CMSC 216 Project 1 Fall 2021

Grades Calculator **Due date posted on Piazza**

1 Overview

For this project you will write a C program that reads assignment scores and computes numeric grades and statistical information. The objective is to practice functions and arrays.

1.1 Obtaining project files

Copy the folder project1 available in the 216public projects directory to your 216 directory. This folder has the file (.submit) that will allow you to submit your project.

This project description can be found in the 216public project_descriptions directory.

1.2 Late submission

You can submit one day late with a 12 point penalty.

1.3 Good Faith Attempt

Remember that you need to satisfy the good faith attempt for every project in order to pass the class (see syllabus). Check the course web page for the good faith attempt deadline; **it is not at the end of the semester**.

1.4 Individual effort

IMPORTANT: You must implement this project individually. Treat it like a take-home exam. Do not work with others. Doing so is an academic integrity violation.

1.5 Protect your work

Please do not leave your computer unattended and unlocked. If you do, someone can execute submit on your project folder and steal all your code.

1.6 Debugging Guidelines

Before looking for help during office hours or posting a message in Piazza, make every attempt to debug your program. Make sure you are familiar with the information provided at

`http://www.cs.umd.edu/~nelson/classes/resources/cdebugging/`

2 Academic integrity

Please **carefully read** the academic honesty section of the course syllabus. We take academic integrity matters seriously. Please do not post assignment solutions online (e.g., Chegg, github) where others can see your work. This can lead to an academic integrity case where you will be reported to the Office of Student Conduct.

3 Specifications

3.1 Input Data

Your project will read information about class assignments and compute a numeric score. The data provided consists of:

- Number of assignments (N for short)
- Points penalty per day late (P for short)
- Number of assignments to drop (X for short)
- Whether statistical information is to be generated (W)
- N lines of assignment information, each consisting of assignment number, score, weight, and days late.

The data format is:

P *X* *W*
N
Assignment_Info #1 // line with assignment number, score, weight (percentage), days late
Assignment_Info #2
...
Assignment_Info #*N*

You can assume that input *W* is a character, and every other input (*P*, *X*, *N*, and each entry in Assignment_Info) is an integer.

The following is an example of the data your program will process:

```
10 2 Y
4
2, 82, 40, 1
1, 91, 40, 0
4, 84, 10, 3
3, 73, 10, 3
```

The above data says there is a 10 points penalty per day late, 2 assignments are to be dropped, statistical information is to be generated (Y), and there are 4 assignments in total. The first Assignment_Info line says that for assignment number 2, the student's score is 82, the assignment represents 40% of the student's grade, and it was submitted 1 day late.

3.2 Processing

Your program will compute the numeric score by first dropping the *X* "lowest valued" (defined below) assignments, and then taking into account days late, penalty per day late, and the weight associated with the assignments. If statistical information is requested, the mean and standard deviation will be computed. For example, for the above data, your program should generate the following output:

```
Numeric Score: 81.5000
Points Penalty Per Day Late: 10
Number of Assignments Dropped: 2
Values Provided:
Assignment, Score, Weight, Days Late
1, 91, 40, 0
2, 82, 40, 1
3, 73, 10, 3
4, 84, 10, 3
Mean: 65.0000, Standard Deviation: 18.2346
```

Regarding the data and processing:

1. Use double as your floating-type (e.g., double tmp, double numeric_score).
2. The assignment number is an integer between 1 and *N*. You can assume *N* is at least 1.
3. Assignment_Info lines can be provided in any order of assignment numbers. However they must be printed in increasing order of assignment number. You can assume there are *N* Assignment_Info lines with assignment numbers 1 through *N* (so no missing or duplicated assignment numbers).
4. An assignment score is an integer value between 0 (inclusive) and 100 (inclusive). You can assume we provide valid scores.

5. The weight is an integer value between 0 (inclusive) and 100 (inclusive). You need to check the that sum of weights for all assignments add to 100. If after reading the data the total weights do not add to 100, your program will generate the error message **ERROR: Invalid values provided** and the program will terminate. The message should be printed on a line by itself.
6. Your program should remove the X lowest-valued assignments before performing any numeric score computation. The **value** of an assignment is defined as the assignment score $\times$ the assignment weight. You can assume that X will be in the inclusive range $0..N-1$. Note that the number of days late and the penalty per day IS NOT used in order to decide what assignment to drop. If two assignments have the same value (score $\times$ weight), drop the one with the lowest assignment number.
7. The numeric score is a value between 0 (inclusive) and a 100 (inclusive). For the numeric grade computation, adjust the score for an assignment based on the number of days late and the points penalty per day late. An assignment score is set to 0 if the assignment's score becomes less than 0 after the late penalty is applied. This adjusted score along with the assignment's weight will allow you to compute the numeric score.
8. If any assignment is dropped, the sum of assignment weights will, nearly always, not correspond to 100.
9. For W equal to either 'Y' or 'y', print statistical information. Any other character for W means that no statistical information is to be generated.
10. For the computation of the mean and standard deviation, apply the late penalty but do not drop any assignments (even if there was a assignment drop request). In addition, do not use weights for the computation of mean and standard deviation.
11. You don't need to implement a sorting algorithm. The assignment number can be used to generate the index where an assignment should be.

3.3 Functions Requirements

1. You must have at least two other functions in addition to main.
2. One of your functions must take at least one array as a parameter.

3.4 Other

1. Use `%5.4f` as the format for a float.
2. The maximum number of assignments (N) in the input is 50.
3. IMPORTANT: You may not use the following C constructs. If you do you will lose significant credit.
   a. C structures.
   b. Global variables.
   c. Two-dimensional (2D) arrays.
   d. An array of pointers to arrays. We consider them 2D arrays.
   e. Dynamic memory allocation (e.g., malloc, calloc).
4. To use the sqrt function or any function from the math library, include the file <math.h> and compile with the -lm option (e.g., gcc grades.c -lm). You can find additional information about the math library by using the linux man pages ("man sqrt" on grace).
5. You must name your C file grades.c, otherwise it will not compile on the submit server.
6. You may not use the qsort function.
7. If you decide to use the indent tool make sure you define the appropriate alias as specified in the indent_utility_info.txt file that can be found in the grace info folder.
8. You need to use #define (e.g., instead of 50 use #define to define a symbolic constant).
9. A standard deviation calculator can be found at:

   http://www.mathsisfun.com/data/standard-deviation-calculator.html

3.5 Compilation

Make sure your gcc alias has been set as defined at

http://www.cs.umd.edu/~nelson/classes/resources/setting_gcc_alias/

3.6 Execution

We will use input and output redirection in order to execute your program. For example, assuming data is present in the public01.in file, we will run your program as follows: a.out < public01.in. You can compare the results of your program against expected results by using the diff command. Information about the diff command can be found at http://www.cs.umd.edu/~nelson/classes/resources/cdebugging/diff/. Make sure you remove output files created while using output redirection. If your code has bugs (e.g., infinite loop) you may create large files that impact grace's quota.

4 Grading Criteria

Your project grade will be determined with the following weights:

Results of public tests	20%
Results of secret tests	50%
Code style grading	30%

4.1 Style grading

For this project, your code is expected to conform to the following style guidelines:

- Your code must have a comment at the beginning with your name, university ID number, and UMD Directory ID (i.e., your username on Grace).
- Avoid lines longer than 80 columns. You can check your code's line lengths using the `linecheck` program in grace. Just run "`linecheck filename.c`" and it will report any lines that are too long.
- Do not use global variables.
- Feel free to use helper functions for this project.
- Each function must have, at a minimum, a comment describing its purpose and operation. If you use a complicated algorithm to implement a function, you definitely need an extra comment explaining the complicated steps of your algorithm.
- Follow the C style guidelines available at:

  http://www.cs.umd.edu/~nelson/classes/resources/cstyleguide/
- TAs will look at each of the following items while grading your style:

  1. Good names for variables, constants, and functions. The only place where a variable name with a single character is acceptable is the iteration variable of a for loop; otherwise you need to have descriptive variable names. If something represents the mean, called it mean, not m.
  2. Good indentation (3 or 4 spaces). Use a proper editor (e.g., emacs, vim) that assists with indentation. TAs will check your indentation with the emacs editor.
  3. If variables have the same type declare them on the same line if possible.
  4. Leave one blank line between variable declarations and the first line of code in a function.
  5. Consistent style (e.g., use of curly brackets). Opening brace must be on the same line as conditional or function.
  6. Do not use CamelCase. Use underscores for multi-word variables. For example, use hot_water_temperature instead of hotWaterTemperature.

7. Define values as constants when needed (do not use variables for constants). Do not use numbers in your expressions if those numbers have a special meaning (e.g., 3.1415); instead define constants (e.g., using #define) for them.
8. #defined constants must be in uppercase (e.g., ALL_CAPS).
9. In your code you should leave one blank space between operators (e.g., x = 5 + 7).
10. Leave one space after a comma.
11. Use braces; avoid loops and conditionals without them.

5 Testing

Make sure you test your code with different input data sets (sets different from the ones we have provided as public input). You can take one of the provided input files (e.g., public01.in), update it with different values, and use input redirection to generate output. You will need to manually check your results (you may not compare your results with the results of another student's code). To come up with test cases read the project description carefully. It is best if you think of test cases as you implement your project.

6 Submission

6.1 Deliverables

For this project, the only file that we will grade is `grades.c` (which **must** be the name of your source file).

6.2 Procedure

You can submit your project by executing, in your project directory (project1), the **submit** command. This will prompt you for your UMD Directory ID and password, and if all goes well, inform you of a successful submission. You should then log onto the submit server (there is a link on the course website) and check your public test results to be sure that things worked as you expected.

You need to execute submit in the project1 directory, as we gave you a .submit file that allows you to submit. If you did not copy the project1 folder we have provided, you will not be able to submit (you will be missing the necessary .submit file).

Immediately after copying the project1 folder, try to submit your project (even if you have not started). Do not wait until the day the project is due in order to try the submission process.

6.3 Possible problem with submit command

If you try to submit your project in grace, and you get the error:

"Exception in thread 'main' java.lang.OutOfMemoryError: unable to create new native thread"

then close all terminals windows except one, and try to submit again.