Assignment #4 – Navigating the country

Nate Paternoster

Code:

#include <string>

#include <fstream>

#include <iostream>

#include <stdio.h>

#include <math.h>

#include <vector>

#include <sstream>

using namespace std;

const int tabsize = 100000;

const double pi = acos(-1.0);

struct position {

double lat, lon;

position(double latdeg, double londeg) {

lat = latdeg \* pi / 180;

lon = londeg \* pi / 180;

}

};

double distance(position from, position to) {

double a1 = sin((from.lat-to.lat) / 2.0);

double a2 = sin((from.lon-to.lon) / 2.0);

double a = a1 \* a1 + cos(to.lat) \* cos(from.lat) \* a2 \* a2;

double d = 7919.2 \* asin(sqrt(a));

return d;

}

double direction(position from, position to) {

double d1 = sin(to.lon-from.lon) \* cos(to.lat);

double d2 = cos(from.lat) \* sin(to.lat) - sin(from.lat) \* cos(to.lat) \* cos(to.lon-from.lon);

return atan2(d1, d2) \* 180.0 / pi;

}

int hash(string s) {

const int init = 21512712, mult = 96169, emergency = 876127;

int v = init;

for (int i=0; i<s.length(); i+=1)

v = v \* mult + s[i];

if (v < 0) v = -v;

if (v < 0) v = emergency;

return v % tabsize;

}

string lowercase(string word) {

for (int i=0; i<word.length(); i++) {

if (word[i] >= 'A' && word[i] <= 'Z') word[i] = word[i] + 32;

}

return word; //returns the string in all lowercase

}

class location {

protected:

int FIPSstate, FIPScensus, population, roadref;

string state, town;

double area, latitude, longitude, roaddist;

public:

location(int, int, string, string, int, double, double, double, int, double);

void print();

string getstate(); //returns the town's original state abbreviation

string getstatelow(); //returns the town's state abbreviation in all lowercase

string gettown(); //returns the town's name in all lowercase

string gettowncaps(); //returns the original town name

int getroad();

};

class Link {

public:

location\* data;

Link\* next;

Link(location\* d, Link\* n = NULL) {

data = d;

next = n;

}

~Link() {

delete data;

next = NULL;

}

void print() {

data->print();

}

};

class List {

protected:

Link\* first;

Link\* last;

int length;

public:

List() {

first = NULL;

last = NULL;

length = 0;

}

~List() {

Link\* prev = NULL;

while (first != NULL) {

prev = first;

first = prev->next;

delete prev;

}

first = NULL;

last = NULL;

length = 0;

}

void add\_to\_front(location\*);

void add\_to\_end(location\*);

Link\* get\_first();

};

class connection {

protected:

string road, type;

double roadlen;

int start, end; //these are just ref numbers, not really start and end

public:

connection(string, string, int, int, double);

string getroad();

int getstart();

int getend();

double getlen();

};

class intersection {

protected:

double longitude, latitude, towndist;

string state, town;

vector<connection\*> connections;

public:

intersection(double, double, double, string, string);

void print();

void add\_connection(connection\*);

vector<connection\*> getconnections();

double gettowndist();

string gettown();

string getstate();

double getlon();

double getlat();

};

Link\* List::get\_first() { return first; } //methods to return protected variables

string location::getstate() { return state; }

string location::getstatelow() { return lowercase(state); }

string location::gettown() { return lowercase(town); }

string location::gettowncaps() { return town; }

int location::getroad() { return roadref; }

string connection::getroad() { return road; }

vector<connection\*> intersection::getconnections() { return connections; }

int connection::getstart() { return start; }

int connection::getend() { return end; }

double connection::getlen() { return roadlen; }

double intersection::gettowndist() { return towndist; }

string intersection::gettown() { return town; }

string intersection::getstate() { return state; }

double intersection::getlon() { return longitude; }

double intersection::getlat() { return latitude; }

void List::add\_to\_front(location\* x) {

first = new Link(x, first);

if (last==NULL) last = first;

length++;

}

void List::add\_to\_end(location\* x) {

Link\* n = new Link(x, NULL);

if (last != NULL) last->next = n;

else first = n;

last = n;

length++;

}

location::location(int FS, int FC, string s, string t, int p, double a, double lat, double lon, int rr, double rd):

FIPSstate(FS), FIPScensus(FC), state(s), town(t), population(p), area(a), latitude(lat), longitude(lon), roadref(rr), roaddist(rd)

{ }

void location::print() {

cout << town << ", " << state << "., pop. " << population << ", area " << area << " sq. mi., " << latitude << "N " << longitude << "W, " << roaddist << " mi. from intersection " << roadref << ".\n";

}

intersection::intersection(double lon, double lat, double dist, string s, string t):

longitude(lon), latitude(lat), towndist(dist), state(s), town(t)

{ }

void intersection::print() {

cout << longitude << ", " << latitude << ". " << towndist << " miles away from " << town << ", " << state;

cout << ". Connects to: ";

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

cout << connections[i]->getroad();

if (i<connections.size()-1) cout << ", ";

}

cout << ".\n";

}

void intersection::add\_connection(connection\* x) {

connections.push\_back(x);

}

connection::connection(string r, string t, int s, int e, double l):

road(r), type(t), start(s), end(e), roadlen(l)

{ }

location\* readline(string &input) {

int pos = 0;

string FIPSs, FIPSc, s, t, pop, a, lat, lon, rr, rd;

location\* output = NULL;

FIPSs = ""; FIPSc = ""; s = ""; t = ""; pop = "";

a = ""; lat = ""; lon = ""; rr = ""; rd = "";

FIPSs = input.substr(0,2);

FIPSc = input.substr(2,5);

s = input.substr(7,2);

for (int i=9; input.substr(i,2)!=" "; i++) { //reading in the town name of variable length

t += input[i]; //building up the string char by char

if (input.substr(i+1,2)==" ") pos = i+1; } //updating the current position

while (input[pos]==' ') pos++; //incrementing to the next piece of data

for (int i=pos; input[i]!=' '; i++) { //reading in the population

pop += input[i];

if (input[i+1]==' ') pos = i+1; } //updating the current position

while (input[pos]==' ') pos++; //incrementing to the next piece of data

for (int i=pos; input[i]!=' '; i++) { //reading in the area

a += input[i];

if (input[i+1]==' ') pos = i+1; } //updating the current position

for (int i=pos+1; input[i]!=' ' && input[i]!='-'; i++) { //reading in the latitude

lat += input[i];

if (input[i+1]==' ' || input[i+1]=='-') pos = i+1; } //updating the current position

while (input[pos]==' ') pos++;

for (int i=pos; i<(input.length()-13); i++) { //reading in the longitude

lon += input[i];

if (i+1==(input.length()-13)) pos = i+1; } //updating the current position

while (input[pos]==' ') pos++;

for (int i=pos; i<(input.length()-8); i++) { //reading in the road reference number

rr += input[i];

if (i+1==(input.length()-8)) pos = i+1; } //updating the current position

while (input[pos]==' ') pos++;

for (int i=pos; i<(input.length()); i++) { //reading in the distance to road

rd += input[i]; }

output = new location(atoi(FIPSs.c\_str()), atoi(FIPSc.c\_str()), s, t, atoi(pop.c\_str()), atof(a.c\_str()), atof(lat.c\_str()), atof(lon.c\_str()), atoi(rr.c\_str()), atof(rd.c\_str()));

return output;

}

void read\_file(string filename, List\*\* &hashtable) {

ifstream f(filename.c\_str());

if (f.fail()) {

cerr << "Failed to open town file\n";

exit(1);

}

int hashval;

location\* temp = NULL;

string input;

while (!f.eof()) {

input = "";

getline(f,input);

if (f.fail()) break;

temp = readline(input);

hashval = hash(temp->gettown()); //hash value is determined with a lowercase town name

if (hashtable[hashval]->get\_first() != NULL) //when that value on the hash table has been taken

hashtable[hashval]->add\_to\_end(temp);

else hashtable[hashval]->add\_to\_front(temp);

}

f.close();

}

void read\_inter(string filename, vector<intersection\*> &data) {

ifstream f(filename.c\_str());

if (f.fail()) {

cerr << "Failed to open intersecion file\n";

exit(1);

}

string lon, lat, dist, state, town;

string input;

lon = ""; lat = ""; dist = ""; state = ""; town = ""; input = "";

while (!f.eof()) {

getline(f,input);

if (f.fail()) break;

istringstream parse(input);

parse >> lon >> lat >> dist >> state;

town = input.substr(30,input.length());

data.push\_back(new intersection(atof(lon.c\_str()), atof(lat.c\_str()), atof(dist.c\_str()), state, town));

}

f.close();

}

void read\_connect(string filename, vector<connection\*> &data, vector<intersection\*> &inter) {

ifstream f(filename.c\_str());

if (f.fail()) {

cerr << "Failed to open connection file\n";

exit(1);

}

string name, type, ref1, ref2, len;

string input;

name = ""; type = ""; ref1 = ""; ref2 = ""; len = ""; input = "";

connection\* temp = NULL;

while (!f.eof()) {

getline(f, input);

if (f.fail()) break;

istringstream parse(input);

parse >> name >> type >> ref1 >> ref2 >> len;

temp = new connection(name, type, atoi(ref1.c\_str()), atoi(ref2.c\_str()), atof(len.c\_str()));

data.push\_back(temp);

inter[atoi(ref1.c\_str())]->add\_connection(temp);

inter[atoi(ref2.c\_str())]->add\_connection(temp);

}

f.close();

}

Link\* search(List\*\* table, string Town) {

string town = lowercase(Town);

int key = hash(town);

if (table[key]->get\_first() == NULL) {

cerr << "Town not found.\n";

return NULL;

}

if (table[key]->get\_first()->data != NULL) {

if (table[key]->get\_first()->next == NULL && table[key]->get\_first()->data->gettown() == town) return table[key]->get\_first();

else if (table[key]->get\_first()->next != NULL) {

Link\* cur = table[key]->get\_first();

int counter = 0;

if (cur->data->gettown() == town) counter++;

while (cur->next != NULL) {

cur = cur->next;

if (cur->data != NULL && cur->data->gettown() == town) counter++; }

cur = table[key]->get\_first();

if (counter>1) { //if the town appears in the list multiples times

cout << cur->next->data->gettowncaps() << " appears in: ";

while (cur != NULL) {

if (cur->data->gettown() == town) cout << cur->data->getstate() << ", ";

cur = cur->next; }

cout << "Which one? ";

string whichstate;

getline(cin, whichstate);

whichstate = lowercase(whichstate); //converting to lowercase for comparison

cur = table[key]->get\_first();

if (cur->data->getstatelow() == whichstate) return cur;

while (cur->next != NULL) {

cur = cur->next;

if (cur->data != NULL && cur->data->getstatelow() == whichstate) return cur; }

cerr << "Invalid state!\n";

return NULL;

}

else if (counter==1) { //if the town appears in the list once

if (cur->data->gettown() == town) return cur;

while (cur->next != NULL) {

cur = cur->next;

if (cur->data !=NULL && cur->data->gettown() == town) return cur; }

cerr << "Town not found.\n";

return NULL;

}

else {

cerr << "Town not found.\n";

return NULL;

}

}

cerr << "Town not found.\n";

return NULL;

}

else {

cerr << "Town not found.\n";

return NULL;

}

}

void userquery(List\*\* table) {

string query = "";

while (true) {

cout << "-- Enter name of town: ";

getline(cin, query);

if (query=="Exit" || query=="exit") exit(1);

Link\* result = search(table, query);

if (result!=NULL) result->print();

}

}

void adventure(List\*\* towns, vector<intersection\*> inters, vector<connection\*> connects) {

cout << "Enter a town to begin your adventure (type 'exit' to exit): ";

string query = "";

getline(cin, query);

if (query=="exit" || query=="Exit") exit(1);

Link\* result = search(towns, query);

while (result==NULL) {

getline(cin, query);

if (query=="exit" || query=="Exit") exit(1);

result = search(towns, query);

}

int num = (result->data->getroad()); //num is the line in the intersection file that we're at

if (inters[num]==NULL) { //do we need this?

cerr << "Town has no intersection!\n";

return;

}

cout << "Your location is " << inters[num]->getlat() << "N " << inters[num]->getlon() << "E. ";

cout << "You are " << inters[num]->gettowndist() << " mi. from" << inters[num]->gettown() << ", " << inters[num]->getstate() << ".\n";

while (true) {

cout << "Choose one of these roads (Type 'new' to start over):\n";

for (int i=1; i<inters[num]->getconnections().size()+1; i++) {

cout << i << ": " << inters[num]->getconnections()[i-1]->getroad() << " (";

cout << inters[num]->getconnections()[i-1]->getlen() << " miles long)\n";

}

string choice = "";

int x;

getline(cin, choice);

if (choice=="exit" || choice=="Exit") exit(1);

if (choice=="new" || choice=="New") adventure(towns,inters,connects);

x = atoi(choice.c\_str()); //atoi returns 0 if non-int value is entered

while (x<1 || x>=inters[num]->getconnections().size()+1 || x==0) {

cerr << "Invalid road\n";

getline(cin, choice);

if (choice=="exit" || choice=="Exit") exit(1);

x = atoi(choice.c\_str());

}

if (inters[num]->getconnections()[x-1]->getstart() == num) //updating to new intersection

num = inters[num]->getconnections()[x-1]->getend();

else num = inters[num]->getconnections()[x-1]->getstart();

cout << "Your location is " << inters[num]->getlat() << "N " << inters[num]->getlon() << "E. ";

cout << "You are " << inters[num]->gettowndist() << " mi. from" << inters[num]->gettown() << ", " << inters[num]->getstate() << ".\n";

}

}

int main() {

List\*\* hashtable = new List\* [tabsize]; //creating the hashtable

for (int i=0; i<tabsize; i++) { //initializing all Lists in table to NULL

hashtable[i] = new List();

}

vector<intersection\*> interarray;

vector<connection\*> connectarray;

read\_file("/home/www/class/een318/named-places.txt", hashtable);

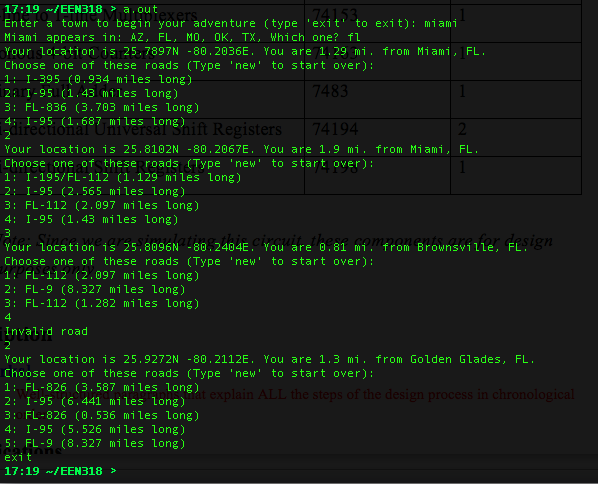
read\_inter("/home/www/class/een318/intersections.txt", interarray);

read\_connect("/home/www/class/een318/connections.txt", connectarray, interarray);

adventure(hashtable, interarray, connectarray);

return 0;

}

Output: