#include <iostream>

#include <string>

#include <iomanip>

#include <vector>

#include <fstream>

using namespace std;

class place {

public:

string state, name;

int pop, inter;

float area, lat, longi, dist;

public:

place() {

name = "DNE";}

place(string st, string na, int po, int inte, float a, float la, float lo, float dis) {

state = st;

name = na;

pop = po;

inter = inte;

area = a;

lat = la;

longi = lo;

dist = dis; }

void printAll() {

char latd = 'N';

char longd = 'E';

//to print only positive values for latitude and longitude

if(lat < 0) {

lat = lat \* -1.0;

latd = 'S'; }

if(longi < 0) {

longi = longi \* -1.0;

longd = 'W'; }

cout << name << ", " << state << endl;

cout << "Population " << pop << endl;

cout << fixed << setprecision(6) << "Area " << area << endl;

cout << "Latitude " << lat << " " << latd << endl;

cout << "Longitude " << longi << " " << longd << endl;

cout << "Intersection " << inter << endl;

cout << setprecision(4) << "Distance " << dist << endl << endl; } };

class hashLink {

public:

place \* data;

hashLink \* next;

public:

hashLink() {

data = NULL;

next = NULL; }

hashLink(place \* da) {

data = da;

next = NULL; }

hashLink(place \* da, hashLink \* ne) {

data = da;

next = ne; }

void add(place \* da) {

if(next == NULL) {

next = new hashLink(da);

return; }

else {

next -> add(da);

return; } } };

class hashList {

public:

vector<hashLink \*> data;

public:

hashList() {

vector<hashLink \*> da(1);

data = da;}

hashList(int n) {

vector<hashLink \*> da(n);

data = da; }

void add(place \* pl) ;

hashLink\* search(string na); };

string simplify(string name) {

string simplified = "";

for(int i = 0; i < name.size(); i++) {

if((name[i]>64)&&(name[i]<91)) {

simplified += name[i]; }

else if((name[i]>96)&&(name[i]<123)) {

simplified += (name[i] - 32); } }

//simplified is made by removing everything non-alphabetical

//from name, and then capitalizing all of the letters

return simplified; }

void printStates(hashLink \* list) {

hashLink \* current = list;

bool first = true;

cout << "Select from ";

while(current -> data != NULL) {

if(first) {

first = false;

cout << current -> data -> state; }

else

cout << ", " << current -> data -> state;

if(current -> next != NULL)

current = current -> next;

else {

cout << ": ";

return; } }

return; }

hashLink \* findAll(hashLink \* link, string na) {

hashLink \* list = new hashLink();

hashLink \* current = link;

hashLink \* DNE = new hashLink(new place());

//Comparing "simplified" strings to make the search

//1)case insensitive

//2)non-alphabetical character insensitive

//ex: Miami Beach == MIAmi beaCh == miamiBEACH

//ex2: Boston-Morea == bostonMorea

string simple = simplify(na);

while(current -> data != NULL) {

if(simplify(current -> data -> name) == simple) {

if(list -> data == NULL)

list = new hashLink(current -> data);

else

list -> add(current -> data); }

if(current -> next == NULL) {

if(list -> data == NULL)

return DNE;

return list; }

current = current -> next; }

return list; }

int hashVal(string name, int vsize) {

//Generates a hash value for each entry

string s = simplify(name);

int val = 888;

for(int i = 0; i < s.size(); i++) {

val = val \* 69 + s[i]; }

if(val < 0)

val = -val;

return val % vsize; }

void hashList::add(place \* pl) {

int val = hashVal(pl -> name, data.size());

if(data[val] == NULL) {

data[val] = new hashLink(pl);

return; }

else {

data[val] -> add(pl);

return; } }

hashLink\* hashList::search(string na) {

int val = hashVal(na, data.size());

//Returns a list with place DNE to signal that this city

//is nonexistent

if(data[val] == NULL) {

return new hashLink(new place()); }

return findAll(data[val], na); }

string rmspaces(string word) {

size\_t end = word.find(" ");

int intend = static\_cast<int>(end);

//Maximum of 1 space between two words in a name

//any double space represents the end of the city name

string word2 = word.substr(0, intend);

return word2; }

int stoi(string s) {

int val = 0;

for(int i = 0; i < s.size(); i++) {

if((s[i] >= 48)&&(s[i] <= 57))

val = val \* 10 + (s[i] - 48); }

return val; }

float stof(string s) {

//starts as double so it can store a large number

//floats are precise to 6 decimal places, so there

//is some error for longitude and longitude

//but floats use half as many bytes as doubles

//the error is plus or minus .000001

//so I will accept the trade off

double val = 0.0;

float divisor = 1.0;

bool neg = false, dec = false;

for(int i = 0; i < s.size(); i++) {

if((s[i] >= 48)&&(s[i] <= 57)) {

if(dec)

divisor = divisor \* 10.0;

//numeric value from ascii math

val = val \* 10 + (s[i] - 48); }

if(s[i] == 46)

dec = true;

if(s[i] == 45)

neg = true; }

float val2 = static\_cast<float>(val) / divisor;

if(neg)

val2 = val2 \* -1.0;

return val2; }

hashList \* readFile(int size) {

string line, st, na;

int pop, inter, numels = 0, size3qtr = (3\*size/4);

float area, lat, longi, dist;

hashList \* theList;

theList = new hashList(size);

place \* pl;

ifstream fin("places.txt");

if(fin.fail()) {

cout << "error\n";

exit(1); }

while(true) {

//To keep the hash table well balanced, we double

//the size of the vector when the number of elements

//stored exceeds 0.75 the size of the table

if(numels >= size3qtr) {

fin.close();

theList = readFile(size\*2);

return theList; }

getline(fin, line);

if(fin.fail()) {

fin.close();

return theList; }

st = line.substr(8,2);

na = rmspaces(line.substr(10,49));

//Largest city population is 8M (New York)

//the int starts at pos 60, and will take at most 7 chars

pop = stoi(static\_cast<string>(line.substr(59,7)));

//substr extraction is easy since the file is somewhat

//well formatted

area = stof(static\_cast<string>(line.substr(66,14)));

lat = stof(static\_cast<string>(line.substr(80,10)));

longi = stof(static\_cast<string>(line.substr(90,11)));

inter = stoi(static\_cast<string>(line.substr(101,5)));

dist = stof(static\_cast<string>(line.substr(106,8)));

pl = new place(st, na, pop, inter, area, lat, longi, dist);

theList -> add(pl);

numels++; } }

void inState(hashLink \* list, string st) {

//puts st in all caps just in case

string state = simplify(st);

hashLink \* current = list;

string city = list -> data -> name;

while(true) {

if(current -> data -> state == state) {

current -> data -> printAll();

return; }

else if(current -> next == NULL) {

cout << "There is no " << city << " in " << st << endl << endl;

return; }

else

current = current -> next; } }

void searchCity(hashList \* list) {

string name, state, trash;

hashLink \* results;

while(true) {

cout << "Please enter a name: ";

getline(cin, name);

results = list -> search(name);

if(results -> data -> name == "DNE") {

cout << name << " does not exist. Perhaps there is a spelling error.\n\n";

continue; }

printStates(results);

cin >> state;

inState(results, state);

getline(cin, trash); } }

int main() {

hashList \* myList;

myList = readFile(10000);

searchCity(myList); }