Structure Array, Character Strings, Searching and Sorting

This programming assignment uses many of the ideas presented in sections 10 and 11 of the course notes. You are advised to read those carefully. Read and follow this program specification closely. Your scores on this project include the score you receive from the auto-grader and the score you receive for following the instructions in the Programming Standards, which includes the program design.

The Program Specification: QCA calculation

For every student the university stores a wealth of information. Much of this data pertains to the student's quality credit average, (QCA). 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.)

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

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.)

The program you create for this assignment will compute an individual student's overall QCA and major QCA for a list of courses the student has attempted. In addition, your program will produce a bar chart showing the progressive term by term QCAs.

Input file descriptions and samples:

Your program **must** read its input from **two** files named `student.dat` and `courses.dat` — use of other file names will result in a score of zero. The `student.dat` file will store the individual student information. The format of the `student.dat` file will now be described. The first line of the student input file is a label line that must be ignored. The second line specifies twelve 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 three fields are the 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.

Student Data Field Name	Field Contents
ID number	9 character string
Name	20 character string
Address	25 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 real value
Alt. QCA	4 decimal place real value
Qual. Cred.	3 digit non-negative integer
Hrs. Att.	3 digit non-negative integer

The contents of the `courses.dat` file will now be explained. The first line contains a non-negative integer number which represents the number of course records in the file. These records start on the third line. All other data following the integer on line 1 must be ignored. The second line is a label line that must be skipped. Each course record, starting on line 3 of the courses file, specifies 9 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 the courses file matches the ID number in the student file. 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 above 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.dat` input file is shown below, (the first two lines are column labels and are not part of the file):

```
0000000001111111111222222222233333333334444444444555555555566666666667777777778
1234567890123456789012345678901234567890123456789012345678901234567890123456789
// ID       Name              Address           St    Zip   Majr  Minr  Rk
      QCA         AltQCA      Crd   Hrs
135792468   Wayne,John Duke   101 Hollywood Way CA    40815 CS    MATH  10
      3.2667      4.0000      49    15
```

Note that due to page limitations, (not file line size), the two lines of the input file have wrapped onto the next lines. (See the last pages of this specification for a non-wrapped view of this sample student file.)

A sample `courses.dat` input file is shown below, (the first two lines are column labels and are not part of the file):

```
0000000001111111111222222222233333333334444444444555555555566666666667777777778
1234567890123456789012345678901234567890123456789012345678901234567890123456789
 12 = Number of Courses
// 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
```

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 and is correctly specified on the first line. 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, just as in the previous projects.

Output description and sample:

The first line of your output must include your name only; the second line must include the title "QCA calculation" only; the third line must be a line of underscore characters; the fourth line must display the column labels shown below. 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 below. Note that the QCA calculations must be based upon all of the course file records. 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 be blank.

The next section of the output will give a horizontal bar chart of the student's term by term QCA progression in chronological order by term and year. To assure that the QCA bar chart output is ordered correctly, a sort on the course records must be performed, ordering upon the year and term. Please be aware that the academic year begins with the Fall semester. The eleventh line must contain the bar chart label as shown. The twelfth line must have the chart columns labels exactly as shown. The number of following lines of the output file, (the QCA bar chart lines), will depend upon the number of academic year course listings stored in the courses input file. Each line, one per term, will contain the term code and year aligned under the labels, immediately followed by a colon. Beginning in column eleven on each line will be a number of asterisks representing the QCA bar chart value for that term. Forty columns will be used for the bar chart line representation of each term QCA. To determine how many asterisks for a term QCA should be output, a mapping of the numerical term QCA value to the maximum column length must be performed. The equation for this mapping is:

$\left(\frac{TermQCA}{MaxQCA}\right) \times MaxColLength$ or for this particular bar chart the formula becomes: $\left(\frac{termQCA}{4.0}\right) \times 40$. The term QCA over 4.0 gives a QCA percentage used to determine the number of asterisks output. The mapping should be rounded to the nearest integer value for an exact number of asterisks to be output.

Your program must write output data to a file named `qca.dat` — use of any other output file name will result in a score of zero. The sample output file shown below corresponds to the input data given above, (the first two lines of column labels are not part of the file):

```
0000000001111111111222222222233333333334444444444555555555566666666667777777778
1234567890123456789012345678901234567890123456789012345678901234567890123456789
Dwight Barnette
QCA calculation
_____________________________________________________
ID Number Name               Major Minor Rank
135792468 Wayne,John Duke    CS    MATH  10

QCA    AltQCA QCredits Hours Att.
3.2097 3.6333 99.5000     31
_____________________________________________________

QCA Progression:
Term Year ........1.0.......2.0.......3.0.......4.0
FS   1999:********************************
SS   2000:********************************
```

You are NOT required to use this exact horizontal spacing, but your output must satisfy the following requirements:

- You must use variables of type `int` for the hours attempted and rank.
- You must use variables of type `double` for all the computed values, if you use float variables, your answers may be incorrect due to numerical roundoff.
- All decimal values must be printed to show the same number of decimal places, 4, as in this sample.
- You must use the exact specified header and column labels given above.
- Include your name in the first line as shown.

- You must arrange your output in neatly aligned columns, with the identifying labels shown in the example. Note that while the auto-grader doesn't deduct points for horizontal alignment, the TA who grades your source code for programming standards may do so.
- You must use the same ordering of the columns as shown here.
- You must print a newline at the end of each line, including the last line.

Programming Standards:

You'll be expected to observe good programming/documentation standards. All the discussions in class about formatting, structure, and commenting your code will be enforced. A copy of *Elements of Programming Style* is included with the course notes and is available on-line from the course Web site. Some specific requirements follow.

Documentation:

- For the outline of your design:
  You must include, below the header comments, an outline program design. The design must reflect your top-down functional/modular decomposition of the problem. The design should follow the layout of the payroll design example in appendix 4 of the course notes.

- You must include header comments specifying the compiler and operating system used and the date completed.
- Your header comment must describe what your program does; don't just plagiarize language from this spec.
- You must include a comment explaining the purpose of every variable or named constant in your program.
- You must use meaningful identifier names that suggest meaning or purpose of the constant, variable, function, etc.
- Precede every major block of your code with a comment explaining its purpose. You don't have to describe how it works unless you do something so sneaky it deserves special recognition.
- Each user-defined function must have a valid C++ prototype and the definition of each user-defined function (except `main`) must be preceded by a block comment, explaining in one sentence what the function does, and listing each parameter to the function and explaining its purpose.
- You must use indentation and blank lines to make control structures like loops and if-else statements more readable and comment each logical section/structure.
- You may copy, (that's copy not move), parts of your design outline from the top of your file to relevant points in your code to use as comments and to clearly show how your code matches your design. However, unless you have created an very "complete" design, you will still need to supplement it with inline code comments as explained in the "Elements of Programming Style".

Implementation:

- Use named constants instead of variables or literal constants where appropriate, (e.g., array dimensions).
- Use `bool` variables, typedefs and enum types where appropriate.
- Do not use `if…else` statements, when a `switch` statement can be used.
- Choose your control structures appropriately. Do not use while loops for count controlled loops.
- Your implementation must minimally include **five** user-defined functions, not counting main.
- You may use file-scoped function prototypes , constants and user-defined types.
- You may <u>not</u> use file-scoped variables of any kind.
- 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.
- The definition of main() must be the first function definition in your source file. The remaining function definitions should be grouped logically.
- 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.
- Parallel arrays may not be used in this program. Use of an **array of structures** is required.

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 may be selected for grading. 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.

Incremental Development:

You'll find that it's easier and faster to produce a working program by practicing incremental development. In other words, don't try to solve the entire problem at once. First, develop your design. When the time comes to implement your design, do it piece by piece. Here's a suggested implementation strategy for this project. As usual, verify and correct as necessary after each addition to your implementation.

- First, write the code necessary to read the entire input file into simple variables and echo it. This should be similar to the input code used in all other program assignments.
- Second, add the code to store the input data in an array of structures and produce output lines, (table headings, IDs, names, etc.), that does not require calculation.
- Third, add the code to compute the QCAs, credits and hours.
- Fourth, write the code to output the labels for the bar chart
- Fifth, implement code to sort the course records by term and year.
- Sixth, add code to compute, and output the term QCAs.
- Finally, write code to map the term QCAs to asterisk lines and output them.

Now you have a substantially complete program. At this point, you should clean up your code, eliminating any unnecessary instructions and fine-tuning the documentation you already wrote. Check your implementation and output again to be sure that you've followed all the specifications given for this project, especially those in the Programming Standards section above. At this point, you're ready to submit your solution to the auto-grader.

Testing:

At minimum, you should ascertain that your program produces the output given above when you use the given input file. However, verifying your program produces correct results on one test case does not constitute a satisfactory testing regimen.

Additional input/output file examples will be posted on the Web site. You may use those for additional testing. You should make up and try additional input files as well; determining by hand what the correct output would be.

Student input file sample:

A sample `student.dat` input file is shown below, (the first two lines are column labels and are not part of the file):

```
0000000001111111111222222222233333333334444444444555555555566666666667777777777888888888899999999990000000000111111111122222222223
12345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
// ID       Name              Address           St    Zip   Majr  Minr  Rk    QCA         AltQCA      Crd   Hrs
135792468   Wayne,John Duke   101 Hollywood Way CA    40815 CS    MATH  10    3.2667      4.0000      49    15
```