Homework 3 – Hash Tables

Nathan Paternoster

**Code**

#include <string>

#include <fstream>

#include <iostream>

using namespace std;

const int tabsize = 100000;

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;

}

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

};

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();

};

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++;

}

Link\* List::get\_first() {

return first;

}

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";

}

string location::getstate() {

return state;

}

string location::getstatelow() {

return lowercase(state);

}

string location::gettown() {

return lowercase(town);

}

string location::gettowncaps() {

return town;

}

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

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

if (f.fail()) {

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

exit(1);

}

int hashval;

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

location\* temp = NULL;

while (!f.eof()) {

int pos = 0;

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

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

getline(f,input);

if (f.fail()) break;

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];

}

string t\_lower = lowercase(t);

hashval = hash(t\_lower); //hash value is determined with a lowercase town name

temp = 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()));

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();

}

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 << "Search failed.\n";

return NULL;

}

else {

cerr << "Search failed.\n";

return NULL;

}

}

cerr << "Search failed.\n";

return NULL;

}

else {

cerr << "Search failed.\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();

}

}

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();

}

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

userquery(hashtable);

return 0;

}

**Output**

