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 expected 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.

1. PROJECT DESCRIPTION

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

- Programming Projects
- 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:

- 2 programming projects, together worth 65% of the overall percentage score, with maximum possible scores of 100 points (PP#1) and 200 points (PP#2);
- 1 final exam, worth 35% of the overall percentage score, with a maximum possible score of 500 points.

NO ASSIGNMENT HAS ANY BONUS, so the maximum score for a particular assignment is genuinely the maximum score that anyone can score for that assignment.

The expression for calculating grades in CS1313 is a subset of the one appearing on the Spring 2015 CS1313 syllabus, at the top of page 3. But, for PP#5 there will be no quiz term, no lab term, no in-class exam term, no short programming assignment (CodeLab) term and no iClicker term in the expression.

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.
II. PROGRAM DESCRIPTION

Write a program to calculate grading information from assignment scores. The execution section (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 program body.

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* project1_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 section 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 programming project scores, in order (PP#1, PP#2);
      ii. the final exam score.

Several such data files will be provided, each representing an individual run. Students will be identified by student ID, not by name. 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).

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 student ID numbers and 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. Therefore, **ALL** arrays **MUST** be declared as **POINTERS**. **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, your **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, 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. You should idiotproof each value immediately after it is input. **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 500.000000.

E. CALCULATION SUBSECTION
In the calculation subsection, the program **MUST** calculate:

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

- for each assignment:
  1. the mean and variance (see below);

- for the entire class:
  1. the mean and variance of the overall percentage scores of all students in the course (see below);
  2. the number of students who got each of the 5 letter grades.

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. (for example, a variable representing the student’s total score on all programming projects, in the **for loop** that calculates the overall percentage score).
F. MEAN AND VARIANCE

Given a list of $n$ real numbers $x_1, x_2, \ldots, x_n$

the mean of the values in the list is an average, which is a value that is typical of the values in the list. The mean, here denoted $\bar{x}$ (pronounced “x-bar”), is calculated as the sum of the all the values in the list, divided by the number of values in the list:

$$\bar{x} = \frac{x_1 + x_2 + \cdots + x_n}{n}$$

The bias-corrected sample variance of the list of real numbers, known as the variance for short and denoted $v$, is obtained by:

$$v = \frac{(x_1 - \bar{x})^2 + (x_2 - \bar{x})^2 + \cdots + (x_n - \bar{x})^2}{n - 1}$$

Note that, for a list of length 1, the variance is 0 by definition.

In your program, you are absolutely forbidden to use variable names such as $x_{\text{bar}}$ or $v$, because they violate the favorite professor rule.

Note that statistics such as mean and variance cannot be guaranteed to have integer values, and in real life very rarely are integers.
G. OUTPUT SUBSECTION
In the program’s output subsection, you MUST first output the year, the semester and the number of students. You are ABSOLUTELY FORBIDDEN to output the semester code; instead, you output the word “Spring” or the word “Fall” as appropriate.
Immediately following that, you MUST output the information in the list for the calculation section (above), in the order listed.
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 print format descriptors in your placeholders (for example, "%%%5.2f"), but you AREN’T required to use them.

H. 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:

```
free(project1_score);
project1_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:

% 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

\texttt{actual}\_

Idiotproofing test files have file names beginning with the prefix

\texttt{idiot}\_

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

% 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, because 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:

% grades < ~neem1883/CS1313pp5/actual_cs1313_2097spring.txt

This use of a file is referred to as \textit{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 idiotproof run commands MUST look like this, but for each of the idiotproof filenames:

% grades < ~neem1883/CS1313pp5/idiot_cs1313_2098spring_01.txt

This means, “run the executable named grades, redirecting input from the file named idiot_cs1313_2098spring_01.txt that’s in the directory named

~neem1883/CS1313pp5.”

Likewise, your actual run commands MUST look like the example above, but for each of the actual\_ filenames.
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.

The following additional grading criteria also apply to PP#5 (and will likewise apply to PP#6).

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", &student_ID[student], &project1_score[student], &project2_score[student], &final_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:

   ```
   /*
    * 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:

   ```
   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:

   ```
   for (student = first_student; student < number_of_students; student++)
   
   printf("About to input data for student #%d.\n", student);
   scanf("%d %f %f %f",
       &student_ID[student],
       &project1_score[student], &project2_score[student],
       &final_score[student]);

   printf("Done inputting data for student #%d: \n", student);
   printf(" student_ID[%d]: %d\n", student, student_ID[student]);
   printf(" project1_score[%d]: %f\n", student, project1_score[student]);
   printf(" project2_score[%d]: %f\n", student, project2_score[student]);
   printf(" final_score[%d]: %f\n", student, final_score[student]);
   ...
   
   /* for student */
   ```

Once you’ve completed debugging, you MUST delete ALL debugging printf statements.

**EXTRANEOUS OUTPUTS WILL BE SEVERELY PUNISHED.**
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 8.
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 1, the extra credit bonus will be worth 5% of the total value of PP#5, but from Mon Apr 6 through Wed Apr 8, 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.**
This page intentionally left blank.
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.

CS1313 PROGRAMMING PROJECT #5 BONUS REQUEST FORM

Name  ____________________________________________  Lab __________
Help Session Date ________________
Help Session Time (Arrive) _____________  Help Session Time (Depart) _______________

Instructor Signature  __________________________________________

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.