#include <string>

#include <iostream>

#include <fstream>

using namespace std;

struct PERSONAL

{

string ssn;

string bd;

string fn;

string ln;

string zip;

};

struct Node

{

PERSONAL personal;

Node \*left, \*right;

Node(string s = NULL, string b = NULL, string f = NULL, string l = NULL, string z = NULL, Node \*le = NULL, Node \*ri = NULL)

{

personal.ssn = s;

personal.bd = b;

personal.fn = f;

personal.ln = l;

personal.zip = z;

left = le;

right = ri;

}

};

class Tree

{

protected:

Node \*root;

public:

Tree()

{

root = NULL;

}

void insert(Node \*&t, string s, string b, string f, string l, string z)

{

if (t == NULL)

{

t = new Node(s, b, f, l, z);

return;

}

if (f + l < t->personal.fn + t->personal.ln)

insert(t->left, s, b, f, l, z);

else

insert(t->right, s, b, f, l, z);

}

void del(Node\*& t, string f, string l)

{

if (t == NULL) return;

if (f + l < t->personal.fn + t->personal.ln)

{

del(t->left, f, l);

}

else if (f + l > t->personal.fn + t->personal.ln)

{

del(t->right, f, l);

}

else // found it

{

Node \*tmp = t;

if (t->left == NULL && t->right == NULL)

t = NULL;

else if (t->left == NULL)

t = t->right;

else if (t->right == NULL)

t = t->left;

else

{

tmp = t->right;

while(tmp->left)

{

tmp = tmp->left;

}

t-> personal = tmp->personal;

del(t->right, tmp->personal.fn, tmp->personal.ln);

}

}

}

// void delete\_smallest(Node\*& t)

// {

// if (t != NULL)

// {

// while(NULL != t->left) t = t->left;

// }

//

// if(t->left == NULL) t = t->right;

// else

// {

// delete\_smallest(t->left);

// }

// }

void del(string f, string l)

{

del(root, f, l);

}

void insert\_loop(Node \*&t, string s, string b, string f, string l, string z)

{

if (t == NULL)

{

t = new Node(s, b, f, l, z);

return;

}

Node \*prev = NULL;

while (t != NULL)

{

prev = t;

if (f + l < t->personal.fn + t->personal.ln)

t = t->left;

else

t = t->right;

}

if (f + l < t->personal.fn + t->personal.ln)

prev->left = new Node(s, b, f, l, z);

else

prev->right = new Node(s, b, f, l, z);

}

void insert(string s, string b, string f, string l, string z)

{

insert(root, s, b, f, l, z);

}

void print(Node \*t) // print everything reachable from this node

{

if (t == NULL)

return;

print(t->left);

cout << t->personal.ssn + "\t" << t->personal.bd + "\t" << t->personal.zip + "\t" << t->personal.fn + " " << t->personal.ln + "\t" << endl;

print(t->right);

}

void find\_zip(Node \*t, string z)

{

if (t == NULL)

return;

find\_zip(t->left, z);

if (z == t->personal.zip)

cout << t->personal.ssn + "\t" << t->personal.bd + "\t" << t->personal.zip + "\t" << t->personal.fn + " " << t->personal.ln + "\t" << endl;

find\_zip(t->right, z);

}

void find\_oldest(Node \*t, PERSONAL &personal)

{

if (t == NULL)

return;

find\_oldest(t->left, personal);

if (personal.bd > t->personal.bd)

{

personal.fn = t->personal.fn;

personal.ln = t->personal.ln;

personal.ssn = t->personal.ssn;

personal.bd = t->personal.bd;

personal.zip = t->personal.zip;

}

find\_oldest(t->right, personal);

}

void find\_oldest()

{

if(root == NULL) return;

PERSONAL personal;

personal.zip = root->personal.zip;

personal.bd = root->personal.bd;

personal.ssn = root->personal.ssn;

personal.fn = root->personal.fn;

personal.ln = root->personal.ln;

find\_oldest(root, personal);

cout << personal.ssn + "\t" << personal.bd + "\t" << personal.zip + "\t" << personal.fn + " " << personal.ln + "\t" << endl;

}

void print()

{

print(root);

}

void find\_zip(string zip)

{

find\_zip(root, zip);

}

bool find\_name(string fn, string ln)

{

Node \*curr = root;

cout << fn + " " + ln << endl;

while (curr != NULL)

{

if (curr->personal.fn + curr->personal.ln == fn + ln)

{

cout << curr->personal.ssn + "\t" << curr->personal.bd + "\t" << curr->personal.zip + "\t" << curr->personal.fn + " " << curr->personal.ln + "\t" << endl;

return true;

}

else if (fn + ln < curr->personal.fn + curr->personal.ln)

curr = curr->left;

else

curr = curr->right;

}

return false;

}

};

int main()

{

cout << "Nodeed list database demo" << endl;

ifstream myfile;

myfile.open("/home/218/database.txt");

cout << "open file success" << endl;

//read file

int i = 0;

Tree tree;

string ssn, birth, fname, lname, zipcode;

while (true)

{

if (myfile.fail())

break;

myfile >> ssn >> birth >> fname >> lname >> zipcode;

if (ssn == "")

break;

tree.insert(ssn, birth, fname, lname, zipcode);

}

myfile.close();

cout << "read database success" << endl;

string operation, operation1, operation2;

while (true)

{

cout << "--->operation e.c [f] [p] [z] [o] [d] or [e]" << endl;

cin >> operation;

if (operation == "e")

return 0;

if (operation == "f")

{

cout << "input firstname lastname:";

cin >> operation1;

cin >> operation2;

tree.find\_name(operation1, operation2);

}

if (operation == "p")

{

tree.print();

}

if (operation == "z")

{

cout << "input zipcode:";

cin >> operation1;

tree.find\_zip(operation1);

}

if (operation == "o")

{

tree.find\_oldest();

}

if (operation == "d")

{

cout << "input firstname lastname:";

cin >> operation1;

cin >> operation2;

tree.del(operation1, operation2);

}

}

return 0;

}

