = T2;

x->height = 1 + max(height(x->left), height(x->right));

y->height = 1 + max(height(y->left), height(y->right));

return y;

}

TreeNode \*insert (TreeNode\* node , string key, string value){

if(!node)

return new TreeNode(key,value);

if( key < node->key)

node -> left = insert(node ->left , key , value);

else if(key > node -> key)

node-> right = insert(node->right,key ,value);

else{

node->value = value;

return node ;

}

node->height = 1 + max(height(node ->left), height(node->right));

int balance = balanceFactor(node);

if(balance > 1 && key < node->left->key)

return rotateRight(node);

if(balance < -1 && key > node->right->key)

return rotateLeft(node);

if(balance > 1 && key > node->left->key){

node->left = rotateLeft(node->left);

return rotateRight(node);

}

if (balance < -1 && key < node->right->key) {

node->right = rotateRight(node->right);

return rotateLeft(node);

}

return node;

}

TreeNode\* minValueNode(TreeNode\* node){

TreeNode\* current = node;

while(current -> left){

current = current -> left;

}

return current;

}

string find(TreeNode\* node ,string key ){

while(node){

if(key == node->key)

return node->value;

else if (key < node -> key)

node = node -> left;

else

node = node->right;l

}

return "Not found";

}

void inorder(TreeNode\* node){

if(node){

inorder(node->left);

cout << node->key << " : " << node->value << endl;

inorder(node -> right);

}

}

int main(){

AVLTree avl;

avl.insert("apple", "A fruit");

avl.insert("banana", "Another fruit");

avl.insert("orange", "Yet another fruit");

avl.insert("apple", "A different meaning for apple");

cout << "In order Traversal " << endl;

avl.displayInOrder();

cout << "\n Search 'orange' :: "<< avl.search("orange") << endl;

avl.remove("banana");

avl.displayInOrder();

}