

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 about the same degree of difficulty as the map mileage program from Project 1, but don't underestimate it, especially if you've not done much programming before. You should be able to use the same data processing approach from program 1.

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:

Rainbow Height

This program is taken from “*Programming and Problem Solving with C++*”, second ed. N. Dale, C. Weems & M. Headington, © 2000, Jones and Bartlett Pub.s, pages 208-209. How tall is a rainbow? Because of the way in which light is refracted by water droplets, the angle between the level of your eye and the top of a rainbow is always the same. If you know the distance to the rainbow, you can multiply it by the tangent of that angle to find the height of the rainbow. The magic angle is 42.3333333 degrees. The C++ standard (math) library works in radians, however, so you have to convert the angle to radians with this formula:

$$\text{radians} = \text{degrees} \times \frac{\pi}{180}$$

where π equals 3.14159265.

Through the header file `<math.h>`, the C++ standard library provides a tangent function named `tan`. This is a value-returning function that takes a floating-point argument and returns a floating-point result:

```
x = tan(someAngle);
```

If you multiply the tangent by the distance to the rainbow, you get the height of the rainbow.

Sometimes you can see a second, fainter rainbow outside a bright rainbow. This second rainbow has a magic angle of 52.25 degrees.

First write a modular decomposition in outline form and then write a C++ program to read from each line of an input file a single floating point value — the distance to the rainbow — and compute the height of the main rainbow, height of the secondary rainbow, and to echo the distance to the main rainbow. The values are to be output in a labeled tabular format as described below in the output file section. The decomposition outline should be included at the top of the source code as comments following the honor code pledge.

Input file description and sample:

Your program **must** read its input from a file named `rainbow.dat` — use of another input file name will result in massive deductions. The first line of the input file contains a label, which should be ignored. Each remaining line of the input file will contain one floating-point number, (the distance to a rainbow in miles):

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

```
Miles to rainbows:
1.500
2.333
1.250
3.666
0.875
```

Note that you must **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 map miles program 1; see also slides 4.20 – 4.22 in the course notes.

Output description and sample:

The first line of your output should include your name only; the second line should include the title “Rainbow Heights (in miles)” 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 height of the main rainbow, the height of the secondary rainbow and the distance to the rainbow. There should be a line of delimiters immediately after the last line of the table body.

The values for all output values should have precision 4, i.e., show four digits after the decimal.

Your program must write output data to a file named `heights.dat` — 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:

| | | |
|----------------------------|-----------|----------|
| Dwight Barnette | | |
| Rainbow Heights (in miles) | | |
| | | |
| Rainbow | Secondary | Distance |
| ----- | | |
| 1.3665 | 1.9373 | 1.5000 |
| 2.1253 | 3.0131 | 2.3330 |
| 1.1387 | 1.6144 | 1.2500 |
| 3.3397 | 4.7347 | 3.6660 |
| 0.7971 | 1.1301 | 0.8750 |
| ----- | | |

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 Curator doesn't deduct points for horizontal alignment, the TA who may grade 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 it is strongly suggested 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 map mileage code from Project 1 is a good guide.

Your submission that receives the highest score may 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 map mileage 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 main rainbow height and change your output code to print that to the output file. 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 height of the secondary rainbow 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 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. Programming Project 1 also shows how this is done.

The C++ language includes most of the standard mathematical functions. 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:

As with Project 1, 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 automated grader.
```

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

To learn more about rainbows see:

- http://astro.geoman.net/us/dossiers/8/arc_en_ciel.html
- <http://www.unidata.ucar.edu/staff/blynds/rnbw.html>