CCIS: Class Census Information System

Fundamental Concepts: data class, dynamic array of objects

The point of this assignment is to give you beginning programming experience with programmer-defined classes and dynamic memory. This project will require you to apply what you have learned about classes and dynamic memory in CS 1704. This assignment will modify and extend SCIS. All of the specified functionality of SCIS will be retained. In order to make the program more useful, two other action commands will be implemented. The data structure that holds the in-memory census area database will also be changed; see the section on Dynamic Array Management below for details. CCIS will first load the initial area population data file as before, creating an in-memory census area database array, and then read and process actions from the `Actions.txt` script file. As with SCIS, when all the specified actions have been processed, CCIS will exit.

File Descriptions:

The initial area population data file and the script actions file will have precisely the same syntax as per SCIS, aside from some new actions.

Each line of the actions file will contain one of the commands described in the SCIS specification, or one of the new commands described below. As before, commands are case-sensitive and take a fixed number of tab-delimited arguments. The command names will be valid, and each command will include the correct number of arguments.

`find <FIPS>`

This command results in a search being performed to locate the census area object with the corresponding FIPS number. If located, the command will display the `<index>` of the object in the array followed by the `<area>` and `<state>` member fields of the object. The command output must be in the following format made to the output file, `dbase.txt`:

```
Find:     <index>     <area>      <state>
```

If a corresponding FIPS object cannot be located the output of the command must be the following, where `<FIPS>` is replaced by the FIPS number that was to be found:

```
Find:     <FIPS>      ***MISSING***
```

`gender <FIPS>`

This command also results in a search being performed to locate the census area object with the corresponding FIPS number. If located the command will display the `<area>` and `<state>` member fields of the object followed by the percentage breakdown, (to one decimal place), of females and males out of the total population in the area. The command output must be in the following format made to the output file, `dbase.txt`:

```
Gender:   <area>      <state>     Female%= 52.7     Male%= 47.2
```

If a corresponding FIPS object cannot be located the output of the command must be the following, where `<FIPS>` is replaced by the FIPS number that was to be found:

```
Gender:   <FIPS>      ***MISSING***
```

For both input files, a newline character will terminate each input line, including the last. You may assume that all of the input values will be syntactically correct, and that they will be given in the specified order.

Updated sample input files for CCIS will be posted on the course website soon. When they are available, an announcement will be posted on the course Web site.

Simple Census Area Class & Objects

You are **required** to change your census area struct type into a simple data class type. This will require the creation of constructor, reporter (get), mutator (set) and summation member function(s). Under no circumstances is the class to contain any dynamic memory allocation. The census area class will be the only allowed programmer-defined class in the program. Since the class will be a simple data class, containing no dynamic memory, no destructor will be required. In addition, no overloading of the assignment operator will be required. Member functions should be documented the same as non-member functions. See the course notes for a description of how the class member functions are to be depicted on the structure chart design.

Dynamic Array Management

You are **required** to use a dynamically-allocated array of census area class objects. Use of any STL templates for the dynamic memory allocation or any other purpose is expressly forbidden in this assignment. The dynamic array of census area objects must **not** be implemented as a class itself. As before, if the script actions file contains add commands that would overflow the array capability the commands will be ignored. The array of census area class objects must be allocated with a size to store 125% of the number of census area records in the `AreaData.txt` input file, truncated to an integer,

Input File Descriptions and Samples:

Initial Area Population File

The format of this file is unchanged from SCIS. A sample `AreaData.txt` input file is shown below.

Area	State	FIPS	SqrMiles	POP90	FEMALE90	MALE90	WHITE90	BLACK90	HISP90	OTHER90
Alexandria	VA	51510	15.847	111183	58442	52741	76789	24339	10778	4785
Bedford	VA	51515	8.574	6073	3220	2853	4691	1338	53	0
Bristol	VA	51520	11.590	18426	10202	8224	17240	1063	64	8
Buena Vista	VA	51530	2.914	6406	3499	2907	6093	282	12	0
Charlottesville	VA	51540	11.759	40341	21406	18935	30684	8561	476	94
Chesapeake	VA	51550	351.980	151976	77509	74467	107399	41662	1913	594
Clifton Forge	VA	51560	3.491	4679	2585	2094	3967	695	25	0
Colonial Height	VA	51570	9.328	16064	8554	7510	15502	129	161	70
Covington	VA	51580	5.532	6991	3729	3262	5953	969	27	44
Danville	VA	51590	17.454	53056	28864	24192	33247	19431	276	25

Database Actions File

There is no guaranteed limit on the number of actions. The changes to this file have been discussed previously, (see the File Descriptions section above). A small sample `Actions.txt` input file is shown below.

```
sort Area
del  51500
add  Alexandria     VA     51510   15.555  111111  44444   55555   16666   33333   17777   4444
del  51590
add  Falls Church   VA     51610   1.999   9999    5000    4444    8333    299     600     244
del  51610
add  Falls Church   VA     51610   1.976   9578    5005    4573    8533    298     604     247
sort FIPS
add  Galax          VA     51640   3.778   6670    3663    3007    6219    387     65      41
add  Roanoke        VA     51770   249.914 96397   51807   44590   71907   23395   665     180
add  Winchester     VA     51840   9.000   21111   11111   10000   19999   2111    211     88
del  51840
add  Winchester     VA     51840   9.059   21947   11450   10497   19453   2199    219     86
sort Area
find 51500
find 51510
find 51840
find 51850
gender      51550
gender      51770
gender      51600
dump
```

Output description and sample:

Your program must write its output data to a file named `dbase.txt` — use of any other output file name **will** result in a runtime testing score of zero. Here is a possible output file corresponding to the given sample input files:

```
Programmer:  Dwight Barnette
Class Census Information System

Find:         51500      ***MISSING***
Find:             0      Alexandria       VA
Find:         51840      ***MISSING***
Find:         51850      ***MISSING***
Gender:                  Chesapeake       VA      Female%= 51.0   Male%= 49.0
Gender:                  Roanoke          VA      Female%= 53.7   Male%= 46.3
Gender:       51600      ***MISSING***

Area           State   FIPS    SqrMiles POP90      FEMALE90  MALE90  WHITE90 BLACK90 HISP90  OTHER90
Alexandria       VA    51510    15.847   111183       58442   52741    76789  24339   10778   4785
Bedford          VA    51515     8.574     6073        3220    2853     4691   1338      53      0
Bristol          VA    51520    11.590    18426       10202    8224    17240   1063      64      8
Buena Vista      VA    51530     2.914     6406        3499    2907     6093    282      12      0
Charlottesville  VA    51540    11.759    40341       21406   18935    30684   8561     476     94
Chesapeake       VA    51550   351.980   151976       77509   74467   107399  41662    1913    594
Clifton Forge    VA    51560     3.491     4679        2585    2094     3967    695      25      0
Colonial Height  VA    51570     9.328    16064        8554    7510    15502    129     161     70
Covington        VA    51580     5.532     6991        3729    3262     5953    969      27     44
Falls Church     VA    51610     1.976     9578        5005    4573     8533    298     604    247
Galax            VA    51640     3.778     6670        3663    3007     6219    387      65     41
Roanoke          VA    51770   249.914    96397       51807   44590    71907  23395     665    180
```

The first line of your output must include your name only. The second line must include the title "`Class Census Information System`" only. The third line must be a line of underscore characters. The first and last line of each dump must be a line of underscores. The second line of a dump command output will contain the area data echoed from the current census area database array, aligned under the appropriate headers. The column field headings should be repeated for each display listing resulting from a dump. However, the three lines (programmer, program title and underscore lines) are not to be repeated.

You are not required to use the exact horizontal spacing shown in the example above, but your output must satisfy the following requirements:

- You must use the specified header and column labels, and print a row of underscore delimiters before and after the table body, as shown.
- You must arrange your output in neatly aligned columns. Use spaces, not tabs to align your output.
- You must use the same ordering of the columns as shown here, and print the Sqr Miles field with precision three.

Programming Standards:

You'll be expected to observe good programming/documentation standards. All the discussions in class, in the course notes and on the course Web site about formatting, structure, and commenting your code should be followed. Some specifics:

Documentation:

- You must include the honor pledge in your program header comment, (see below).
- You must include a header comment that describes what your program does and specifying any constraints or assumptions of which a user should be aware, (such as preset file names, value ranges, etc.).
- You must include a comment explaining the purpose of every variable or named constant you use in your program.
- You must use meaningful identifier names suggesting the meaning/purpose of the constant, variable, function, etc.
- Precede every major block of your code with a comment explaining its purpose.

- Precede every function you write with a header comment. This should explain in one sentence what the function does, then describe the logical purpose of each parameter (if any), describe the return value (if any), and state reasonable pre- and post-conditions and invariants.
- Use the assert function to check for error conditions and verify function pre- and post-conditions whenever possible.
- You must use indentation and blank lines to make control structures like loops and if-else statements more readable.

You are also required to conform to the coding requirements specified below.

Coding:

- Implement your solution in a **single source file**, with no user-defined header files. (This restriction is for ease of testing and evaluation.)
- Use named constants instead of variables where appropriate.
- Use `double` variables for all decimal numbers.
- Use an array of objects to store the census area data.
- Use C++ `string` objects, not C-style `char` arrays to store character strings, (aside from string literals).
- Declare and make appropriate use of an enumerated type in your program.
- You must make good use of user-defined functions in your design and implementation. To encourage this, the body of `main()` must contain no more than 20 executable statements and the bodies of the other functions you write must each contain no more than 40 executable statements. An executable statement is any statement **other than** a constant or variable declaration, function prototype or comment. Blank lines do not count.
- You must write at least ten functions, besides `main()`.
- The definition of `main()` must be the first function definition in your source file. You may use file-scoped function prototypes and you may use file-scoped constants. You may also make the `class` declaration statement for your census area class type file-scoped (in fact you must do this).
- You may not use file-scoped variables of any kind.
- Function parameters should be passed appropriately. Use pass-by-reference only when the called function needs to modify the parameter. Pass array parameters by constant reference (using `const`) when pass-by-reference is not needed. Pointers should be passed by reference, `const` pointer and/or `const` target as appropriate.

Interim Design:

You will produce an interim design for CCIS, and represent that design in a modular structure chart. The structure chart must indicate your design plans for CCIS. It is expected that your final design will differ from the interim design. Nevertheless, your interim design should be relatively complete. If the interim design is incomplete or if the differences between your interim design and your final code are excessive, you will be penalized. That means that you should take the production of the interim design seriously, but **not** that you should avoid changes that would improve your final implementation of CCIS. You must submit this interim design, to the Curator System, no later than midnight Friday, Feb. 9, (i.e. prior to Feb 10). Submit a MS Word.doc file. Do **NOT** compress, (zip), the interim design submission!

Testing:

Obviously, you should be certain that your program produces the output given above when you use the given input files. However, verifying that your program produces correct results on a single test case does not constitute a satisfactory testing regimen.

At minimum, you should test your program on **all** the posted input/output examples. The same program that will be used to test your solution generated those input/output examples. You could make up and try additional input files as well; of course, you'll have to determine by hand what the correct output would be.

Submitting your solution:

You will submit your source code electronically, as described here. `CCIS` will be subjected to runtime testing by the Curator automated grading system. The relative weights of the two scores will be announced. You will be allowed to make up to five submissions of `CCIS` to the Curator.

Instructions for submitting your program are available in the *Student Guide* at the Curator Homepage:

http://ei.cs.vt.edu/~eags/Curator.html

Read the instructions carefully.

Note well: your submission that receives the highest score will be graded for adherence to these requirements, whether it is your last submission or not. There will be absolutely no exceptions to this policy! If two or more of your submissions are tied for highest, the earliest of those will be graded and also evaluated by the TAs who will assess a deduction for adherence to the specified programming standards. The deduction will be applied to your highest score from the Curator. Therefore: implement and comment your C++ source code with these requirements in mind from the beginning rather than planning to clean up and add comments later.

Pledge:

Each of your project submissions to the EAGS must be pledged to conform to the Honor Code requirements for this course. Specifically, you **must** include the following pledge statement in the header comment for your program:

```
//    On my honor:
//
//    - I have not discussed the C++ language code in my program with
//      anyone other than my instructor or the teaching assistants
//      assigned to this course.
//
//    - I have not used C++ language code obtained from another student,
//      or any other unauthorized source, either modified or unmodified.
//
//    - If any C++ language code or documentation used in my program
//      was obtained from another source, such as a text book or course
//      notes, that has been clearly noted with a proper citation in
//      the comments of my program.
//
//    - I have not designed this program in such a way as to defeat or
//      interfere with the normal operation of the Curator Server.
```

Failure to include this pledge in a submission is a violation of the Honor Code.