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.

SARP: Static Advising Records Program

Walden College needs a software system to help students and advisors track a students progress towards a degree. The program will read an input file that will specify an individual student's academic information (see student file description below). The student file will consist of student identification, address, and academic information at the beginning of the file. This will be followed by a listing of the course data for the student. You should `typedef` an appropriate structure variable to store all the information about a particular student and their courses.

The program will first read and store the student data, in a in-memory database structure, and then process a second input file specifying actions to take on the course data. The supported actions include adding a course to the database, dropping (deleting) a course from the database, sorting the database on various 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 contains the student's information and initial course list and will be named `student.data`. The second contains a list of actions to be performed on the course 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 Student File

The first line of the student input file is a label line that specifies column labels. The second line specifies thirteen data fields, separated by one tab character. 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 last four fields are the overall quality credit average, (QCA), alternate (major) QCA, the overall quality credits earned and the overall hours attempted. Note that the contents of the last four fields may not be up-to-date. The third line of the file will be a separator line of dashes.

Student Data Field Name	Field Contents
ID number	9 character string
Name	20 character string
Address	20 character string
City	10 character string
State	2 character string
Zip	5 digit positive integer
Major	4 character string
Minor	4 character string
Rank	2 digit positive integer
QCA	4 decimal place float
Alt. QCA	4 decimal place float
Qual. Cred.	2 decimal place float
Hrs. Att.	3 digit non-negative integer

The fourth line will contain labels for the course record fields. Starting on the fifth line, the student's course records will begin and continue until the end of the file. Each course record, specifies nine data fields, separated by one tab character. The order of the data fields on a line and the type of value in the field are given in the table at the right. It may be assumed that the ID number in each courses record matches the ID number in the student record. It may **NOT** be assumed that the course records in the file will be in any particular order. You may assume that the maximum number of courses is a positive integer less than or equal to 100. You may also assume that the student and course data is valid and does not need to be checked for errors. Your program must be written so that it will detect when it's out of input and terminate correctly. The term field abbreviations are as follows: FS – fall semester, SS – Spring Semester, U1, first summer term, and U2 – second summer term. The grade field codes are given below in the letter grade quality credits table.

Courses Data Field Name	Field Contents
ID number	9 character string
Index	4 digit positive integer
Department	4 character string
Course #	4 digit positive integer
Term	2 character string
Year	4 digit positive integer
Credit Hours	2 decimal place float
Grade	2 character string
Title	20 character string

A sample `student.data` input file is shown below, (the first two lines preceding the box are column label numbers and are not part of the file). Note that due to page limitations, (not file line size), the first two lines of the input file have wrapped onto the next lines.

```
000000000111111111122222222223333333333444444444455555555556666666666777777777 8
123456789012345678901234567890123456789012345678901234567890123456789012345678 90
// ID       Name              Address           City        St    Zip
      Majr  Minr  Rk    QCA         AltQCA      Crd   Hrs
135792468   Wayne,John Duke   101 Hollywood Way Hollywood   CA    40815 CS
      MATH  10    3.2667      4.0000      49.00 15
-------------------------------------------------------------------------------
// ID       Indx  Dept  Crse  Tm    Year  Cred  Grd   Title
135792468   9345  CS    1044  FS    1999  3.00  A     Intro Prog in C
135792468   5231  CS    1205  FS    1999  1.00  A     Oper Sys Tools I
135792468   1350  CS    1206  SS    2000  2.00  B+    Oper Sys Tools II
135792468   9268  CS    1104  FS    1999  3.00  A     Intro to CS
135792468   1365  CS    1704  SS    2000  3.00  B     Intro Data Struct/SE
135792468   6075  ENGL  1105  FS    1999  3.00  C     Fresh English
135792468   2080  ENGL  1106  SS    2000  3.00  B+    Fresh English
135792468   7410  MATH  1114  FS    1999  2.00  C+    Elem Lin Alg
135792468   7430  MATH  1205  FS    1999  3.00  B+    Calculus
135792468   3470  MATH  1206  SS    2000  3.00  B     Calculus
135792468   3510  MATH  1224  SS    2000  2.00  B-    Vector Geometry
135792468   8082  PHYS  2074  SS    2000  3.00  A-    Hilights of Physics
```

There will never be two different course records with the same Index 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 courses database at any time.

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

Note that the alignment of the student and course information in the initial student file may not be perfect, because the combination of tabs and spaces may not align the numbers/fields correctly. This is a good example of why we suggest that you use spaces (not tabs) to align your output. Here, we use the tab character 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   <IdNumber> < IndexNumber > <Dept> <CourseNumber> <Term> <Year> <CreditHours>
      <Grade>     <Title>
```

This causes the insertion of a new course 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 course list will be given in arbitrary order, and you must initially sort it by index number before further processing. If an `add` instruction specifies the <IndexNumber> number of an item that's already in the list, the list will not be modified.

```
drop <IndexNumber>
```

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

```
sort <FieldSpecifier>
```

This causes the courses list to be sorted into ascending order by the specified field. The FieldSpecifier must be one of: `Index`, or `Course`. If a `sort` instruction specifies an invalid FieldSpecifier, the list will not be modified. You should use the selection sort algorithm. For the `Course` FieldSpecifier the sort should be performed on both the `Department` and `Course` number fields.

```
dump
```

This causes the student's QCA, Alternate QCA, Quality Credits and Credit hours to be computed, the student information and the current courses list, (omitting the ID number), to be printed to an output file named `dbase.list`. Printing should be in the physical order of the list. (For instructions on how to compute the QCAs see the end of this document.)

There is no guaranteed limit on the number of actions. A sample `actions.data` input file is shown below:

```
drop   8082
add    135792468    8082   PHYS   2074   SS     2000   3.00   A      Hilights of Physics
drop   6075
add    135792468    6075   ENGL   1105   FS     1999   3.00   C-     Fresh English
sort   Course
add    135792468    9267   CS     2704   FS     2000   3.00   C+     Obj Orient Prog
add    135792468    9260   CS     2504   FS     2000   3.00   C      Intro Comp Org
add    135792468    7264   MATH   2224   FS     2000   3.00   C-     Mult Var Calc
drop   9260
add    135792468    9260   CS     2504   FS     2000   3.00   B      Intro Comp Org
sort   Index
drop   9267
add    135792468    9276   MATH   2534   FS     2000   3.00   B+     Discrete Math
add    135792468    9267   CS     2704   FS     2000   3.00   C      Obj Orient Prog
add    135792468    8140   PHYS   2305   FS     2000   4.00   B-     Found Phys I
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
Static Advising Records Program

ID Number Name               Major Minor Rank
135792468 Wayne,John Duke    CS    MATH  10

QCA    AltQCA QCredits Hours Att.
2.9851 3.2556 140.3000    47.00

Indx  Dept  Crse  Tm    Year  Cred  Grd   Title
1350  CS    1206  SS    2000  2.00  B+    Oper Sys Tools II
1365  CS    1704  SS    2000  3.00  B     Intro Data Struct/SE
2080  ENGL  1106  SS    2000  3.00  B+    Fresh English
3470  MATH  1206  SS    2000  3.00  B     Calculus
3510  MATH  1224  SS    2000  2.00  B-    Vector Geometry
5231  CS    1205  FS    1999  1.00  A     Oper Sys Tools I
6075  ENGL  1105  FS    1999  3.00  C-    Fresh English
7264  MATH  2224  FS    2000  3.00  C-    Mult Var Calc
7410  MATH  1114  FS    1999  2.00  C+    Elem Lin Alg
7430  MATH  1205  FS    1999  3.00  B+    Calculus
8082  PHYS  2074  SS    2000  3.00  A     Hilights of Physics
8140  PHYS  2305  FS    2000  4.00  B-    Found Phys I
9260  CS    2504  FS    2000  3.00  B     Intro Comp Org
9267  CS    2704  FS    2000  3.00  C     Obj Orient Prog
9268  CS    1104  FS    1999  3.00  A     Intro to CS
9276  MATH  2534  FS    2000  3.00  B+    Discrete Math
9345  CS    1044  FS    1999  3.00  A     Intro Prog in C
```

The first line of your output must include your name only. The second line must include the title "`Static Advising Records Program`" 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 student data echoed from the input file, aligned under the appropriate headers. The sixth line must be blank. The seventh line must display the column labels for the QCA calculations as shown above. Note that the QCA calculations must be based upon all of the current course records in the database list. The eighth line must contain the results of the QCA calculations, aligned under the appropriate headers. The ninth line must be a line of underscore characters.

The tenth line must display column labels for the course records. Next your output file will contain a table, with a line of output for each course record in the database list. Each line should contain the index number, department, course number, term, year, credits, grade and the title of the course. After the last line of the table, print a line of underscore character delimiters.

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 QCA, Alternate QCA and Quality Credits with precision four and the hours attempted with precision two.

Programming Standards:

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

Documentation:

- You must include the honor pledge in your program header comment.
- 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 that suggest the meaning or 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.
- 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 course 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 given along with this specification. 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. SARP will be subjected to runtime testing by the Curator automated grading system and also scored by the GTAs 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 SARP 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.

QCA computation

A QCA is computed by dividing the quality credits achieved by the credit hours attempted. Quality credits are earned for passing grades in courses. The quality credits earned in a course are obtained by multiplying the corresponding letter grade quality credits by the credit hours for the course. (See the table at the right for the letter grade quality credits.)

A student has several QCAs: overall QCA, term QCA and major QCA. The overall QCA is the total quality credits achieved in all courses divided by the total credit hours attempted. A term or semester QCA is simply the quality credits earned during a particular term divided by the credit hours attempted during the term. A student's major QCA is the sum of all the quality credits earned for courses taken in a student's department divided by the total major departmental hours attempted. (Note – some schools and departments define major QCA differently. Courses passed with a grade of 'P' do not affect a QCA, i.e. the hours are not counted in the hours attempted.)

Letter Grade	Quality Credit
A	4.0
A-	3.7
B+	3.3
B	3.0
B-	2.7
C+	2.3
C	2.0
C-	1.7
D+	1.3
D	1.0
D-	0.7
F	0.0
P	0.0