Fundamental Concepts: array of structures, string objects, searching and sorting

The point of this assignment is to validate your understanding of the basic concepts presented in CS 1044. If you have trouble implementing the following specification correctly, that may be good evidence that either you will have difficulty in this course or that you are not ready to attempt CS 1704. Feel free to consult the online notes for CS 1044 as a reference. Follow the given specification exactly — this assignment will be scored using an automated grading system and deviations will generally be penalized heavily.

SCIS: Simple Census Information System

With the year 2000 census data starting to become available, programs are needed to manage, tabulate and help analyze the data. The SCIS program will read an input file that will specify area/city, state and population information (see file description below). The area population data file will begin with a header line that should be ignored. The remainder of the file will consist of lines of data that conform to the format specification:

```
<area name><tab><state name><tab><FIPS><tab><square
miles><tab><total population><tab><female
population><tab><male population><tab><white
population><tab><black population><tab><hispanic
population><tab><other population><newline>
```

The number of data lines is unknown (although it will never exceed 100), so your program must read until input failure at the end of the file. A sample area data file is given later in this specification. You must define an appropriate structured type to store all the information about each area.

The program will first read and store the area data, in an in-memory database structure, and then process a second input file specifying actions to take on the area data. Supported actions include adding an area record to the database, deleting an area record from the database, sorting the database on various fields (keys), and dumping a display of the database to file. When all the specified actions have been processed, the program will exit.

Input file descriptions and samples:

This program requires the use of two input files. The first file contains the area population data and will be named `AreaData.txt`. The second file contains a list of actions to be performed on the area record database, and will be named `Actions.txt`. Note that, due to the automated testing process, use of incorrect input file names will result in a score of zero.

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.

Initial Area Population File

The first line of the area population input file is a label line that specifies column labels. On each of the remaining lines will be eleven data fields, separated by tab characters. The order of the data fields on the line and the type of value in the field are given in the table at the right.

A sample `AreaData.txt` input file is shown below.

Area Data Field Name	Field Contents
Area Name	character string
State Name	character string
FIPS	integer
Square Miles	double
Total Population	integer
Females	integer
Males	integer
Whites	integer
Blacks	integer
Hispanics	integer
Other	integer

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

There will never be two different area records with the same FIPS, (Federal Information Processing Standards), number in the database at the same time. (The FIPS codes define unique state/place coordinate systems.) Each of the other fields may be duplicated within the database. There will be no more than 100 items in the area population database at any time.

You are **required** to use a statically-allocated array of structures to store the area information. Use of pointers, classes, STL templates and/or dynamic memory allocation is expressly forbidden in this assignment.

Note that the alignment of the area population information in the initial file may not be perfect, because the combination of tabs may not align the numbers/fields correctly. This is a good example of why it is suggested that you use spaces (not tabs) to align your output. Here, the tab character is used because it makes it somewhat easier for you to parse the item descriptions.

Database Actions File

Each line of the actions file will contain one of the commands described below. Commands are case-sensitive and take a fixed number of arguments. The command names will be valid and each command will include the correct number of arguments. Command arguments will be tab-delimited.

```
add   <area name><tab><state name><tab><FIPS><tab><square miles><tab><total
population><tab><female population><tab><male population><tab><white
population><tab><black population><tab><hispanic population><tab><other
population><newline>
```

This causes the insertion of a new area population record into the database list. Insertion should place the new record in the proper location with respect to the current sort ordering of the list. The initial area list will be given in arbitrary order, and you must initially sort it by the FIPS number before further processing. If an `add` instruction specifies the <`FIPS`> number of an item that's already in the list, the list will not be modified. Note that due to page limitations, (not file line size), some of the arguments of the add command description above have wrapped onto the next line. In the actual `Actions.txt` input file they will all be tab separated on the same line. Note: appending the new record to the end of the array and re-sorting is not an insertion operation and will be penalized. If the number of area population records stored is at the maximum, 100, and an add operation is encountered then the list must not be modified and the add operation will be skipped and never processed.

```
del <FIPS>
```

This causes the deletion of the area record for the indicated <`FIPS`> number from the database list. If a `del` instruction specifies the number of an item that's not in the list, the list will not be modified.

```
sort <FieldSpecifier>
```

This causes the area list to be sorted into ascending order by the specified field. The FieldSpecifier must be one of: `FIPS`, or `Area`. If a `sort` instruction specifies an invalid FieldSpecifier, the list will not be modified. You must use the bubble sort algorithm, ordering the records first by Area name and then by State name string fields alphabetically when an Area FieldSpecifier is indicated.

`dump`

> This causes the area population information to be printed to an output file named `dbase.txt`. Printing should be in the physical order of the list resulting from the last sort. More than one dump command may exist in the actions file in which case the area listings for each dump command must be appended to the last listing.

There is no guaranteed limit on the number of actions. A small sample `Actions.txt` input file is shown below. Note that due to page limitations, (not file line size). In the actual `Actions.txt` input file they will all be tab separated on the same line.

```
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
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 an output file corresponding to the given sample input files:

```
Programmer:  Dwight Barnette
Simple Census Information System
_________________________________________________________________________________________________
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
Winchester      VA     51840    9.059    21947       11450   10497   19453   2199    219    86
_________________________________________________________________________________________________
```

The first line of your output must include your name only. The second line must include the title "`Simple Census Information System`" only. The third line must be a line of underscore characters; the fourth line must display the column labels shown above. The fifth line will contain the area data echoed from the current database, aligned under the appropriate headers. The last line of each dump must be a line of underscore characters. 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 without any user-defined classes.
- 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 structure variables to store the movie 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 `typedef` statement for your structured variable 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.

Design:

An initial structure chart design of the program is NOT required. However, students may wish to go ahead and produce a structure chart design as all other projects will require structure chart designs and will build off of this project. If a design is produced it is not to be submitted in any manner for evaluation or documentation.

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. `SCIS` 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 `SCIS` 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.