# CS205 C/C++ Programming Assignment1

---

**Name:** Guo Yubin

**SID:** 11912726

---

---

## Analysis

> We aren't in this lab interested in the "province or state" information. You are asked to define a structure to hold city and country name, latitude and longitude. You must create an array containing structures as you have defined them, read the file, and and load data into the array.
>
> User will enter city name first (in the first line). Then user enters two floating numbers : latitude and longitude of city(in the second line).

Design a struct, using std::string to store names, double to store latitudes and longitudes.

> Initially set the maximum length for names (city and country name) to 25, and the array size to 800. Your program should issue a warning when data is truncated or not loaded, but it mustn't crash.

Use try and catch, and when exception occurs, generate a int exception_code, throw it, which caught by showException, and then show the message.

> Rename the file to world_cities.tmp. Run your program. It mustn't crash and should display a warning about the missing file.

Same way as above.

> If the city isn't found or if the length of the name is shorter than three letters, a message must be displayed and the user must be prompted for another name.

First, check that size >= 3. Then, use an similarity-EditDistance algorithm to strengthen this program, which will recommend a list of your misspelled query, if your query is not found and also not a substring of any city_name. For example, Beizing -> Beijing.

> Please note that in the file New York appears as "New York City". If people type "New York", then "New York City" must be retrieved. However, if users only type "New" (minimum acceptable length), it can match several cities. The list of the matched cities must be displayed, and the user prompted for the right one.

Use std::string.find to check if it is a substring.

> As before, your program should ignore all whitespaces on both ends of user inputs.

Use std::string.erase and std::string.find_first_not_of.

## Code

Jump To Result & Verification

// 11912726

// Guo Yubin

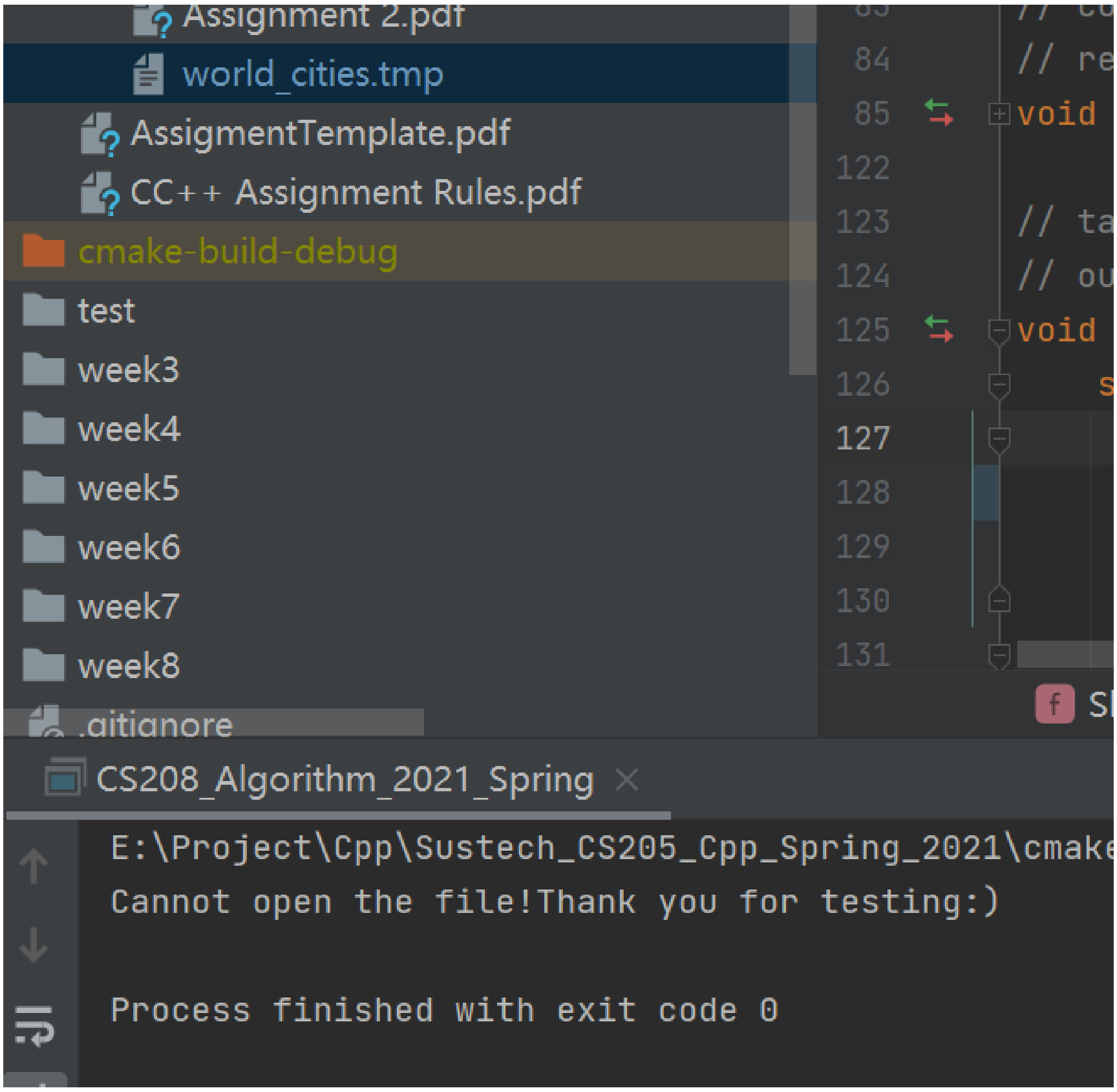
// c++ 17, gcc (x86_64-posix-seh-rev0, Built by MinGW-W64 project) 8.1.0

```
// 11912726
// Guo Yubin
// c++ 17, gcc (x86_64-posix-seh-rev0, Built by MinGW-W64 project) 8.1.0


#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <cmath>
#include <queue>
#include <stack>

#define MAX_NAME_SIZE 46  // 46 is the min value which can read the files in
without exceeding. I don't know why. Everything works well but still a may-be-bug
here.
#define MAX_ARR_SIZE 1000

const int kMaxNameSize = MAX_NAME_SIZE;
const int kMaxArrSize = MAX_ARR_SIZE;
const int kEarthRadius_KM = 6371;
const int kRecommendQueueSize = 3;
const int kMinQuerySize = 3;
const char* kExitCommand= "bye";

struct City{
    std::string city_name;
    std::string province_name;
    std::string country_name;
    double latitude = 0;
    double longitude = 0;
};

class Compare {
public:
    bool operator () (std::pair<City*, double> &a, std::pair<City*, double> &b)
const
    {
        return a.second > b.second;
    }
};


void ReadCities(const std::string& source, std::vector<City>& cities);
void ConvertLineToCity(std::string& line, City& city);
void ShowException(int id);
```

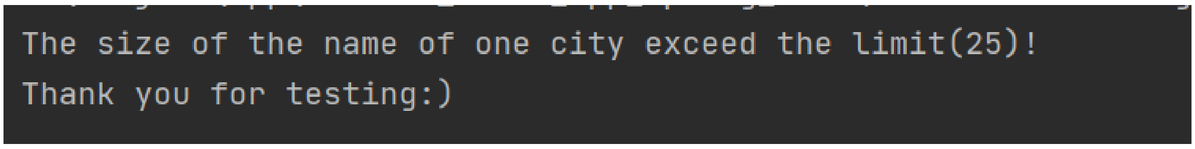
```
double Similarity_EditDistance(std::string str1, std::string str2);
void RecommendQuery(std::priority_queue<std::pair<City*, double>,
std::vector<std::pair<City*, double>>, Compare>& similar_ans);
void CompleteQuery(std::vector<City*>& including_ans);
City *SearchCity(std::vector<City>& cities, const std::string& query);
void UserInterface(std::vector<City> &cities);
double CalculateDistance(City &city1, City &city2);
std::string IgnoreSpace(std::string& str);
std::string StrToLower(std::string str);

int main() {
    std::vector<City> cities;
    try {
        ReadCities("world_cities.tmp", cities);
    } catch (int exception_id) {
        ShowException(exception_id);
    }

    UserInterface(cities);
    std::cout << "Thank you for testing:)\n";
}


// open file and take lines one by one and convert them as cities
// return exception_id
void ReadCities(const std::string& source, std::vector<City>& cities) {
    std::ifstream in_file("world_cities.tmp", std::ios_base::in);
    if (!in_file) {
        throw 10; // exception_id: 11: when read city, cannot open file;
    }
    std::string tmp_line;

    int count = 0;
    while (std::getline(in_file, tmp_line)) {
        if (count >= kMaxArrSize) {
            throw 11; // exception_id: 12: when read city, over max arr size
        }
        City tmp_city;
        ConvertLineToCity(tmp_line, tmp_city);

        ++count;
        cities.push_back(tmp_city);
    }
}

// convert one line in the file(string) to city data
// return int exception_id
void ConvertLineToCity(std::string& line, City& city) {
    int col = 0; // count of column.
    int start = 0; // current index of start char for current column
    int end = 0; // current index of end char for current column
    for (int i = 0; i < line.size(); ++i) {
        if (line[i] == ',' || i == line.size() - 1) {
            if (i == line.size() - 1) {
                ++end;
            }
            if ( col >= 0 && col <=2 && (end - start > kMaxNameSize)) {
                throw 20; // exception_id: 2_1: when converting line to city,
over max name size;
            }
            switch (col) {
                case 0:
                    city.city_name = line.substr(start, end - start);
                    break;
                case 1:
                    city.province_name = line.substr(start, end - start);
                    break;
                case 2:
                    city.country_name = line.substr(start, end - start);
                    break;
                case 3:
                    city.latitude = std::stod(line.substr(start, end - start));
                    break;
                case 4:
                    city.longitude = std::stod(line.substr(start, end - start));
                    break;
                default:
                    break;
            }
            ++col;
            start = i + 1;
        }
        ++end;
    }
}

// take an exception_id as argument
// output the exception message
void ShowException(int id) {
    switch (id) {
        case 10: {
            std::cout << "Cannot open the file!\n";
            break;
        }
        case 11: {
            std::cout << "The amount of cities exceed the limit(" << kMaxArrSize
<< ")!\n";
            break;
        }
        case 20: {
            std::cout << "The size of the name of one city exceed the limit(" <<
kMaxNameSize << ")!\n";
            break;
        }
        default: {
            std::cout << "Unexpected exception, try again or contact us:)\n";
        }
    }
    std::cout << "Thank you for testing:)\n";
    exit(0);
}

// return the similarity of two input string by calculating edit_distance
double Similarity_EditDistance(std::string str1, std::string str2) {
    int len1 = str1.size();
```

```cpp
    int len2 = str2.size();

    int** p_distances = new int*[len1 + 1];
    for (int i = 0; i < len1 + 1; ++i) {
        p_distances[i] = new int[len2 + 1];
        p_distances[i][0] = i;
    }

    for (int i = 0; i < len2 + 1; ++i) {
        p_distances[0][i] = i;
    }

    for (int i = 1; i < len1 + 1; ++i) {
        for (int j = 1; j < len2 + 1; ++j) {
            p_distances[i][j] = std::min(p_distances[i-1][j]+1, p_distances[i]
[j-1]+1);
            p_distances[i][j] = std::min(p_distances[i][j], p_distances[i-1][j-
1]+1 + (str1[i-1]==str2[j-1] ? 0:1));
        }
    }

    double result = p_distances[len1][len2];

    for (int i = 0; i < len1 + 1; ++i) {
        delete[] p_distances[i];
    }
    delete[] p_distances;
    return 1 - result/std::max(len1, len2);
}

void RecommendQuery(std::priority_queue<std::pair<City*, double>,
std::vector<std::pair<City*, double>>, Compare>& similar_ans) {
    std::cout << "City Not Found-!QAQ!-Do you mean these cities?\n";
    std::cout << "And then enter it again\n";
    std::cout << "Copy and paste is strongly recommended for a exact answer!\n";
    std::stack<std::pair<City*, double>> temp_container;

    std::cout << "------------------------------------------\n";
    int n = similar_ans.size();
    for (int i = 0; i < n; ++i) {
        std::cout << similar_ans.top().first->city_name << '\n';
        temp_container.push(similar_ans.top());
        similar_ans.pop();
    }
    std::cout << "------------------------------------------\n";

    for (int i = 0; i < n; ++i) {
        similar_ans.emplace(temp_container.top());
        temp_container.pop();
    }
}

void CompleteQuery(std::vector<City*>& including_ans) {
    std::cout << "Check which one you are looking for.\n";
    std::cout << "And then enter it again\n";
    std::cout << "Copy and paste is strongly recommended for a exact answer!\n";

    std::cout << "------------------------------------------\n";
    for (auto tmp: including_ans) {
        std::cout << tmp->city_name << '\n';
    }
    std::cout << "-------------------------------------------\n";
}

City *SearchCity(std::vector<City>& cities, const std::string& query) {
    if (query.size() < kMinQuerySize) {
        std::cout << "Too short the query is!\n";
        return nullptr;
    }

    std::priority_queue<std::pair<City*, double>, std::vector<std::pair<City*,
double>>, Compare> similar_ans;
    std::vector<City*> including_ans;
    City* p_exact_ans = nullptr;
    for (auto& tmp : cities) {
        std::string tmp_city_name = StrToLower(tmp.city_name);
        if (tmp_city_name == query) {  // if exact answer
            p_exact_ans = &tmp;
            break;
        }
        if (tmp_city_name.find(query) != std::string::npos) { // if contains
query
            including_ans.emplace_back(&tmp);
            continue;
        }

        double tmp_weight = Similarity_EditDistance(query, tmp_city_name);
        if (similar_ans.size() >= kRecommendQueueSize) {
            if (similar_ans.top().second < tmp_weight) {
                similar_ans.pop();
                similar_ans.emplace(&tmp, tmp_weight);
            }
        } else {
            similar_ans.emplace(&tmp, tmp_weight);
        }
    }

    if (p_exact_ans != nullptr) {
        return p_exact_ans;
    } else if (!including_ans.empty()) {
        if (including_ans.size() == 1) {
            return including_ans[0];
        }
        CompleteQuery(including_ans);
    } else {
        RecommendQuery(similar_ans);
    }

    return nullptr;
}

// a command line function for user to search for cities
void UserInterface(std::vector<City> &cities) {
    std::string tmp;
    while (true) {
        using std::cout;
        using std::cin;
        using std::getline;

        City *p_city1 = nullptr;
        City *p_city2 = nullptr;

        cout << "Enter any word to continue, enter \"bye\"(without quotation) to
quit\n";
        getline(cin, tmp);
        IgnoreSpace(tmp);
        if (StrToLower(tmp) == kExitCommand) {
            return;
        }

        while (p_city1 == nullptr) {
            cout << "Enter the name of the first city for searching:\n";
            getline(cin, tmp);
            IgnoreSpace(tmp);
            if (StrToLower(tmp) == kExitCommand) {
                return;
            }
            p_city1 = SearchCity(cities, StrToLower(tmp));
        }

        while (p_city2 == nullptr) {
            cout << "Enter the name of the Second city for searching:\n";
            getline(cin, tmp);
            IgnoreSpace(tmp);
            if (StrToLower(tmp) == kExitCommand) {
                return;
            }
            p_city2 = SearchCity(cities, StrToLower(tmp));
        }

        std::cout << "The distance is: " << CalculateDistance(*p_city1,
*p_city2) << " (km)\n";
    }
}

double CalculateDistance(City &city1, City &city2) {

//    if (city1.latitude < -90 || city1.latitude > 90 || city1.longitude < -180
|| city1.longitude > 180) {
//        std::cout << "Check your location data! (0 <= phi <= 180, -180 <= theta
<= 180)\n";
//    }
//    if (city2.latitude < -90 || city2.latitude > 90 || city2.longitude < -180
|| city2.longitude > 180) {
//        std::cout << "Check your location data! (0 <= phi <= 180, -180 <= theta
<= 180)\n";
//    }

    double phi1 = (90 - city1.latitude) * M_PI / 180;
    double phi2 = (90 - city2.latitude) * M_PI / 180;
    double theta1 = (city1.longitude) * M_PI / 180;
    double theta2 = (city2.longitude) * M_PI / 180;

    return kEarthRadius_KM * acos(sin(phi1) * sin(phi2)
             * cos(theta1 - theta2)
             + cos(phi1) * cos(phi2));
}

std::string IgnoreSpace(std::string& str){
    str.erase(0, str.find_first_not_of(" \t")); // delete spaces in the front
    str.erase(str.find_last_not_of(" \t") + 1); // delete spaces behind
    return str;
}

std::string StrToLower(std::string str) {
    for (int i = 0; i < str.size(); ++i) {
        str[i] = tolower(str[i]);
    }
    return str;
}
```

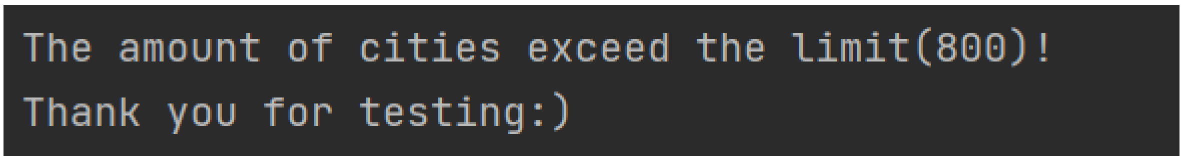
## Result&Verification



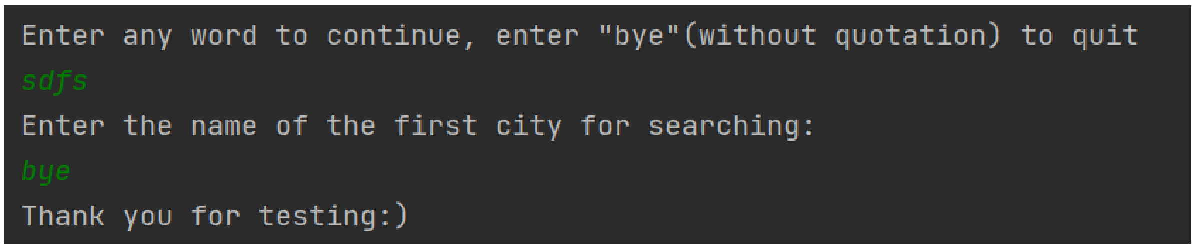
```
E:\Project\Cpp\Sustech_CS205_Cpp_Spring_2021\cmake
Cannot open the file!Thank you for testing:)

Process finished with exit code 0
```



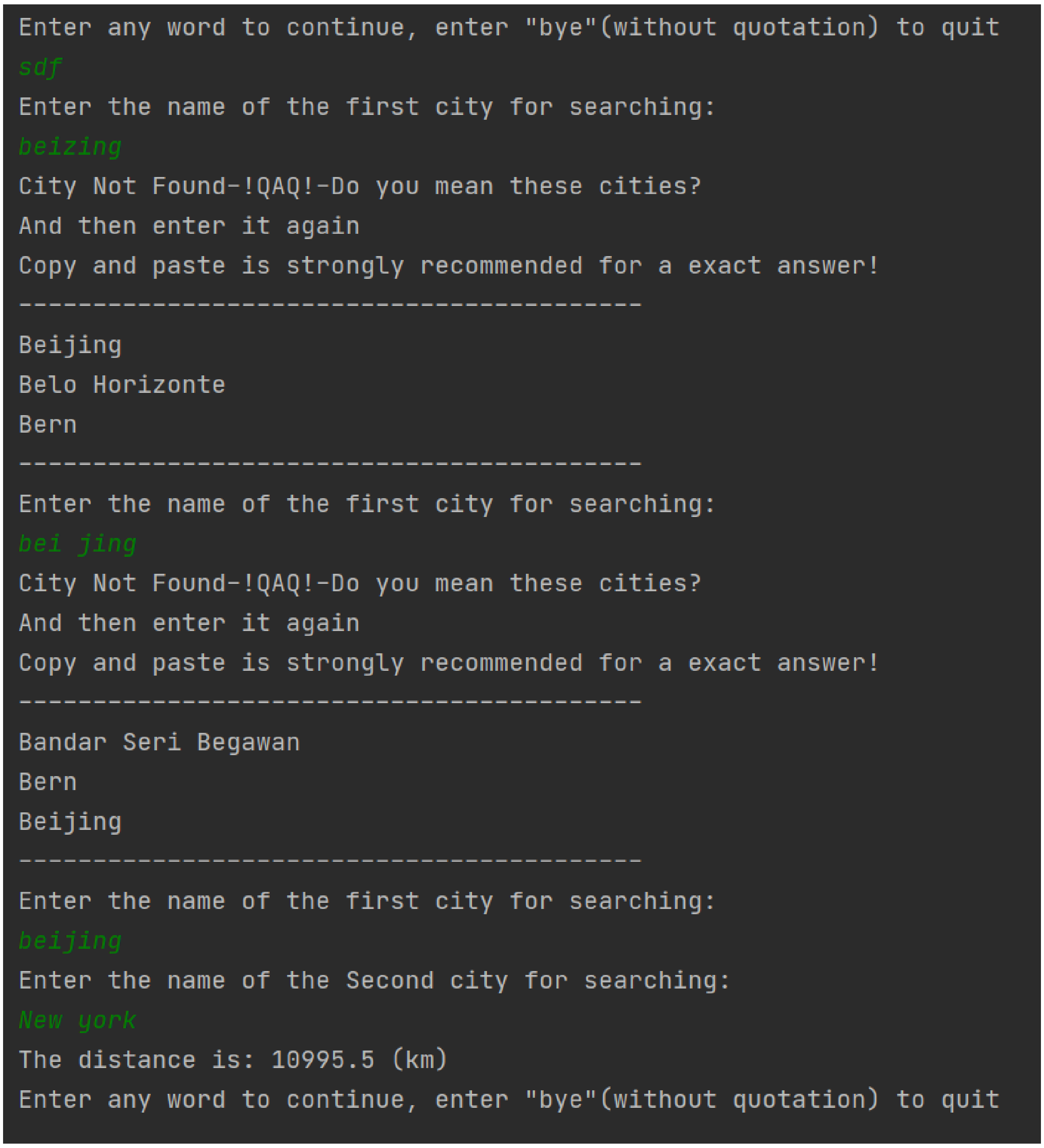
```
The size of the name of one city exceed the limit(25)!
Thank you for testing:)
```



```
The amount of cities exceed the limit(800)!
Thank you for testing:)
```



```
Enter any word to continue, enter "bye"(without quotation) to quit
sdfs
Enter the name of the first city for searching:
bye
Thank you for testing:)
```



```
Enter any word to continue, enter "bye"(without quotation) to quit
sdf
Enter the name of the first city for searching:
beizing
City Not Found-!QAQ!-Do you mean these cities?
And then enter it again
Copy and paste is strongly recommended for a exact answer!
------------------------------------------
Beijing
Belo Horizonte
Bern
------------------------------------------
Enter the name of the first city for searching:
bei jing
City Not Found-!QAQ!-Do you mean these cities?
And then enter it again
Copy and paste is strongly recommended for a exact answer!
------------------------------------------
Bandar Seri Begawan
Bern
Beijing
------------------------------------------
Enter the name of the first city for searching:
beijing
Enter the name of the Second city for searching:
New york
The distance is: 10995.5 (km)
Enter any word to continue, enter "bye"(without quotation) to quit
```

## Difficulties&Solutions

```
Difficulties: 46 is the min value for MAX_NAME_SIZE which can read the files in
without exceeding. I don't know why. Everything works well but still a may-be-bug
here.

Solutions: to be continue.
```