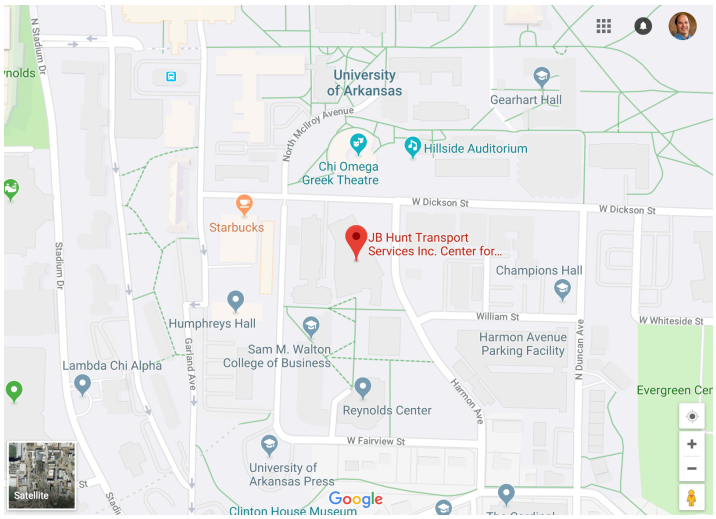
CSCE 2014 – Programming Project 3

Midpoint Due Date – 09/26/2022 at 11:59pm
Final Due Date – 10/03/2022 at 11:59pm

1. Problem Statement:

Geographic information system (GIS) applications have literally taken over the world in recent years. One of the most popular is Google Maps, which lets users view maps centered on specific addresses, with roads and nearby sites of interest displayed. Behind the scenes Google has created some amazing algorithms and data structures to support millions of users displaying maps simultaneously.



The goal of this programming project is to develop a program that reads and processes GIS information to answer two questions (1) what is the location (latitude, longitude) of a specific street address? (2) what addresses are within distance D of a specified location? City planners or business owners could use this GIS tool to help plan where to locate new schools, or build the next fast-food restaurant. This project has four tasks described below.

Task 1: Read and store GIS information

The data we will be working with for this project consists of 33,000 synthetic street addresses in the Fayetteville area. The input files will contain one address record per line that contains the following four data fields in this order:

- Latitude – a float value between 36.03 and 36.15
- Longitude – a float value between -94.08 and -94.20
- House number – an integer value between 1 and 250
- Street name – a string containing one or more words

In order to store address information above, create a container class called "Address" that has private variables for the four fields above. This class should implement a default constructor, a non-default constructor, a copy constructor, an empty destructor function, four get methods, four set methods, and a print method.

In your main program you should implement a "read_file" function that opens and reads an input file and stores this GIS data in a vector of Address objects. The two advantages of using a vector are: 1) we do not need to know how many address records there are in the input file when we write the program, and 2) we can use traditional array indexing to search for desired address fields in the vector.

Task 2: Use recursive binary search to find an address

The input data file "address.txt" has been sorted by street address and then street number. You should call the "read_file" function to read this file and store this Address data in a vector called "address".

In your main program you should implement a "search_address" function that uses recursive binary search on the "address" vector to lookup a specific street address to find the Address record containing the latitude and longitude of this address. If there is no exact match for the street number, you should return the closest address (i.e. the last address you looked at).

If your program was somehow integrated into Google Maps, you could search for "227 N Harmon Ave" to find the (latitude, longitude) of the JB Hunt building and plot the red icon at the correct location on the image above.

Task 3: Use iterative binary search to find nearby locations

The input data file "location.txt" has been sorted by latitude and then longitude. You should call the "read_file" function to read this file and store this Address data in a vector called "location".

In your main program you should implement a "search_location" function that uses iterative binary search to print all addresses that are within a specified distance D of location (A,B). To do this, you should search the "location" vector to find the address at location (A-D, B-D) and then perform a linear scan of the vector to print all addresses where the distance to (A,B) is less than D. You should stop this linear scan when you reach the point (A+D, B+D) because all remaining addresses will be outside the specified distance.

If your program was somehow integrated into Google Maps, you could plot icons on the image for all addresses that within one mile of your favorite school or fast-food restaurant.

Task 4: Implement a main program to perform two searches

Your main program should read both of the input files “address.txt” and “location.txt” into two vectors and then perform a number of calls to the two binary search functions above to demonstrate their correctness. Your program should have a simple user interface that allows the user to select which search operation to perform and enter the necessary search parameters (either a street address, or a latitude, longitude value and distance).

2. Design:

Your first design task is to create a C++ class called "Address" that contains all of the private data fields and methods listed above. You can base this class on any of our previous class examples. Be sure to choose sensible names for all variables and methods.

Your second task is to work out the logic for how to read address data from an input file and store it in a vector of Address objects. The goal is to store the data in the vector in the same order that it appears in the input file. There are several ways to do this using the constructor functions and get/set methods you created above.

Your third task is to design the recursive binary search by address. One tricky aspect of this process is that you will need to combine house number and street name when comparing Address objects. For example, “20 Maple Street” is less than “10 Oak Drive” because “M” is alphabetically before “O”. When the street names match exactly, we should compare house numbers numerically, so “7 Walnut Road” is less than “42 Walnut Road”.

Your fourth task is to design the iterative binary search by location. The tricky part of this search process is that we want Address records where the distance between the input (A,B) and nearby (latitude, longitude) is less than a specified value D. To do this, you need to look at a range of address records that lie between (A-D, B-D) and (A+D, B+D). There is a complicated formula for calculating the exact distance between two (latitude, longitude) values, but for our purposes, the Euclidean distance should be close enough.

3. Implementation:

To implement your project, you should break your code down into **three** files: address.h, address.cpp, and main.cpp. You are strongly encouraged to look at programs on the class website for sample code to assist in the implementation of this project.

As always, it would be a good idea to start with "skeleton methods" to get something to compile, and then add the desired code to each method incrementally writing comments, adding code, compiling, debugging, a little bit at a time. Once you have

the methods implemented, you can create a main program with s simple menu interface that calls these methods to complete your project.

Remember to use good programming style when creating your program (good names for variables and constants, proper indenting for loops and conditionals, clear comments). Be sure to save backup copies of your program somewhere safe. Otherwise, you may end up retyping your whole program if something goes wrong.

4. Testing:

For this project students are required to compile and test their program on Linux. Once you have logged into turing.csce.uark.edu, copy your source code into a folder called "project3". Go into this folder and type in the command "script". This will create a "typescript" file that records all user input and program output until you type "exit".

Compile your program using "g++ -Wall *.cpp -o main.exe". Then run your program by typing "./main.exe". Enter a sequence of commands to verify that your program operates correctly for all of the requirements listed above. Also check for the error handling capabilities of your code.

When you are finished testing your program, type "exit". Copy the "typescript file back to your personal computer to be included with your code and project report when you submit your completed project into Blackboard.

5. Documentation:

When you have completed your C++ program, write a short report using the project report template describing what the objectives were, what you did, and the status of the program. Does it work properly for all test cases? Are there any known problems? Save this report to be submitted electronically.

6. Midpoint Submission:

For the midpoint submission, you should upload your code after you are approximately 1/3 to 1/2 way through this implementation process. It is OK to have skeleton methods that simply print debugging messages at this stage. The main thing we want to see is code that compiles and is part of your final solution.

7. Project Submission:

In this class, we will be using electronic project submission to make sure that all students hand their programming projects and labs on time, and to perform automatic plagiarism analysis of all programs that are submitted.

When you have completed the tasks above, copy all of your source code, your typescript file, and your project report into a folder called “project2”. Compress this directory into a single ZIP file called “project2.zip”, and upload this ZIP file into Blackboard. The GTAs will download and unzip your ZIP file and compile your code using “g++ -Wall *.cpp” and run your program to verify correctness.

The dates on your electronic submission will be used to verify that you met the due date above. All late projects will receive reduced credit:

- 10% off if less than 1 day late,
- 20% off if less than 2 days late,
- 30% off if less than 3 days late,
- no credit if more than 3 days late.

You will receive partial credit for all programs that compile even if they do not meet all program requirements, so handing projects in on time is highly recommended.

8. Academic Honesty Statement:

Students are expected to submit their own work on all programming projects, unless group projects have been explicitly assigned. Students are **not** allowed to distribute code to each other, or copy code from another individual or website. Students **are** allowed to use any materials on the class website, or in the textbook, or ask the instructor and/or GTAs for assistance. This course will be using highly effective program comparison software to calculate the similarity of all programs to each other, and to homework assignments from previous semesters. Please do not be tempted to plagiarize from another student.

Violations of the policies above will be reported to the Provost's office and may result in a zero on the programming project, an XF in the class, or suspension from the university, depending on the severity of the violation and any history of prior violations.