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 much difficulty implementing the following specification correctly, that may be good evidence 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.

SAMS: Statistical Appraisal Movie System

A local movie rental company needs to track information about its movies and tapes. The program will read an input file that will specify movie/tape information (see file description below). The movie file will consist of two lines of data for each movie. The first line will contain typical film data. This line will be followed by a short one line review of the movie. You should `typedef` an appropriate structure variable to store all the information about each movie.

The program will first read and store the movie data, in an in-memory database structure, and then process a second input file specifying actions to take on the movie data. Supported actions include adding a movie to the database, deleting a movie 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 rental company's movie information and will be named `movie.data`. The second contains a list of actions to be performed on the movie database, and will be named `actions.data`. Note that, due to the automated testing process, use of incorrect input file names will usually 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 Movie File

The first line of the movie input file is a label line that specifies column labels. Each of the following pair of lines comprises a movie record. The first line of each pair specifies nine data fields, separated by one or more 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. The second line of each pair is simply a character string containing a very short synopsis/review of the film.

Movie Data Field Name	Field Contents
Title	character string
Director	character string
Release Year	integer
Performers	character string
Rating	character string
Stars	real
Stock	integer
Rental	integer
Serial	character string

A sample `movie.data` input file is shown below, (the first two lines preceding the box are column label numbers and are not part of the file).

```
000000000111111111122222222223333333333444444444455555555556666666666777777777 8
123456789012345678901234567890123456789012345678901234567890123456789012345678 90
// Title       Director       Release Performers            Rating  Stars   Stock   Rental  Serial
Dragon Slayer  Unknown        0       R Richardson          G       3.5     2       1       FAN121610
Whoever produced this movie should be Walt Disney's son.
Alien          Cameron        1978    Sigourney Weaver      PG13    4.0     3       0       SCI010121
The special effects were  not only technically good, but innovative.
Star Wars      G Lucas        1976    Hamill, Ford, Fisher  G       5.0     5       2       SCI328412
A classic SciFi version of the eternal conflict of Good versus Evil.
Blade Runner   R Scott        1982    H Ford, R Hauer       R       3.0     2       0       SCI216193
This movie is a cult classic, a must see.
Predator       Unknown        1988    Arnie                 R       3.0     1       1       SCI482617
Alien picture where IT doesn't land in boondocks fighting Jeb and Billy Bob.
Terminator     Unknown        0       Arnie, Linda Hamilton R       2.5     2       0       SCI628105
Robot from future trying to change the future.
```

There will never be two different movie records with the same Serial number in the database at the same time. Each of the other fields may be duplicated within the database. There will be no more than 100 items in the movies database at any time.

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

Note that the alignment of the movie information in the initial movie 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 <Title> <Director> <Release Year> <Performers> <Rating> <Stars> <Stock> <Rental> <Serial> <Review>`

This causes the insertion of a new movie 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 movie list will be given in arbitrary order, and you must initially sort it by serial number before further processing. If an `add` instruction specifies the <`Serial`> 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), the last three arguments of the add command description above have wrapped onto the next line. In the actual `actions.data` 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. The number of movie 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 <Serial>`

This causes the deletion of the movie record for the indicated <`Serial`> 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 movie list to be sorted into ascending order by the specified field. The FieldSpecifier must be one of: `Serial`, or `Title`. If a `sort` instruction specifies an invalid FieldSpecifier, the list will not be modified. You should use the selection sort algorithm, ordering the records by the string fields using ASCII ordering, (not alphabetically).

`dump`

This causes the movie information, (excluding the one-line reviews) to be printed to an output file named `dbase.list`. 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 movie 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.data` input file is shown below. Note that due to page limitations, (not file line size), some of the fields and one line reviews of the add commands have wrapped onto the next lines. In the actual `actions.data` input file they will all be tab separated on the same line.

```
del     SCI628105
add     Terminator      Cameron         0       Arnie, Linda Hamilton  R       3.0     2       0
        SCI628105       Robot from future trying to change the future.
del     FAN121610
add     Dragon Slayer   Robbins         1981    R Richardson           G       3.5     2       1
        FAN121610       Whoever produced this movie should be Walt Disney's son.
sort    Title
del     FAN121619
add     Forbidden Planet        FM Wilcox       1956    L Neilsen & W Pidgeon  G       3.0     1       0
        SCI408731       This movie overall was quite good for early Sci-Fi.
add     Fright Night    T Holland       1985    R McDowell & C Sarandon        PG13    3.0     1       1
        HOR010232       This movie overall was modern fright.
add     GoodBye Girl    Herbert Ross    1977    R Dreyfuss & M Mason           G       2.5     2       0
        COM319631       This movie overall was the best movie of the year.
sort    Serial
del     COM319631
add     GoodBye Girl    Herb Ross       1977    R Dreyfuss & M Mason           G       3.0     2       0
        COM319631       This movie was the best film of 1977.
sort    Title
dump
```

Output description and sample:

Your program must write its output data to a file named `dbase.list` — 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
Statistical Appraisal Movie System

Title              Director   Release    Performers          Rating  Stars   Stock   Rental  Serial
Alien              Cameron    1978    Sigourney Weaver         PG13  4.0     3       0       SCI010121
Blade Runner       R Scott    1982    H Ford, R Hauer          R     3.0     2       0       SCI216193
Dragon Slayer      Robbins    1981    R Richardson             G     3.5     2       1       FAN121610
Forbidden Planet   FM Wilcox  1956    L Neilsen & W Pidgeon    G     3.0     1       0       SCI408731
Fright Night       T Holland  1985    R McDowell & C Sarandon  PG13  3.0     1       1       HOR010232
GoodBye Girl       Herb Ross  1977    R Dreyfuss & M Mason     G     3.0     2       0       COM319631
Predator           Unknown    1988    Arnie                    R     3.0     1       1       SCI482617
Star Wars          G Lucas    1976    Hamill, Ford, Fisher     G     5.0     5       2       SCI328412
Terminator         Cameron    0       Arnie, Linda Hamilton    R     3.0     2       0       SCI628105
```

The first line of your output must include your name only. The second line must include the title "`Statistical Appraisal Movie 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 movie data echoed from the current database, aligned under the appropriate headers. The last line must be a line of underscore characters. Each listing must be terminated by a line of underscore characters. The column field headings should be repeated for each movie 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 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 Stars field with precision one.

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. SAMS will be subjected to runtime testing by the Curator automated grading system and also scored by the TAs for adherence to the specified programming standards. The relative weights of the two scores will be announced. You will be allowed to make up to five submissions of SAMS to the Curator.

Instructions for submitting your program are available in the *Student Guide* at the EAGS 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. If two or more of your submissions are tied for highest, the earliest of those will be graded. 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.