

## Implementing User-defined Functions

This assignment involves rewriting your solution to Project 2 to make use of a variety of user-defined functions. Once you've finished this, you should understand the various parameter-passing options and how to use the return value from a function.

The problem statement is exactly the same as for Project 2. The only changes are in the sections labeled Function Requirements and Incremental Development.

### The Problem:

### 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*, 9<sup>th</sup> 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 at the end of this specification.

## Sample Input:

Your program **must** read its input from a file named `Input.txt`. The first two lines of the input file contain column labels which should be ignored. Each remaining line of the input file will contain some string data and three numbers, formatted as follows:

```
<last name> ", "<first name>' \t' <weight><spaces><drag coeff><spaces><time>' \n'
```

Key:

- The first and last names are arbitrary character strings (no commas). The only guarantees are that neither will contain commas or tabs, and their total length won't exceed 25 characters.
- The weight, drag coefficient and time are all positive decimal numbers.
- Weight is given in pounds, and time is given in seconds.

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

Name	Weight	k	Time
de Bergerac, Cyrano	306.50	0.02	6.3
Ramotswe, Precious	163.30	0.07	7.3
Groan, Titus	194.30	0.02	36.8
Wu, Louis	190.90	0.06	49.6
de Bergerac, Cyrano	179.10	0.07	28.4
Flagg, Randall	366.30	0.02	12.8

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 has been discussed in class and was used in the code given for the first two projects.

## 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 `<cmath>` in order to use these functions.

**Sample Output:**

The first line of output should display the column labels shown below, and the second line should delimit the table of calculated values.

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 name of the person, the person's mass (not their weight), the person's coefficient of drag, the amount of time the person has been falling, and the person's velocity in both feet per second and miles per hour.

Note that the name has been reformatted in the output, so that the first name is listed, followed by a space and then the last name.

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 `Output.txt`. The sample output file shown below corresponds to the input data given above:

Name	Mass	Drag	Time	fps	mph
Cyrano de Bergerac	9.519	0.020	6.3	114.79	78.27
Precious Ramotswe	5.071	0.070	7.3	48.29	32.93
Titus Groan	6.034	0.020	36.8	98.56	67.20
Louis Wu	5.929	0.060	49.6	56.41	38.46
Cyrano de Bergerac	5.562	0.070	28.4	50.58	34.49
Randall Flagg	11.376	0.020	12.8	134.72	91.86

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.
- You must arrange your output in neatly aligned columns, with a label identifying the contents of each column. Note that while the Curator 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. See the *Programming Standards* page on the course website. 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 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.

## Function Requirements:

You must implement the following functions. The supplied code on the website shows how the functions might be used in your solution. You are free to make as much or as little use of that code as you like, but you must satisfy the requirements below. The parameter lists and return types are not shown; you must decide what they should be.

```
printTableHeader(. . .)
```

Writes the column labels and the row of hyphens to the output file.

```
printTableRow(. . .)
```

Writes one row of the body of the table to the output file; doesn't perform any calculations itself.

```
printTableFooter(. . .)
```

Writes the row of hyphens at the bottom of the table to the output file.

```
readTableRow(. . .)
```

Reads all the values from one line of the table in the input file; doesn't perform any calculations.

```
calculateMass(. . .)
```

Calculates the mass of an object, given the weight of the object; uses `return` to report the value.

```
calculateFPS(. . .)
```

Calculates the velocity of the skydiver in feet per second; uses `return` to report the value.

```
calculateMPH(. . .)
```

Calculates the velocity of the skydiver in miles per hour; uses `return` to report the value.

## Incremental Development:

In this case, you should already have a working solution, so a lot of the problem-solving aspects are already taken care of.

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:

- The first and third output functions are quite simple. Modify your working solution to implement them first.
- Implement the function to compute the mass of the skydiver and incorporate that into your solution. This is the simplest of the three functions that calculate and `return` a value; once you've got this one working, it will be easier for you to implement the other two.
- Implement the function to calculate the velocity of the skydiver in feet per second, and incorporate that into your solution.
- Implement the function to calculate the velocity of the skydiver in miles per hour, and incorporate that into your solution. (This might use the results of the previous function.)
- Add the function to write one line of the output table and incorporate it into your solution. This one is somewhat more complicated since it involves more communication issues than the other output functions, but you will have dealt with similar issues when you implemented the three calculation functions.
- Finally, implement the function to read one line of the input table, and incorporate that into your solution. This is the most complicated of the required functions. It has some interesting communication issues that don't arise in any of the other required functions.

Make sure you've documented your functions as specified on the course website. For each function you write, you should have a header comment similar to the ones shown in class and on the website. And, of course, the bodies of the functions should be commented.

## 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 <cmath>`

## Testing:

At minimum, you should be certain that your program produces the output given above when you use the given input files. Since this uses exactly the same files as Project 2, no new ones are supplied. 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.

**Evaluation:**

Since this assignment doesn't add any new operational features, relative to Project 2, your score from the Curator will make up only a small part of your final score. The TA will assess your best submission to see if you've met the specific requirements for functions that were stated above, and produce a deduction (ideally 0, but possibly as much as 100) that will be applied to your score from the Curator.

**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 latest of those will be graded.**

**Pledge:**

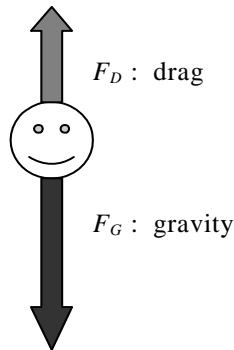
Each of your submissions to the Curator must be pledged to conform to the Honor Code requirements for this course. Specifically, you **must** include the following pledge statement in the header comment for your program:

```
//      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 System.  
//  
//      <Student Name>
```

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

**Appendix:**

Suppose an object with mass  $m$  is dropped from a certain initial altitude and allowed to fall freely; i.e., the only forces acting on it after its release are gravity and the resistance of the air through which it falls:



First, we need a little physics. The force exerted by gravity equals the mass of the object times the gravitational acceleration constant  $g$ :

$$F_G = m \times g$$

Assuming constant air density (reasonable if the distance fallen is not too great) and a stable orientation of the object as it falls, the force of air resistance on the object is proportional to the velocity of the object squared:

$$F_D = -k \times v^2$$

where  $k$  is a constant that depends on the shape of the object and the density of the air through which it falls and  $v$  is the velocity of the object. The force is negative since it acts in the opposite direction to the object's motion.

Now, Newton's Second Law of Motion says that the total force acting on an object equals the mass of the object times its acceleration  $a$ , so we have:

$$F = F_G + F_D \quad \text{or} \quad m \times a = m \times g - k \times v^2$$

Now we need a little calculus. The acceleration and velocity are both functions of time  $t$ , and the acceleration  $a$  is the derivative of the velocity  $v$ . So we have:

$$m \times \frac{dv}{dt} = m \times g - k \times v^2 \quad \text{and also} \quad v(0) = 0$$

since the object isn't moving when it is released (at time 0). This is an example of a separable nonlinear differential equation with initial condition (covered in Math 1206 and 2214). It is possible to solve for a function  $v$  that satisfies both the differential equation and the initial condition.

To start, express in differential notation and do some algebra to separate the variables:

$$\frac{mdv}{mg - kv^2} = dt$$

Now antidifferentiate each side of the equation. This requires a  $u$ - $du$  substitution on the left hand side:

$$\int \frac{m dv}{mg - kv^2} = \int dt$$

Let

$$u = \sqrt{\frac{k}{mg}} \times v \quad \text{then} \quad du = \sqrt{\frac{k}{mg}} dv$$

This substitution transforms the left-hand integral above in the following way:

$$\int \frac{m dv}{mg - kv^2} = \frac{1}{g} \sqrt{\frac{mg}{k}} \int \frac{du}{1-u^2} = \sqrt{\frac{m}{kg}} \tanh^{-1} \left( \sqrt{\frac{mg}{k}} v \right) + C$$

The right-hand integral is trivial, and we obtain:

$$\sqrt{\frac{m}{kg}} \tanh^{-1} \left( \sqrt{\frac{mg}{k}} v \right) + C = t$$

Since the object is dropped from a stationary position at time  $t = 0$ , we have  $v = 0$  when  $t = 0$ . Substituting that into the equation above yields that  $C = 0$ . Then, performing algebra to solve for velocity  $v$  as a function of time  $t$  yields:

$$v = \sqrt{\frac{mg}{k}} \tanh \left( \sqrt{\frac{gk}{m}} t \right)$$