This fifth programming project will give you experience writing programs that involve for loops
and arrays. This programming project will use the same development process as in Programming
Projects #2, #3 and #4, and will be subject to the same rules and grading criteria, along with some
new criteria.

This specification will be less detailed than for previous programming projects. YOU ARE EX-
Pected TO KNOW HOW TO PERFORM BASIC TASKS WITHOUT HAVING TO BE
Told explicitly, based on your experience with previous programming projects.

To get full credit on this programming project, you MUST use for loops and dynamically
allocated arrays appropriately.

NOTE: The next programming project (#6) will be Based on PP#5, and you will use your
program for PP#5 as the starting point.

IMPORTANT IMPORTANT IMPORTANT IMPORTANT IMPORTANT IMPORTANT!!!

For this project, you are Absolutely Forbidden to have any user-defined functions. (We’ll
use those in PP#6.)

I. PROJECT DESCRIPTION

You’ve been hired to develop grade calculation software for a special section of CS1313, which
has the following categories of assignments:

- Quizzes
- Programming Projects
- Lab Attendance
- Final Exam

For this section of CS1313, there are a certain number of each type of assignment, and each type
of assignment is given a weight that governs how much that assignment type contributes to each
student’s overall grade. Specifically:

- 4 quizzes, together worth 10% of the overall percentage score, with maximum possible
  scores of 30 points (Quiz #1), 170 points (Quiz #2), 130 points (Quiz #3) and 115 points
  (Quiz #4);
- 3 programming projects, together worth 45% of the overall percentage score, with maximum
  possible scores of 250 points (PP#1), 1000 points (PP#2) and 2000 points (PP#3);
- 6 lab sessions, worth 10% of the overall percentage score (note that the lab score will be ex-
  pressed as a single value representing the total number of times that the student was marked
  “present” at lab during the semester, with 2 such opportunities per lab session);
- 1 final exam, worth 35% of the overall percentage score, with a maximum possible score of
  800 points.

The expression for calculating grades in CS1313 is a subset of the one appearing on the Spring
2017 CS1313 syllabus, at the top of page 4. But, for PP#5 there will be no in-class exam term
and no short programming assignment (CodeLab) term in the expression. Note that there are NO
FREE LABS in this version of the course.
As for letter grades, a student needs an overall percentage score of:

- at least 90% to get an A;
- at least 80% to get an B;
- at least 70% to get an C;
- at least 60% to get an D;
- otherwise, the student will get an F.

**NOTE:** In this program, there **WON’T** be any bonus points in any of the students’ scores.

**HOW TO COPY-AND-PASTE IN PUTTY**

In PP#5, you’ll find that using copy-and-paste will be **EXTREMELY** helpful.

Here’s how to copy-and-paste:

1. Using your mouse (or laptop touchpad), position the mouse cursor at the very left of the text that you want to copy.
2. Hold down the left mouse button.
3. To copy, drag the mouse cursor over the text that you want to copy, which will highlight it. The text is now copied (so no need to Ctrl-C or anything).
4. Position the green text cursor where you want to paste.
5. To paste, (single) click the right mouse button.

**II. PROGRAM DESCRIPTION**

Write a program to calculate grading information from assignment scores. The body of the program **MUST** be broken into **THREE** subsections, rather than the usual four subsections (there **WON’T** be a greeting subsection):

1. an input subsection;
2. a calculation subsection;
3. an output subsection.

Because of how data will be input (see below), **THERE WON’T BE A GREETING SUBSECTION.**

You are **ABSOLUTELY FORBIDDEN** to have:

- **ANY** calculations in the input subsection (and the only outputs should be idiotproofing error messages);
- **ANY** inputs or outputs in the calculation subsection;
- **ANY** inputs or calculations in the output subsection.

That is, the three subsections **MUST BE COMPLETELY SEPARATE, and MUST BE CLEARLY LABELED.**

For this project, **if** blocks can occur in any subsection of the execution section (body) of the program.
A. ARRAY DECLARATIONS
You **MUST** use **DYNAMIC** memory allocation and deallocation for **ALL** arrays. Therefore, **ALL** arrays **MUST** be declared as **POINTERS**. For example:

```c
float* quiz1_score = (float*)NULL;
```

B. INPUT SUBSECTION
The program will take its input from a data file, rather than from a user typing live at the keyboard (see part III, INPUT DATA FILES, below).

The input data will be in the following format:

1. the year, and the semester indicated by a numeric code (1 for spring, 2 for fall);
2. the number of students enrolled in the class;
3. for each student:
   a. the student ID number;
   b. the score for each of the assignments, in the following order:
      i. the quiz scores, in order (Quiz #1, Quiz #2, Quiz #3, Quiz #4);
      ii. the programming project scores, in order (PP#1, PP#2, PP#3);
      iii. the lab score;
      iv. the final exam score.

Several such data files will be provided, each representing an individual run. **YOU** should determine how to input the data **BY EXAMINING THE INPUT DATA FILES** (see **HOW TO FIND AND EXAMINE THE INPUT DATA FILES**, below).

Students will be identified by student ID, not by name.

**IMPORTANT IMPORTANT IMPORTANT IMPORTANT IMPORTANT IMPORTANT!!!**
Because of how the data will be input, **YOU WON'T PROMPT THE USER FOR THE INPUTS** (see **HOW THE DATA WILL BE INPUT**, below).

You **MUST** store the input data — that is, the assignment scores — in appropriate one-dimensional arrays. You are **ABSOLUTELY FORBIDDEN** to use multidimensional arrays in PP#5.

C. ALLOCATING ARRAYS
You **MUST** use **DYNAMIC** memory allocation and deallocation for **ALL** arrays. Any statically allocated arrays will be **SEVERELY PENALIZED**. Therefore, **ALL** arrays **MUST** be declared as **POINTERS**.

**IMPORTANT IMPORTANT IMPORTANT IMPORTANT IMPORTANT IMPORTANT!!!**
Note that **ALL** of the arrays **MUST** be allocated, at runtime, in the execution section, **IMMEDIATELY AFTER INPUTTING AND IDIOTPROOFING THE NUMBER OF STUDENTS**. In other words, once you have input and idiotproofed the number of students, you **MUST IMMEDIATELY** allocate all of the arrays. After allocating each array, the program **MUST** check whether the array was allocated successfully, and if not, the program **MUST** output a suitable, **UNIQUE** error message and then **MUST EXIT**.

For details on dynamically allocating and deallocating arrays, see the lecture slide packet “Array Lesson 2,” slides 26-33.
D. IDIOTPROOFING

YOU MUST IDIOTPROOF ANY input that needs idiotproofing, to make sure that it has an appropriate value. YOU are responsible for figuring out all of the possible cases of idiocy that could come up. ALL IDIOTPROOFING MUST BE COMPLETED BEFORE ANY CALCULATIONS ARE PERFORMED; that is, idiotproofing belongs in the input subsection.

Note that, for this programming project, you are ABSOLUTELY FORBIDDEN to use while loops for your idiotproofing; that is, upon detecting idiocy, the program MUST EXIT.

Idiotproofing error messages MUST be clear, complete English sentences that COMPLETELY AND UNAMBIGUOUSLY state the nature of the error. Thus, EACH ERROR MESSAGE MUST BE UNIQUE. For example, an error message might be:

ERROR: invalid Final Exam score -388.000000
for student #7 (student ID 808080808)
should be between 0.000000 and 800.000000.

E. CALCULATION SUBSECTION

NOTE TO SELF: Need to explain how to find lowest and highest values.

In the calculation subsection, the program MUST calculate the following values, in the following order:

- for each student:
  1. their overall percentage score for the semester;
  2. their overall letter grade for the semester;

- for each assignment:
  1. that assignment’s mean score among all the students (see below);
  2. that assignment’s lowest score among all the students (see below);
  3. that assignment’s highest score among all the students (see below);

- for the entire class:
  1. the mean of the overall percentage scores of all students in the course;
  2. the lowest overall percentage score of all students in the course;
  3. the highest overall percentage score of all students in the course;

You may assume that no two students will have exactly the same overall percentage score.

In any for loop in the calculation subsection, you MUST calculate EXACTLY ONE kind of result; that is, you are ABSOLUTELY FORBIDDEN to calculate multiple kinds of results in a single for loop.

For example, the for loop that calculates the number of students who receive an A CANNOT also calculate the number of students who receive a B.

However, within a particular for loop, you may choose to calculate temporary scalar variables representing various subexpressions.
HOW TO CALCULATE THE MEAN OF THE VALUES IN AN ARRAY
See the lecture slide packet “User Defined Functions Lesson 1,” slide #2.

IMPORTANT IMPORTANT IMPORTANT IMPORTANT IMPORTANT IMPORTANT!!!
For this project, you are ABSOLUTELY FORBIDDEN to have any user-defined functions. (We’ll use those in PP#6.)

HOW TO FIND THE LOWEST VALUE IN AN ARRAY

1. Start by assuming that the lowest value in the array is the value of the first element of the array.
2. Loop over all of the elements in the array:
   (a) compare the value of each element in the array to what you currently have for the lowest value found so far;
   (b) if the value of the current element of the array is less than what you currently have for the lowest value found so far, then the new lowest value is the value of the current element of the array.

HOW TO FIND THE HIGHEST VALUE IN AN ARRAY
Do exactly what you did for the lowest value, except using greater than instead of less than.

F. OUTPUT SUBSECTION
In the program’s output subsection, you MUST output the following:

- the name of the course, the semester, and the year, in that order (for example, “CS1313 Spring 2017”);
- the number of students;
- each of the items calculated, in the same order as in the calculation subsection.

Each output MUST be accompanied by helpful explanatory text; for example, the OUTPUT might look like this:

Quiz #1 mean score: 27.250000

You may output these quantities in any format that you like, as long as the meaning of the quantities is CLEARLY EXPLAINED in the outputs. You are welcome to use format descriptors on your placeholders (for example, "%10.5f"), but you AREN'T required to use them.

G. DEALLOCATING ARRAYS
At the end of the program, after the output subsection, you MUST deallocate each of the arrays that were allocated in the input subsection, using a free statement for each, and then nullifying the pointer, like so:

```c
free(quiz1_score);
quiz1_score = (float*)NULL;
```

The deallocations MUST occur in the OPPOSITE ORDER from the allocations; that is, whichever array was allocated first MUST be deallocated last, and so on.

For details, see the lecture slide packet “Array Lesson 2,” slides 26-33.
III. INPUT DATA FILES

A. HOW TO FIND AND EXAMINE THE INPUT DATA FILES

The input files for your runs can be found on ssh.ou.edu in the directory
~neem1883/CS1313pp5. You can find the names of all of the data files using the
ls command:

```bash
ls ~neem1883/CS1313pp5
```

The directory contains several data files; some are actual data and some are idiotproofing test files.
You **MUST** perform the runs in alphabetical order.

Actual (non-idiotproofing) test files have file names beginning with the prefix

```bash
actual_
```

Idiotproofing test files have file names beginning with the prefix

```bash
idiot_
```

You **SHOULD CLOSELY EXAMINE** (but not change) the contents of each of the data files
using `nano`:

```bash
nano ~neem1883/CS1313pp5/actual_cs1313_2097spring.txt
```

B. HOW THE DATA WILL BE INPUT

For this programming project, **YOU WON’T PROMPT THE USER FOR THE INPUTS**, be-
cause there won’t be a user as such. Instead, the inputs will come from a file. To get the inputs
from the file, you'll use a command like this at the Unix prompt:

```bash
grading < ~neem1883/CS1313pp5/actual_cs1313_2097spring.txt
```

This use of a file is referred to as **redirecting input**. The less than symbol `<` indicates that the
input will come from the file named `actual_cs1313_2097spring.txt`. In other words,
as far as the program is concerned, the file will appear to be a user typing at the keyboard, and the
program will accept input from the file exactly as if that input were being typed at the keyboard by
a real live user. Thus, you **MUST** write your `scanf` statements exactly as if a user were going
to be typing the data at the keyboard, but without the user needing to be prompted.

However, because there isn’t actually a real live user, it isn’t necessary to greet the user nor to
prompt for inputs; the data file won’t understand the prompts anyway, so to speak.

Your run commands **MUST** look like this example:

```bash
grading < ~neem1883/CS1313pp5/idiot_cs1313_2098spring_01.txt
```

This means, “run the executable named `grading`, redirecting input from the file named
`idiot_cs1313_2098spring_01.txt` that’s in the directory named
`~neem1883/CS1313pp5`.”
IV. RUNS
Run this program several times, using the several different input files that are available (see below). The runs **MUST** be in alphabetical order according to the input file names.
The order of the runs in your script file **MUST** be:

- all actual_ files, in alphabetical order, followed by
- all idiot_ files, in alphabetical order.

V. ADDITIONAL GRADING CRITERIA
All grading criteria for Programming Projects #2, #3 and #4 apply.

1. Declaration order:
   In the declaration section, the order of declarations **MUST** be:
   (a) named constants: float scalars followed by int scalars;
   (b) variables, in the following order:
      i. arrays: float arrays followed by int arrays;
      ii. scalars: float scalars followed by int scalars.

2. Block closes of *for* statements:
   **ALL** block closes associated with *for* statements **MUST** be followed, on the same line, by a space, a comment open, a space, the keyword *for*, a space, the counter variable, a space, and a comment close. For example:

   ```c
   for (student = first_student; student < number_of_students; student++) {
      scanf("%d %f %f %f %f %f %f %f %f",  
          &student_ID[student],  
          &quiz1_score[student], &quiz2_score[student],  
          &quiz3_score[student], &quiz4_score[student],  
          &pp1_score[student], &pp2_score[student], &pp3_score[student],  
          &lab_score[student], &final_exam_score[student]);
   } /* for student */
   ```

3. Indenting *for* statements and their associated block closes:
   For a given *for* loop, the *for* statement and its associated block close **MUST** be indented identically, and this indentation amount **MUST** be appropriate with respect to their position within the program.

4. Indenting inside *for* loops:
   For a given *for* loop, **ALL** statements **INSIDE** the *for* loop **MUST** be indented FOUR SPACES farther than the *for* statement and its associated block close. For example:

   ```c
   overall_score_sum = initial_sum;
   for (student = first_student; student < number_of_students; student++) {
      overall_score_sum += overall_score[student];
   } /* for student */
   overall_score_mean = overall_score_sum / number_of_students;
   ```
5. **Commenting for loops:**
Each *for* loop *MUST* be preceded by a comment that describes what the loop as a whole does. For example:

```c
/*
 * Calculate the sum of all of the overall percentage scores.
 */
for (student = first_student; student < number_of_students; student++) {
    overall_score_sum += overall_score[student];
} /* for student */
```

6. **Commenting inside for loops:**
A statement inside a *for* loop *MUST* be preceded by a comment that describes what the statement does. The comment *MUST* be properly indented, so that the asterisk of the comment lines up with the statement. For example:

```c
for (student = first_student; student < number_of_students; student++) {
    /*
     * Increase the overall score sum by the value of the
     * current student's overall score.
     */
    overall_score_sum += overall_score[student];
} /* for student */
```

VIII. **DEBUGGING VIA printf STATEMENTS**
The best mechanism for debugging this program is to put in lots of printf statements that show where in the program the run currently is. For example:

```c
for (student = first_student; student < number_of_students; student++) {
    printf("About to input data for student #%d.\n", student);
    scanf("%d %f %f %f %f %f %f %f %f %f",
            &student_ID[student],
            &quiz1_score[student], &quiz2_score[student],
            &quiz3_score[student], &quiz4_score[student],
            &pp1_score[student], &pp2_score[student], &pp3_score[student],
            &lab_score[student], &final_exam_score[student]);
    printf("Done inputting data for student #%d:\n", student);
    printf("student_ID[%d]: %d\n", student, student_ID[student]);
    printf("quiz1_score[%d]: %f\n", student, quiz1_score[student]);
    printf("quiz2_score[%d]: %f\n", student, quiz2_score[student]);
    printf("quiz3_score[%d]: %f\n", student, quiz3_score[student]);
    printf("quiz4_score[%d]: %f\n", student, quiz4_score[student]);
    printf("pp1_score[%d]: %f\n", student, pp1_score[student]);
    printf("pp2_score[%d]: %f\n", student, pp2_score[student]);
    printf("pp3_score[%d]: %f\n", student, pp3_score[student]);
    printf("lab_score[%d]: %f\n", student, lab_score[student]);
    printf("final_exam_score[%d]: %f\n", student, final_exam_score[student]);
} /* for student */
```

**IMPORTANT IMPORTANT IMPORTANT IMPORTANT IMPORTANT IMPORTANT!!!**
Once you’ve completed debugging, you *MUST* delete ALL debugging printf statements. **EXTRANEOUS OUTPUTS WILL BE SEVERELY PENALIZED.**
IX. EXTRA CREDIT

You can receive an extra credit bonus of as much as 5\% of the total value of PP#5 by doing the following:

1. Attend at least one CS1313 help session for at least 30 minutes, through Wed Apr 12.
2. During the help session that you attend, work on CS1313 assignments (ideally PP#5, but any CS1313 assignment is acceptable). **YOU CANNOT GET EXTRA CREDIT IF YOU DON’T WORK ON CS1313 ASSIGNMENTS DURING THE HELP SESSION.**
3. Before you leave the help session, fill out BOTH halves of the form on the last page of this project specification and have the help session leader (instructor or TA) sign BOTH halves. **THE FORM CANNOT BE SIGNED UNTIL IT IS COMPLETELY FILLED OUT.**
4. Attach the bottom half of the form to your PP#5 script printout, **AFTER** the script itself, and keep the top half for your own records.

**BONUS VALUE NOTICE:** Up through Wed Apr 5, the extra credit bonus will be worth 5\% of the total value of PP#5, but from Mon Apr 10 through Wed Apr 12, the extra credit bonus will be worth only 2.5\% of the total value of PP#5. That is, **YOU’LL GET TWICE AS MUCH EXTRA CREDIT DURING THE FIRST WEEK AS DURING THE SECOND WEEK.**
CS1313 PROGRAMMING PROJECT #5 BONUS REQUEST FORM

Name ________________________________ Lab __________
Help Session Date ________________
Help Session Time (Arrive) ________________ Help Session Time (Depart) ________________

Instructor Signature ________________________________

Keep this copy for your records.

Submit this copy.
In your submission, attach this copy AFTER your script file printout.
If you put this in the wrong place in your submission, then you WON’T get the extra credit.