Iteration: while, for, do while, Reading Input with Sentinels and User-defined Functions

This programming assignment uses many of the ideas presented in sections 6 and 7 of the course notes. You are advised to read those carefully. Read and follow the following program specification carefully. Your score on this project will be a weighted average of the score you receive for runtime testing and the score you receive for following the instructions in the Programming Standards section below.

The Program Specification: Bowling Scores

Bowling results for a team are placed into a file by an automatic sensor system. The results are stored on a frame by frame basis depicting the outcome of each ball rolled using the standard bowling scoring abbreviation symbols. The symbols and their meaning is given in the table below:

Standard Bowling Abbreviations	
Symbols	Semantics
0 … 9	Pins toppled
/	Spare
X	Strike
–	Gutter
F	Foul

If you don't know how to compute the score of a bowling game the rules are simple. A bowler has two chances, (rolls of the ball), to attempt to knock over the ten pins on the lane. If all ten pins are not toppled in a frame then the game score is simply increased by the number of pins knocked over in that frame on both balls. If all ten pins are toppled with two balls, (a spare), the score for that frame is equal to the score of the previous frame, (zero prior to the first frame), increased by ten plus the number of pins toppled on the next ball, (the first ball of the next frame). Thus the maximum increase in a frame with a spare is twenty. (Assuming all the pins are toppled on the first ball of the next frame, a strike in the next frame). If all ten pins are toppled with a roll of one ball, (a strike), the score for that frame is the game score at that point, (the score of the previous frame), increased by ten plus the number of pins toppled on the next 2 balls rolled. This count will be either the counts of the pins toppled by the balls of the next frame or if the next frame is also a strike it will be twenty plus the count of the first ball in the second succeeding frame. Thus the maximum increase in a frame with a spare is thirty, (if strikes are also rolled in the next two frames). The minimum increase in a frame with a strike is ten if the next two balls are rolled in the gutter or are fouls. Gutter balls and fouls count and score the same, which is zero. There is one exception to the two ball per frame rule. In the tenth and last frame only, if all the pins are knocked down on the first ball, two more balls may be rolled and if all the pins are toppled on two balls then one more ball may be rolled, (to add to the count of the spare). Information on the basics of bowling can be found at: http://www.bowlingindex.com/instruction/totw.htm and a complete discussion of bowling scoring can be found at the site: http://www.bowlingindex.com/instruction/scoring.htm

Input file description and sample:

Your program **must** read its input from a file named `scores.dat` — use of another file name will result in a score of zero. The first line of the input file specifies a positive integer representing the number bowler's scores in the file. Each remaining line of the input file will fit the format given in the table at the right.

Columns	Contents
1 – 19	Bowler's Name
20	Blank
21…	Frame scores

You may assume that all the input values will be logically correct. For instance, given the sample input file below, (the first two lines of column labels are not part of the file) :

```
0000000001111111111222222222233333333334444444444555555555566666666667777777778
1234567890123456789012345678901234567890123456789012345678901234567890123456789
2
Dwight Barnette     X9/XXXXX9/XXX9
Bill McQuain        63-8F/--XX-/X9-8/X
```

Note that you must **not make any assumptions** about the number of bowlers or the number of columns of data used for the frame scores, beyond assuming that the number of bowlers is a positive integer and correctly specified in the file header. There may be as few as 12 columns of frame score data or as many as 21 columns. You may assume that the frame data is valid with respect to the rules of bowling 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.

What to Calculate:

You will write a program to read the input file and use the given values to:

- Compute the frame by frame score for each bowler.
- Compute the total of all bowlers' scores.
- Determine the score of the highest and lowest games.
- Compute the frame by frame totals for all combined bowlers

Output description and sample:

The first line of your output must include your name only; the second line must include the title "Bowling Scores" only; the third line must be a line of 71 underscore characters; the fourth line must display the column labels shown below; the fifth line must consist of the delimiting characters shown below to separate the column labels from the body of the table.

Next your output file will contain a table, with three lines of output for each line of data processed from the input file. The first of these three lines must list the bowler's name, and the frame by frame standard bowling scoring symbols under the appropriate column labels using the spacing in the following example. The second line for each bowler must be a line of frame delimiter characters as depicted below. The third output line for a bowler, must contain the frame by frame scores, (using the standard bowling scoring system explained above), properly delimited. There should be a line of delimiters, (equivalent to the fifth line), immediately after the third line of output for each bowler.

After the last bowler, a line labeled "Total" starting in column one, with frame by frame totals of all combined bowler's scores must be produced with colon delimiters. The last line of table output must be a line of 71 underscore characters. The last two lines of output must label and give the low game score and the high game score of all bowlers as shown below.

Your program must write output data to a file named `bowlers.out` — 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):

```
00000000011111111112222222222333333333344444444445555555555666666666677777777778
12345678901234567890123456789012345678901234567890123456789012345678901234567890
Dwight Barnette
Bowling Scores
_______________________________________________________________________
Bowler             |  1 |  2 |  3 |  4 |  5 |  6 |  7 |  8 |  9 |  10 |
-------------------+----+----+----+----+----+----+----+----+----+-----|
Dwight Barnette    |  X |  9/|  X |  X |  X |  X |  X |  9/|  X |  XX9|
                   |    |    |    |    |    |    |    |    |    |     |
                   |  20|  40|  70| 100| 130| 159| 179| 199| 229|  258|
-------------------+----+----+----+----+----+----+----+----+----+-----|
Bill McQuain       |  63|  -8|  F/|  --|  X |  X |  -/|  X |  9-|  8/X|
                   |    |    |    |    |    |    |    |    |    |     |
                   |   9|  17|  27|  27|  47|  67|  87| 106| 115|  135|
-------------------+----+----+----+----+----+----+----+----+----+-----|
Total              :  29:  57:  97: 127: 177: 226: 266: 305: 344:  393:
_______________________________________________________________________
Low: 135
High: 258
```

You are required to use this exact horizontal spacing.
Your output must additionally satisfy the following requirements:

- You must use variables of type `int` for all the frame scores, totals and low/high game, etc. If you use decimal values, your answers may be incorrect due to numerical roundoff.
- 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 NAG 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:
- 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 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. 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. The payroll code from Project 1 is a good guide.

Outline Design:
- You must include, below the header comments and pledge, 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.

Implementation:
- Use named constants instead of variables or literal constants where appropriate.
- Use `bool` variables 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 four user-defined functions, not counting main.
- Arrays may not be used in this program.

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.

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. This should be similar to the input code used in the payroll program. To test your work, include code to write what you're reading (and nothing else) to the output file.
- Second, add the code to produce output lines, (table headings, frame bowling symbol output, etc.) that does not require calculation.
- Third, add the code to compute the frame scores.
- Fourth, write the code to compute the total combined bowlers' frame scores.
- Finally, implement code to determine the low and high scores.

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 NAG.

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.

Honor Code:

In addition to the limits set in the posted course contract, students are expressly forbidden from providing the following information to the CS 1044 listserv, newsgroup or by private e-mail to other students in this course:

- any explanation of **how** to code the calculation of the frame scores.
- any explanation of **how** to code the calculation of the total frame scores.
- any explanation of **how** to code the determination of the low and high scores.

It **is** permissible to post an input and corresponding output file to the listserv. Be advised such postings may be incorrect.

Pledge:

Each of your submissions to the NAG 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 Automated Grader.
```

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