Classy GIS — C++ `class` types, Separate Compilation

This assignment requires modifying the design and implementation of the Simple GIS to incorporate C++ classes and separate compilation.

A good design for the Simple GIS can easily be turned into an acceptable design for this project. You should implement at least two, and probably three, classes in your design. To encourage good use of classes, you are expressly forbidden to use any `struct` types in this assignment.

When thinking about your design, it is important to assign each of your classes the correct responsibilities. Give careful thought to the importance of information hiding, especially with respect to the relationship between the controller code and the GIS class. (Yes, that's a strong hint.)

Input Data

There is no change to the input data specification given for the Simple GIS project.

System Initialization

System initialization proceeds as in the Simple GIS project.

Script File

The script file will have the same format as before. The only change is the addition of one additional command:

`distance``<tab><city name><tab><state abbrev><tab><city name><tab><state abbrev>`
If the two cities are in the GIS and are neighbors, log the distance between them. If one or both are not in the GIS, log an error message to that effect. If both are in the GIS but are not neighbors, log an error message to that effect.

Each command given in the script file will be syntactically correct, according to the specifications above. Some will raise logical issues, described in the previous section. The system should stop processing the script gracefully if it runs out of commands without finding an explicit `exit` command.

Legend:

`<city name>`	string, not containing a tab
`<state abbreviation>`	string, not containing a tab
`<limit>`	non-negative integer value

There is no guaranteed constraint on the length of city names, so your implementation should handle storing city names of arbitrary length. However, for formatting purposes, you may assume that no city name will be longer than 25 characters.

Log

Aside from the additional command above, there are no changes to the log file specification.

Implementation Constraints

This specification imposes a number of limitations on how you are allowed to implement your solution. Some of these constraints may appear to be unjustified. They are not. Regardless, failure to conform to them will certainly result in a grade penalty on the project.

- You are forbidden to use anything that is not ISO Standard C++. The course notes present only Standard C++. If you are not sure about something, ask. In particular, you may not use any of the non-Standard abominations

commonly presented in the AP Computer Science course. These include, but are not necessarily limited to, `apstring` and `apvector`.

- You are forbidden to explicitly use any dynamic allocation in this assignment (e.g., `new`, `delete`, `malloc`). Dynamic allocation is covered later in the course, and one of the later assignments will require you to modify this implementation to incorporate it.
- You are forbidden to use any STL containers, except `string`, on this assignment. In view of the restrictions given above, you are thus limited to using statically-allocated arrays for storing lists. This will impose some functional limitations on your solution; that's OK.
- You are required to write classes. You are explicitly forbidden to use `struct` types.
- You must implement your design using separate compilation. In particular, you must have a separate header file and cpp file for each class you write, and at least one cpp file for the driver code that uses those classes.
- Without exception, all class data members must be private.

There are also some constraints on the logical functionality of your solution, imposed in order to guarantee that there is only one correct log for each set of input data.

When adding a new city the GIS will append it to the end of the list; this means that cities are not stored in a optimal order as far as searching is concerned. We will deal with that issue in the next project.

When adding a new route to the GIS, you must add new neighbor entries to two city records. Neighbor entries are to be stored in the order they are added, so new entries always go at the end of the city record's neighbor list.

When removing a city, the GIS will not physically remove the entry from its list. This is for simplicity; physically removing a city record would require shifting the contents of the list, and that would require updating many of the neighbor entries to account for the shifting. While that is possible, it is excessively expensive. So, removing a city will involve marking the record in some special way to indicate that it is unused. That could result in significant wasted space over time. We may also deal with that issue in a later project.

Removing a route does not raise the same issues, so there is nothing special there; you will simply delete neighbor entries from the relevant neighbor lists.

Evaluation

Do not waste submissions to the Curator to test your program! There is no point in submitting your program until you have verified that it produces correct results on the sample data files that will be provided. If you waste all of your submissions because you have not tested your program adequately then you will receive a low score on this assignment. You will not be given extra submissions.

Your submitted program will be assigned a score, out of 200, based upon the runtime testing performed by the Curator System. The TAs will also be evaluating your submission of this program for documentation quality, and a few good coding practices. This will result in a deduction (ideally zero) that will be applied to your score from the Curator to yield your final score for this project.

Read the *Programming Standards* page on the CS 1704 website for general guidelines. You should comment your code in some detail. In addition to the requirements given for the Simple GIS project:

- Every class declaration must be preceded by a header comment that explains the purpose of the class, in a short paragraph, and describes the features supported in the public interface of the class.
- Every member function must be documented according to the same rules given earlier for non-memeber functions.
- Every header file must include proper directives for separate compilation.

Understand that the list of requirements given here is not a complete repetition of the *Programming Standards* page on the course website. It is possible that requirements listed there will be applied, even if they are not listed here.

As stated in the course policies, the submission that receives the highest score will be evaluated. If two or more of your submissions are tied for the highest score, then the earliest of those submissions WILL be evaluated. Therefore, DO NOT make undocumented, incomplete submissions to the Curator and then complain that you didn't want those to be evaluated. I will grant NO exceptions.

File Names

We will use the same fixed file names for this assignment. The three input files are named `CityData.txt`, `RouteData.txt`, and `GISScript.txt`, respectively. The log file is named `GISLog.txt`.

If you use the wrong names for these files, you will get poor results when you submit to the Curator.

Submitting Your Program

Submit all the source files for your implementation to the Curator in a single zip archive file.

You will submit this assignment to the Curator System (read the *Student Guid*e), and it will be graded automatically. Instructions for submitting, and a description of how the grading is done, are contained in the *Student Guid*e.

You will be allowed up to five submissions for this assignment. Use them wisely. Test your program thoroughly before submitting it. Make sure that your program produces correct results for every sample input file posted on the course website. If you do not get a perfect score, analyze the problem carefully and test your fix with the input file returned as part of the Curator e-mail message, before submitting again. The highest score you achieve will be counted.

The *Student Guide* and submission link can be found at: http://www.cs.vt.edu/curator/

Pledge

Each of your program submissions must be pledged to conform to the Honor Code requirements for this course. Specifically, you **must** include the pledge statement (provided on the course website) in the header comment for your program.

Failure to include the pledge statement may result in a substantial grade penalty.