

Lia Murphy  
C12109232  
9/19/21  
ECE318 Assignment 2

**CODE:**

```
#include <iostream>
#include <fstream>
#include <cmath>
#include <string>
#include <vector>
#include <sstream>

using namespace std;

struct town
{
    int code;
    string state;
    string name;
    int pop;
    float area;
    float latitude;
    float longitude;
    int xRoad; //code for representative road intersection
    float xDist; //distance to intersection
};

struct Link
{
    town t;
    Link* next;

    Link(town _t, Link* n)
    {
        t = _t;
        next = n;
    }
};

class Hash
{
protected:
    int h_size = 10000;
    Link** hashtable;

public:
    unsigned int hashf(string s)
    {
        unsigned int h = 987123654; // replace with my own prime multiplier
        for (int i = 0; i < s.length(); i++)
            h = h * 651 + s[i];
    }
};
```

```

    return h;
}

vector<town> readFile(istream& stream)
{
    string line;
    vector<town> out;

    while (getline(stream, line))
    {
        if (!stream.eof())
        {
            town t;
            istringstream iss;

            string parse;
            string parsed;
            string others;
            string code, state, name, nameOut;

            parse += line;

            for (int i = 0; i < line.size(); i++)
            {
                if (i < 40)
                    parsed += parse[i];
                else
                    others += parse[i];
            }

            iss.str(others);
            iss >> t.pop >> t.area >> t.longitude >> t.latitude >> t.xRoad >> t.xDist;

            for (int i = 0; i < parsed.length(); i++)
            {
                if (i < 8)
                    code += parsed[i];
                else if (i < 10)
                    state += parsed[i];
                else
                {
                    name += parsed[i];
                    //remove(name.begin(), name.end(), ' ');

                    while (true)
                    {
                        if (name.back() == ' ')
                            name.pop_back();
                        else
                        {
                            name = name;
                            break;
                        }
                    }
                }
            }
        }
    }
}

```

```

    }

    /*int spaces = 0;

    for (int i = 0; i < name.length(); i++)
    {
        if (name[i] == ' ')
        {
            spaces++;
            nameOut += name[i];
        }
        if (spaces == 1)
            nameOut = nameOut;
    }*/
}

}

    t.code = std::stoi(code);
    t.state = state;
    t.name = name;
    out.push_back(t);
}
}
return out;
}

void doHash(vector<town> t)
{
    hashtable = new Link * [h_size];
    for (int i = 0; i < h_size; i++)
        hashtable[i] = NULL;

    for (int i = 0; i < t.size(); i++)
    {
        unsigned int h = hashf(t[i].name + t[i].state);
        h = h % h_size;

        Link* ptr = hashtable[h];

        while (ptr != NULL)
        {
            if (ptr->t.name == t[i].name)
                break;
            ptr = ptr->next;
        }
        if (ptr == NULL)
        {
            hashtable[h] = new Link(t[i], hashtable[h]);
            //cout << hashtable[h]->t.name;
        }
    }
}

```

```

    }
}

void printHash(vector<town> t, string _town, string _state)
{
    unsigned int h = hashf(_town + _state);
    h = h % h_size;

    Link * ptr = hashtable[h];

    while (ptr != NULL)
    {
        if (ptr->t.name == _town && ptr->t.state == _state)
            break;
        ptr = ptr->next;
    }

    cout << ptr->t.name
        << " " << ptr->t.state
        << " | code: " << ptr->t.code
        << " | pop: " << ptr->t.pop
        << " | area: " << ptr->t.area
        << " | lat: " << ptr->t.latitude
        << " | long: " << ptr->t.longitude
        << " | nearest xroad code: " << ptr->t.xRoad
        << " | nearest xroad dist: " << ptr->t.xDist << "\n";
}

void findByName(vector<town> t, string _name)
{
    for (int i = 0; i < t.size(); i++)
    {
        unsigned int h = hashf(_name + t[i].state);
        h = h % h_size;
        Link* ptr = hashtable[h];

        while (ptr != NULL)
        {
            if (ptr->t.name == _name)
                break;
            ptr = ptr->next;
        }

        if (t[i].name == _name)
        {
            cout << ptr->t.name
                << " " << ptr->t.state
                << " | code: " << ptr->t.code
                << " | pop: " << ptr->t.pop
                << " | area: " << ptr->t.area
                << " | lat: " << ptr->t.latitude
                << " | long: " << ptr->t.longitude
                << " | nearest xroad code: " << ptr->t.xRoad
                << " | nearest xroad dist: " << ptr->t.xDist << "\n";
        }
    }
}

```

```

    }
}
};

// int main(int argc, char* argv[])
int main()
{
    ifstream f;
    //string path = "C:/Users/Amelia/Documents/town_database.txt";
    string path = "/home/www/class/een318/named-places.txt";
    f.open(path);
    if (f.fail())
    {
        cout << "Could not open file at path " << path << "\n";
        exit(1);
    }

    Hash h;
    vector<town> towns = h.readFile(f);
    h.doHash(towns);

    while (true)
    {
        cout << "Available commands: S (placename) (state), N (placename), Q \n";

        string command, a, b;
        cin >> command >> a >> b;

        if (command == "S")
        {
            //the computer provides all information known for the indicated place
            h.printHash(towns, a, b);
        }
        else if (command == "N")
        {
            //the computer provides all information known for all places with the given name in any state
            h.findByName(towns, a);
        }
        else if (command == "Q")
        {
            //the program stops
            cout << "Shutting down program...";
            exit(1);
        }
        else
        {
            cout << "INVALID COMMAND\n";
        }
    }
}

```

## Proof of it working :)

```
C:\Users\Amelia\source\repos\ECE 318 Lab 2 Hash Table\Debug\ECE 318 Lab 2 Hash Table.exe
Available commands: S (placename) (state), N (placename), Q
S Glencoe IL
Glencoe IL | code: 31729652 | pop: 8762 | area: 3.77709 | lat: -87.761 | long: 42.1316 | nearest xroad code: 9040 | nearest xroad dist: 1.2935
Available commands: S (placename) (state), N (placename), Q
N Miami -
Miami AZ | code: 70446350 | pop: 1936 | area: 0.964143 | lat: -110.872 | long: 33.3962 | nearest xroad code: 3 | nearest xroad dist: 0.0457
Miami FL | code: 31245000 | pop: 362470 | area: 35.673 | lat: -80.2241 | long: 25.7877 | nearest xroad code: 1 | nearest xroad dist: 0.2844
Miami MO | code: 12947684 | pop: 160 | area: 0.561101 | lat: -93.2249 | long: 39.3225 | nearest xroad code: 6 | nearest xroad dist: 0.3902
Miami OK | code: 54048000 | pop: 13704 | area: 9.71397 | lat: -94.876 | long: 36.8835 | nearest xroad code: 0 | nearest xroad dist: 0.2873
Miami TX | code: 64847988 | pop: 588 | area: 1.16667 | lat: -100.639 | long: 35.693 | nearest xroad code: 14 | nearest xroad dist: 0.2798
Available commands: S (placename) (state), N (placename), Q
```

```
main(int argc, char** argv[])
{
    Available commands: S (placename) (state), N (placename), Q
    S Glencoe IL
    Glencoe IL | code: 31729652 | pop: 8762 | area: 3.77709 | lat: -87.761 | long: 42.1316 | nearest xroad code: 9040 | nearest xroad dist: 1.2935
    Available commands: S (placename) (state), N (placename), Q
}
```