Reading to Input Failure, Simple File I/O, Arithmetic Calculations, Using Standard Functions

This programming assignment uses many of the ideas presented in sections 3 and 4 of the course notes, so you are advised to read those carefully. Read and follow the following program specification carefully. This program is considerably simpler than the payroll program from Project 1, but don't underestimate it, especially if you've not done much programming before.

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: the Free-Falling Body

Suppose an object is dropped from a point at a known distance above the ground and allowed to fall without any further interference; for example, a skydiver leaps from an airplane flying at an altitude of 5000 feet. An object that falls in this manner is said to be in free fall.

The altitude of the object, its distance above the ground, will change as time passes. The velocity at which the object falls will also change (increase, actually) as time passes. Let H represent the altitude of the object and V represent its velocity. It's also clear, intuitively, that the altitude of the object depends on how its velocity changes; in fact, there's a nice mathematical relationship between the altitude and the velocity involving a definite integral. But we're interested only in the velocity of a free-falling object.

It turns out that the velocity of a free-falling object is modeled by the formula

$$V = \sqrt{\frac{m \times g}{k}} \tanh\left(\sqrt{\frac{g \times k}{m}} \times t \right)$$

where:

- *m* is the mass of the object
- *g* is the acceleration due to gravity
- *t* is the number of seconds the object has been in free fall
- *k* is a constant that is determined by the shape of the object and the density of the air through which it falls
- tanh( ) is the hyperbolic tangent function

For simplicity, we will call k the drag constant. We also assume that the distance the object falls is small enough that the density of air doesn't change appreciably from its starting point down to the last point of interest.

For example, suppose a skydiver weighing 130 lb exits her plane and falls for 3 seconds. Her velocity could be computed as follows. First of all, weight equals mass times *g*. Second, in English units, *g* is about 32.2. So, her mass m would be about $130/32.2 \approx 4.037$. Applying the given formula:

$$V = \sqrt{\frac{4.037 \times 32.2}{0.005}} \tanh\left(\sqrt{\frac{32.2 \times 0.005}{4.037}} \times 3 \right) \approx 161.24 \tanh(0.599) \approx 86.48$$

giving her a velocity of about 86.48 feet per second or about 58.96 miles per hour, three seconds after her jump began.

This problem is adapted from an exercise in *Calculus*, 9th Ed., by Thomas and Finney, page 527. It's not necessary, or even useful to understand the derivation of the formula given above to complete this assignment. However, if you're interested, see the Appendix for Project 1 on the website.

Input file description and sample:

Your program **must** read its input from a file named `fall.dat` — use of another input file name will result in massive deductions. The first line of the input file contains column labels which should be ignored. Each remaining line of the input file will contain three numbers, separated by whitespace:

- the weight of an object, assumed to be in pounds and greater than zero
- the drag coefficient of that object, assumed to be greater than zero
- the time the object has been falling, assumed to be in seconds

You may assume that all the input values will be logically correct (no negatives, for instance). For instance:

```
Weight    k          Time
572.900   0.008      26.3
120.400   0.006      23.1
  0.048   0.040      17.8
127.500   0.236       6.7
157.100   0.024      12.7
550.300   0.081      24.6
535.000   0.165       6.1
240.700   0.040      22.4
  3.330   0.010      16.6
290.100   0.026      19.4
```

The input values won't always have the nice patterns shown in this example so don't make any unjustified assumptions. Note that you must **also not make any assumptions** about the number of lines of data in the input file. Your program must be written so that it will detect when it's out of input and terminate correctly. A technique for this will be discussed in class and was used in the payroll program from project 1; see also slides 4.20 – 4.21 in the course notes.

What to Calculate:

You will write a program which will read each line of the input file and use the given values to compute:

- the velocity of the object that data line describes, in feet per second
- the velocity of the object that data line describes, converted to miles per hour

To perform these calculations, you will need to use the formula given above. Use the value 32.2 for *g*. The formula requires taking square roots and also using the hyperbolic tangent function. Fortunately, both of those may be found among the standard library functions in C++. The square root function is called `sqrt( )`; it takes one parameter of type double and returns the square root of that parameter. So, the statement:

```
double aRoot = sqrt(3.24);
```

would assign the value 1.8 to the variable `aRoot`. The hyperbolic tangent function is called `tanh( )`; it also takes one parameter of type double and returns the hyperbolic tangent of that parameter. So, the statement:

```
double hypTangent = tanh(2.72);
```

would assign the value 0.9914 to the variable `hypTangent` (approximately). You must use a compiler directive to include the standard header file `math.h` in order to use these functions.

Output description and sample:

The first line of your output should include your name only; the second line should include the title "Falling Bodies" only; the third line should be blank; the fourth line should display the column labels shown below; the fifth line should consist of some delimiting symbol to separate the column labels from the body of the table.

Next your output file will contain a table, with one line of output for each line of numeric data in the input file. Each line of the table should list the object's mass (not its weight), its coefficient of drag, the amount of time it's been falling, and its velocity in both feet per second and miles per hour. There should be a line of delimiters immediately after the last line of the table body.

The values for mass and the coefficient of drag should have precision 3 (i.e., show three digits after the decimal), while the velocity values should have precision 2 and the time should have precision 1.

Your program must write output data to a file named `fall.out` — use of any other output file name will result in a massive deduction of points. The sample output file shown below corresponds to the input data given above:

```
Bill McQuain
Falling Bodies

  mass     drag       time       fps       mph
-----------------------------------------------
17.792     0.008      26.3    266.65    181.81
 3.739     0.006      23.1    141.65     96.58
 0.001     0.040      17.8      1.10      0.75
 3.960     0.236       6.7     23.24     15.85
 4.879     0.024      12.7     80.90     55.16
17.090     0.081      24.6     82.42     56.20
16.615     0.165       6.1     56.83     38.75
 7.475     0.040      22.4     77.57     52.89
 0.103     0.010      16.6     18.25     12.44
 9.009     0.026      19.4    105.63     72.02
-----------------------------------------------
```

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

- You must use variables of type `double` for all the real numbers. If you use `float` variables, your answers may not be as accurate as needed.
- All decimal values must be printed to show the same number of decimal places as in this sample.
- You must use the specified header and column labels, and include your name in the first line as shown.
- You must arrange your output in neatly aligned columns, with a label identifying the contents of each column. 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 line of hyphens marking the end of the table.

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 — if you don't have a copy I strongly suggest you read the on-line edition (available from the course web page). Some specifics:

- 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.
- Use named constants instead of variables where appropriate.
- 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.
- 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.

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:

- 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. There's not much point in worrying about processing the data further until you know you're reading it correctly.
- Second, add the code to compute the mass and change your output code to print that to the output file, instead of the weight. Also print the specified header to the output file. Verify that the results are correct; if not, determine why and fix the problem.
- Third, add the code to compute the velocity in feet per second and print that to the output file. Verify that the results are correct; if not, determine why and fix the problem.
- Fourth, add the code to convert the velocity from feet per second to miles per hour and print that to the output file. Verify that the results are correct; if not, determine why and fix the problem.

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 Grader.

Hints:

This program requires that you know how to manage file-oriented input/output operations — the slides and the text provide good examples and guidelines. Your program must read lines of input data until there is no more data to be processed. You may want to use the same logical design as in the payroll program from Program 1.

You'll have to use **manipulators** to manage the formatting of your output. (Use of `printf` is strictly forbidden, you must use C++ streams for I/O in the programs for this course.) Read the discussion on slides 4.16 – 4.19 carefully, there are important clues there. The payroll program provided for Programming Project 1 also shows how this is done.

You also have to do some mathematical calculations that go beyond mere arithmetic. The C++ language includes most of the standard mathematical functions, including a square root function and a hyperbolic tangent function. To use them, you need to add another #include at the beginning of your program: `#include <math.h>`

Testing:

At minimum, you should be certain that your program produces the output given above when you use the given input file. However, verifying that your program produces correct results on a single test case does not constitute a satisfactory testing regimen. You should make up and try additional input files as well; of course, you'll have to determine by hand what the correct output would be.

Pledge:

Beginning with Project 2, 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.