

Arrays, Structs, Sorting, Count-controlled Reading

This programming assignment uses many of the ideas presented in sections 3 through 12 of the course notes, so you are advised to read those notes carefully as well as the following program specification. This program will be worth 12% of your final grade. Unlike previous assignments, there will be no specific cap on the possible deductions for style.

The Program Specification:	Rain
-----------------------------------	-------------

You must write a program that will take in daily rainfall amounts for a given city over the course of a year and provide output on a monthly basis.

Struct

Your program must utilize an array of structs to maintain information on monthly rainfall amounts. Your struct must, at minimum, include:

- A string representing the name of the month
- An integer representing the total number of days during the month that it rained
- A double representing the amount of rainfall received during the month
- A double representing the maximum daily rainfall for a single day during the month
- A double representing the minimum **non-zero** rainfall for a single day during the month

Failure to use an array of structs as described will result in a score of zero for this assignment.

Input file descriptions and samples:

Your program **must** read the postal rates from a file named `rainfall.data` — use of other input file names will generally result in a score of zero.

The first line of the input file will contain a string (including whitespace) giving the city and state of the rainfall measurements. The next line will contain an integer indicating the number of subsequent lines that contain rainfall data. Note – this means that you will **not** be reading until input failure, but rather a finite number of times. Each line of rainfall data will contain three columns, separated by tabs

- A string giving the month's name
- An integer giving the day of the month
- A double giving the rainfall for that day

Unfortunately, the file contains some errors, in the form of invalid month/day combinations – for example, September 31. Before adding the rainfall to the month's total, you must make sure that the day is valid for the given month (assume that it is not a leap year). If the day is invalid, simply ignore the data and move on to the next line.

After the specified number of data lines, there will be a final line containing a string – either "Total" or "Days". This string indicates how the output is to be sorted (more on that later).

A (simple) sample input file:

Harrisonburg, VA			
20			
January	10	0.35	
April	13	0.11	
March	21	1.10	
February	29	0.50	
July	16	0.20	
December	25	0.35	
August	3	1.20	
September	2	0.14	
April	14	0.03	
April	15	0.40	
January	11	0.15	
April	10	0.18	
March	31	1.00	
February	20	0.40	
July	17	0.20	
December	21	0.15	
August	9	.50	
September	31	0.11	
April	19	0.12	
April	20	0.43	
Total			

What to Calculate:

For each month your program must calculate the following:

- Total rainfall
- Total number of days that it rained
- Maximum rainfall for a single day
- Minimum **non-zero** rainfall for a single day
- Average amount **per rainfall** (that is, the average amount received in a given rainfall, not the average over the entire month)

After all the data is read, the array should be sorted in **descending** order, based on either the total rainfall or the number of days that had rain, as indicated by the last string on the last line. If there is a tie, the month that is chronologically first should appear first (if you set up your array based on calendar order, this should be the default based on the given sorting algorithms). For sorting, you can use either of the sorting algorithms given in the course notes. Please note that there will be some minor changes required in order for the algorithms to work with this specific program. For those who have been coming to class, we've covered all of the necessary changes in our examples (you just need to be able to recognize and apply them). Remember, there are **two** types of sorts that can be done – you may need multiple sort functions.

Finally, the total rainfall for the entire year must be computed.

Output description and sample:

Your program must write its output data to a file named `rainout.data` — use of any other output file name **will** result in a score of zero. A sample output file produced from the sample input files above is shown below:

```

Programmer:  Chris Knestrick
Rain
Harrisonburg, VA

  Month          Total      Days      Max      Min      Avg
-----
   March         2.10         2        1.10     1.00     1.05
   August        1.70         2        1.20     0.50     0.85
   April         1.27         6        0.43     0.03     0.21
   January       0.50         2        0.35     0.15     0.25
   December     0.50         2        0.35     0.15     0.25
   February     0.40         1        0.40     0.40     0.40
   July          0.40         2        0.20     0.20     0.20
   September    0.14         1        0.14     0.14     0.14
-----
Total:          7.01

```

As usual, the first line of your output should identify you by name, as shown. The second line should include the title “Rain” only. The third line should contain the city and state given in the input file. The fourth line should be blank. The fifth and sixth lines should contain the specified column labels and a row of delimiters to mark the top of the table.

Next your output file will contain a sorted table, with a line of output for each month that had rain. If a month had no rain, it should not appear in the table. Each line should contain the name of the month, the total rainfall for that month, the maximum and minimum rainfall for a single day, and the average amount per rainfall.

After the last line of the table, print a line of delimiters and then display the combined rainfall for the entire year.

You are not required to use the exact horizontal spacing shown in the example above, but your output must satisfy the following requirements:

- You must use the specified header and column labels, and print a row of delimiters before and after the table body, as shown.
- You must arrange your output in neatly aligned columns. Use spaces and `setw()`, not tabs to align your output.
- You must use the same ordering of the columns as shown here, and print the rainfall with precision two.

Programming Standards:

You'll be expected to observe good programming/documentation standards. All the discussions in class about formatting, structure, and commenting your code should be followed.

Documentation:

- You must include the honor pledge in your program header comment.
- Your header comment must describe what your program does.
- 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.
- Precede every function you write with a header comment. This should explain in one sentence what the function does, then describe the logical purpose of each parameter (if any), describe the return value (if any), the name of the calling function(s), the name of any functions that are called, and state reasonable pre- and post-conditions.
- You must use indentation and blank lines to make control structures like loops and if-else statements more readable.

Coding:

- Use named constants instead of variables where appropriate.
- Use `double` variables (with two decimal places of precision) for all rainfall values, and `int` variables for all other numeric input data.
- 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.
- You must write at least seven functions, besides `main()`. For reference, my solution uses 13.
- The definition of `main()` must be the first function definition in your source file. You may use file-scoped function prototypes, file-scoped constants, and type definitions.
- **You may not use any global variables – failure to follow this guideline will result in major deductions.**
- Function parameters should be passed appropriately. Use pass-by-reference only when the called function needs to modify the parameter.

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.

Testing:

Obviously, 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.

At minimum, you should test your program on **all** the posted input/output examples given along with this specification. The same program that will be used to test your solution generated those input/output examples. You could 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:

Each of your submissions to the EAGS 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.