

1 Overview

For this project you will write a C program that reads assignment scores and computes numeric grades and statistical information. There are two deadlines associated with the project:

- Mon Feb 10 11:30pm: Your code must pass the first two public tests (public01, public02). That is the only requirement for this deadline. We will not grade the code for style. This first part is worth .5% of your course grade (NOT .5% of this project grade). You can submit late for this part.
- Thu Feb 13 11:30pm: Final deadline for the project. You can submit late.

2 Objectives

To practice functions and arrays.

3 Warning

Do not leave your computers unattended. If you don't lock your computer, someone can execute submit on your project folder and steal all your code.

3.1 Good Faith Attempt

Remember that you need to satisfy the good faith attempt for every project in order to pass the class (see syllabus). The **deadline for the good faith attempt is not the end of the semester**. Make sure you check the course web page for deadline information.

3.2 Debugging Guidelines

Make sure you are familiar with the information provided at

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

3.3 Obtaining project files

To obtain the project files, copy the folder project1 available in the 216 public directory to your 216 directory. In particular, that folder contains the file (.submit) that allows you to submit your project. The project description can be found in the project_descriptions directory.

4 Specifications

4.1 Input Data

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

- Number of assignments (referred to as **na** below)
- Points penalty per day late
- Number of assignments to drop (referred to as **nd** below)
- Whether statistical information is to be generated
- Assignments information (assignment number, score, weight, days late).

The data format is:

```
Points_Penalty_Per_Day_Late  Number_Of_Assignments_to_Drop  Stats_Y/N
Number_of_Assignments
Assignment Info #1
Assignment Info #2
...
Assignment Info #n
```

Each assignment info entry has the following information: assignment's number, assignment's score, assignment's weight (percentage), days late (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 indicates the following: 10 points penalty per day late; drop 2 assignments ($nd = 2$); generate statistical information (Y); and a total of 4 assignments are provided ($na = 4$). For assignment number 2, the student's score is 82, the assignment is 40% of the student's grade, and it was submitted 1 day late.

4.2 Processing

Your program will compute the numeric score after dropping the nd "lowest-valued" assignments and 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 will be a value between 1 and the number of assignments (na). Assume that the provided assignment numbers are valid.
3. The input data can list the assignments in any order; however your program must print them in order (by assignment number).
4. An assignment score is an integer value between 0 (inclusive) and 100 (inclusive). Assume that the provided scores are valid.

5. The weight is an integer value between 0 (inclusive) and 100 (inclusive). Check that the sum of the weights of the assignments is equal to 100. If after reading the input data, the sum of the weights do not equal 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 nd lowest-valued assignments before performing any numeric score computation. An assignment's **value** is defined as the assignment's score \times the assignment's weight. For a total of na assignments, the number of assignments to drop will be in the inclusive range $0 .. na-1$. Notice that number of days late and the penalty per day WILL NOT be used in order to decide what assignment to drop. If two assignments have the same value (score \times weight) the one with the lower assignment number will be dropped.
7. The numeric score will be a value between 0 (inclusive) and a 100 (inclusive). For the numeric score computation, adjust the score for an assignment based on the number of days late and the points penalty per day late. An assignment score will be 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 a 100.
9. Either 'Y' or 'y' will request statistical information. Any other character will indicate that no statistical information will be generated.
10. For the computation of the mean and standard deviation you need to apply the late penalty, but do not drop any assignments (even if there was an assignment-drop request). In addition, do not use weights for the computation of mean and standard deviation.
11. Points penalty per day late will be an integer value.

4.3 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.

4.4 Other

1. Use `%5.4f` as the format for a float.
2. The input data will have a maximum of 50 assignments ($na \leq 50$).
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 you need to 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 info folder in Grace.
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>

4.5 Compilation

Make sure your gcc alias has been set as defined at

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

4.6 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.

5 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%

5.1 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).
- No lines longer than 80 columns are allowed. 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, call it mean, not m.
 2. Good indentation (3 or 4 spaces). Use a proper editor (e.g., emacs, vi) 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.

6 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. Notice 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.

7 Submission

7.1 Deliverables

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

7.2 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 that contains the `.submit` file needed 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.

7.3 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.

8 Academic Integrity

Please see the syllabus for project rules and academic integrity information. All programming assignments in this course are to be written individually (unless explicitly indicated otherwise in a written project handout). Cooperation between students is a violation of the Code of Academic Integrity.