CPSC 4520 Distributed Systems

Programming Assignment #2: A Three-Tiered Airport Lookup System

1. Preface

- This is a group project. You can form a group with a max size of 3 students. Do not recommend it as an individual project though you can do so.
- Submission
  - One submission per group
  - Two submissions & two deadlines
    - Design document submission via Canvas, due 11:59pm, Sunday, 10/11/2020
    - Implementation source code submission via cs1.seattleu.edu, due 11:59pm, Sunday, 10/25/2020
- Plan ahead
  - We may have reading and written assignments overlapping with this project.
- Debugging & Testing
  - Start early, and perform unit tests on each module
  - Want to discuss with me during office hours and via emails?
    - Please do not simply share your screen to show the problems, but you have no idea what is going with your program.
    - Please do not simply attach the source code or screenshots in your emails but have no idea what is going with your program.
    - I need specific questions. You are required to come up with specific questions, rather than just show me code and expect me to find bugs for you. This makes unit tests imperative.

2. Project description

The project aims to design and implement a three-tiered client-server system to serve a user's airport lookups by using RPCs. The system consists of three components: the client, the places server and the airports server.

Given a client's query, the system is required to find the five nearest airports for the specified location. The data flow looks like:

1) The client contacts the places server with a location (city, state)
2) The places server searches its local database for the location and gets its *latitude* and *longitude*.
3) The places server then contacts the airports server using the latitude and longitude.
4) The airport server returns the five nearest airports to the places server.
5) The places server returns the five nearest airports to the client.
6) The client prints each airport code, name, and distance.

2.1 The client

The client program takes three command-line arguments[1]: The first argument will always be the places server's machine name. The second will be a string specifying the city name. The third will be a string specifying the state name.

Run the client program to look up the five nearest airports for the specified city and state:

./client places_server_host city state

E.g.,

client localhost Princeton NJ

client localhost "New Brunswick" NJ

Note the use of quotes to ensure that the shell treats New Brunswick as one argument.

The client program is expected to print out the five nearest airports' information based on the received response message, as shown below:

./***client localhost*** "Princeton borough" NJ
Princeton borough, NJ: 40.352206, -74.657071
code=TTN, name=Trenton, state=NJ distance: 10 miles
code=WRI, name=Mcguire AFB, state=NJ distance: 23 miles
code=PNE, name=North Philadelphia, state=PA distance: 27 miles
code=NXX, name=Willow Grove, state=PA distance: 28 miles
code=NEL, name=Lakehurst, state=NJ distance: 28 miles

2.2 The places server

The *places server* accepts two arguments: the name of the place and a two-letter abbreviation for the state. It searches its local database to find:

- full place name
- state
- latitude
- longitude

Be sure to return the latitude and longitude not as strings but of type float or double.

An error code is returned if the name is not found and the client program should display a proper error message.

[1] Please refer to http://www.site.uottawa.ca/~lucia/courses/2131-05/labs/Lab3/CommandLineArguments.html for command-line arguments.

Otherwise, the places server contacts the airports server using the latitude and longitude to retrieve the five nearest airports and returns the airport information to the client.

Upon startup, the server reads in the file places2k.txt (extract it from places.zip), a list of place names compiled by the U.S. Census. The places file contains data for all Incorporated and Census Designated places in the 50 states, the District of Columbia and Puerto Rico as of the January 1, 2000. The file is plain ASCII text, one line per record:

columns	meaning
1-2	United States Postal Service State Abbreviation
3-4	State Federal Information Processing Standard (FIPS) code
5-9	Place FIPS Code
10-73	Name
74-82	Total Population (2000)
83-91	Total Housing Units (2000)
92-105	Land Area (square meters) - Created for statistical purposes only.
106-119	Water Area(square meters) - Created for statistical purposes only.
120-131	Land Area (square miles) - Created for statistical purposes only.
132-143	Water Area (square miles) - Created for statistical purposes only.
144-153	Latitude (decimal degrees) First character is blank or "-" denoting North or South latitude respectively
154-164	Longitude (decimal degrees) First character is blank or "-" denoting East or West longitude respectively

2.3 The airport server

The *airport server* accepts two arguments: the latitude and longitude that were sent from the places server. It returns the five closest airports to the places server, with each structure containing:

- Airport code
- Airport name

- Airport's latitude (float or double)
- Airport's longitude (float or double)
- Distance from search city (miles)

Upon startup, the server reads in the file airport-locations.txt (extract it from places.zip), which contains a list of 1,065 airport locations in the U.S. The file is plain ASCII text, one line per record but includes blank lines and a header on the first line (Your code should skip those lines). Its format is:

[*airport_code*] *latitude longitude city*

3. Design issues

In this assignment, performance and usability are our design goals. For performance, the client's query should be answered in a timely manner. To this end, you need to use *good data structures and algorithms* to make the search process fast. Simply using arrays/linked-lists and linear search won't be a good design. Points will be deducted for the naïve design choices.

Usability can be interpreted as follows.

For a normal query submitted by the client: ./client localhost "Princeton borough" NJ

The client can also submit his/her query like this: ./client localhost "Princeton bor" NJ
or even like this: ./client localhost Princeton NJ

In short, usability means that the system should support the longest prefix matching queries. In the presence of ambiguity (i.e., two or more places have the same prefix), the client should ask the client to refine its query. Also, it makes sense if you make the prefix matching case insensitive. If your program does not support this usability requirement, points will be deducted!

4. Implementation hints

4.1 File access

The available file operations that you may use in this assignment:

- fopen(): open a file
- fgets(): read a line from the file
- fclose(): close a file

In C++, you may use fstream library.

4.2 Reading places2k.txt

As mentioned above, fgets() reads a line into a memory buffer (C string). Since the data in the file occupy fixed fields, each datum can be extracted via the *strncpy()* method. You also can use *atoi()*, *atof()* to convert a string to int and float/double.

4.3 Reading airport-locations.txt

To weed out lines that don't contain airport info, you can just skip any lines that don't have a comma; it will save checking if a line is blank or is a header. The airport code, latitude, and longitude occupy fixed columns: you can extract that data with *strncpy()*. The airport name is everything between a tab and a comma. The state is everything after the comma. You can use *strchr()* with the comma to split the airport name and state.

4.4 Calculating distance

You can approximate the distance between two points by using spherical trigonometry and calculating the great circle distance. The distance in nautical miles between two points is:

$$d = 60 \cos^{-1}(\sin(lat_1) \sin(lat_2) + \cos(lat_1) \cos(lat_2) \cos(lon_2 - lon_1))$$

Where *lat* is latitude and *lon* is longitude. Both are expressed in degrees.

Finally, you'll need to convert the result from nautical miles to statute miles by multiplying by 1.1507794 (1 nautical mile = 1.1507794 statute miles).

Here, I provide the functions for you to calculate distance to ease your implementation.

```
/*::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::*/
/*::                                                                          :*/
/*:: This routine calculates the distance between two points (given the     :*/
/*:: latitude/longitude of those points). It is being used to calculate     :*/
/*:: the distance between two ZIP Codes or Postal Codes using our         :*/
/*:: ZIPCodeWorld(TM) and PostalCodeWorld(TM) products.                   :*/
/*::                                                                          :*/
/*:: Definitions:                                                  :*/
/*::   South latitudes are negative, east longitudes are positive         :*/
/*::                                                                          :*/
/*:: Passed to function:                                                :*/
/*::   lat1, lon1 = Latitude and Longitude of point 1 (in decimal degrees)  :*/
/*::   lat2, lon2 = Latitude and Longitude of point 2 (in decimal degrees)  :*/
/*::   unit = the unit you desire for results                           :*/
/*::          where: 'M' is statute miles                           :*/
/*::                 'K' is kilometers (default)                    :*/
/*::                 'N' is nautical miles                          :*/
/*:: United States ZIP Code/ Canadian Postal Code databases with latitude & :*/
/*:: longitude are available at http://www.zipcodeworld.com               :*/
/*::                                                                          :*/
/*:: For enquiries, please contact sales@zipcodeworld.com                 :*/
```

```
/*::                                                                         :*/
/*::  Official Web site: http://www.zipcodeworld.com                       :*/
/*::                                                                         :*/
/*::  Hexa Software Development Center © All Rights Reserved 2004            :*/
/*::                                                                         :*/
/*::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::*/


#include <math.h>

#define pi 3.14159265358979323846

double distance(double lat1, double lon1, double lat2, double lon2, char unit)
{
  double theta, dist;
  theta = lon1 - lon2;
  dist = sin(deg2rad(lat1)) * sin(deg2rad(lat2)) + cos(deg2rad(lat1)) *
         cos(deg2rad(lat2)) * cos(deg2rad(theta));
  dist = acos(dist);
  dist = rad2deg(dist);
  dist = dist * 60 * 1.1515;
  switch(unit) {
    case 'M':
      break;
    case 'K':
      dist = dist * 1.609344;
      break;
    case 'N':
      dist = dist * 0.8684;
      break;
  }
  return (dist);
}

/*::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::*/
/*::  This function converts decimal degrees to radians                    :*/
/*::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::*/
double deg2rad(double deg) {
  return (deg * pi / 180);
}

/*::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::*/
/*::  This function converts radians to decimal degrees                    :*/
/*::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::*/
double rad2deg(double rad) {
  return (rad * 180 / pi);
}
```

5. References

1) https://linux.die.net/man/3/rpc RPC library routines
2) https://cs.nyu.edu/courses/spring00/V22.0480-002/class07.html search “unions” and “xdr_free” on this page for details about union structure and xdr_free() in RPC

6. Submission

Only one submission is required for each group.

This project requires two submissions, as described in the following table.

Submission	Deadline	What to submit
Design document via Canvas	11:59PM, Sunday, 10/11/2020	A 1-2 page design document including: • Team members • IDL programs • Data structures and algorithms used to meet performance and usability requirements • If you use a third-party code/library, please specify it and give the reference If you expect me to give you feedbacks earlier, please send me an email notification after submitting it. I'll try my best to give you the comments as early as possible.
Source code via cs1	11:59PM, Sunday, 10/25/2020	Include: • Readme: It must be a text file, including: ○ team members and their respective contributions ○ data structures and algorithms used for meeting performance and usability requirements ○ third-party code (name and reference, if any) ○ strengths and weakness (be truthful!) ○ instructions to run your programs. ○ No .pdf, .doc, or .docx is accepted! • *.x: all IDL programs • *.h: all .h files • *c: all .c files • Makefile (only one Makefile) • Third-party code (if any) Step 1: create a p2.tar that includes all the aforementioned files Step 2: run the submission script on cs1: /home/fac/zhuy/zhuy/class/SubmitHW/submit4520 p2 p2.tar

If submission succeeds, you will see the message similar to the following one on your screen:

```
[testzhuy@cs1 cs1old]$ /home/fac/zhuy/zhuy/class/SubmitHW/submit4520    p2   p2.tar

======================Copyright(C)Yingwu Zhu==========================
Fri Sep 11 14:35:20 PDT 2020
Welcome testzhuy!
You are submitting pointers.cpp for assignment p2.
```

Transferring file.....
Congrats! You have successfully submitted your assigment! Thank you!
Email: zhuy@seattleu.edu

==========

You can submit your assignment multiple times before the deadline. Only the most recent copy is kept.

The assignment submission will shut down automatically once the deadline passes. You need to contact me for instructions on your late submission. **Do NOT email me your submission!**

7. Grading Criteria

Table 1 Design Document Grading Criteria

Label	Notes
2a. Team member info (1 pt)	Include a complete team member list and what they will be working on respectively.
2b. IDL programs (2 pts)	IDL program blocks and related data structures.
2c. Data structures & algorithms to use (1 pt)	The data structures and algorithms you plan to use for performance and usability requirements. Briefly describe why they can improve performance and usability respectively. If you plan to use a third-party code, please include the references.
2d. Format (1 pt)	The document is well-written, easy-to-follow and well formatted.

Table 2 Source Code Grading Criteria

Label	Notes
2a. Submission (1 pt)	All required files are included. Makefile works properly to generate client and server executables.
2b. Format & Style (1 pt)	Clean, well-commented code. No messy output/debugging messages. Program execution must follow the specification.
2c. Readme file (2 pts)	The Readme file is documented as required. No .pdf, .doc, or .docx is accepted! It is readable on cs1.
2d. Functionality (10 pts)	The 3-tiered system behaves as specified. • The client displays the five nearest airports with code, name, state and distance information • The client asks the user to refine the query if a query causes ambiguity • The client displays error messages if the RPC call fails. • The client displays proper messages if the location is not found.
2e. Performance &	Proper data structures and efficient algorithms are used to improve performance and usability. Simply using arrays/linked-lists/vectors & linear search gets zero

usability (5 pts)	point here.
2f. Avoid memory leak (1 pt)	Use xdr_free() and/or clnt_freeres() properly to release RPC resources. Avoid other memory leaks.
2g. Overriding policy	If the code cannot be compiled or executed (immediate segmentation faults, for instance), it results in zero point. If the submission is incomplete (e.g., missing files) for execution, it results in zero point.
2h. Late submission	Please refer to the late submission policy on Syllabus.

8. Testing Cases

Below are provided two testing cases to help you debug & test your program. No worries if the distance decimal numbers are slightly different those shown here.

[zhuy@cs1 p1]$ ./places_client localhost Seattle WA

seattle city, wa: 47.626354, -122.333145

code=BFI, name=Seattle/Boeing, state=WA, distance:6.88 miles

code=RNT, name=Renton, state=WA, distance:10.20 miles

code=SEA, name=Seattle, state=WA, distance:12.34 miles

code=PAE, name=Everet/Paine, state=WA, distance:20.43 miles

code=PWT, name=Bremerton, state=WA, distance:22.76 miles

[zhuy@cs1 p1]$ ./places_client localhost "New York" NY

new york city, ny: 40.704235, -73.917931

code=LGA, name=New York/La Guardia, state=NY, distance:4.61 miles

code=JFK, name=New York/JFK, state=NY, distance:8.16 miles

code=TEB, name=Teterboro, state=NJ, distance:12.19 miles

code=EWR, name=Newark Liberty International, state=NJ, distance:13.21 miles

code=CDW, name=Fairfield, state=NJ, distance:22.10 miles