1 Instructions

You may work in **pairs** (that is, as a **group of two**) with a **partner** on this lab project if you **wish** or you may work **alone**. If you work with a partner, only submit **one** lab project with **both** of your **names** in the **source code file(s)** to Blackboard for grading; you will each earn the **same** number of points. **What** to hand in, and **by when**, is discussed in **Section 5**; read it.

2 Lab Objectives

After completing this assignment the student should be able to:

- Complete all of the objectives of the previous lab projects.
- Write **#include preprocessor directives** to include C++ Standard Library header files **cmath**, **iomanip**, and **iostream**.
- Define and use a **named constant**.
- Use **cout** to display string literals and **int** and **double** values to the output window.
- Use **endl** to display output on multiple lines of the output window.
- Define **int** and **double** variables and constants.
- Assign values to variables using the **assignment operator =**.
- Store integer and real numbers read from the keyboard via **cin** into **int** and **double** variables.
- Write **arithmetic expressions** using the arithmetic operators: + - * / and %.
- Understand the **operator precedence rules** and the difference between **integer division** and **floating point division**.
- Use the **static_cast<>** operator.
- Call C++ standard library **math functions**.
- Use **fixed** and **setprecision** stream manipulators to output real numbers with a specific number of digits after the decimal point.
- Use **indentation** and **spacing** to properly **format code**.

3 Prelab Exercises

- Download *Lab03.cpp* from the course website. This source code file is a "template" for the program you will write. Then, create a Code::Blocks project (or a project using your preferred IDE) and add *Lab03.cpp* to the project. You will be completing the code in *Lab03.cpp* by writing statements to implement the lab project requirements described in Section 4.

4 Lab Exercise

Suppose a small course contains three students: Homer, Lisa, and Ralph. Each student takes an exam and the exam score is an integer in the range [0, 100]. The exam average can be computed using the formula,

$$exam_{avg} = \frac{(exam_1 + exam_2 + exam_3)}{\text{NUM\_STUDENTS}}$$

where $exam_1$ is Homer's exam score, $exam_2$ is Lisa's exam score, $exam_3$ is Ralph's exam score, and *NUM_STUDENTS* is a constant which is equivalent to 3. In statistics, the variance of this group of three scores is defined to be,

$$exam_{variance} = \frac{(exam_1 - exam_{avg})^2 + (exam_2 - exam_{avg})^2 + (exam_3 - exam_{avg})^2}{\text{NUM\_STUDENTS} - 1}$$

And the standard deviation[1] is,

$$exam_{\text{stddev}} = \sqrt{(exam_{variance})}$$

For this lab project, write a program which asks the user to enter the exam scores (as **int**s) for Homer, Lisa, and Ralph. The program shall then calculate and display the average and standard deviation of the exam scores. The average and standard deviation will be real numbers (i.e., **double**). See the output from my program below, and make yours work just like this. User input is shown in bold.

```
Enter exam score for Homer: 62
Enter exam score for Lisa:  100
Enter exam score for Ralph: 28

The average exam score is:      63.33%
The exam standard deviation is: 36.02
```

1 In statistics, there is a difference between the **population standard deviation** and the **sample standard deviation**. By dividing by NUM_STUDENTS - 1 (which is 2), we are calculating the **sample** standard deviation. To calculate the population standard deviation, one would divide by NUM_STUDENTS. That, I understand how to do. Why, I don't, so don't ask.

4.1 Miscellaneous Hints

1. To display real numbers (i.e., `doubles`) with 2 digits after the decimal point, you must,

   a. Write a preprocessor directive `#include <iomanip>` at the top of the source code file with the other `#include` directives.
   b. Write `cout << fixed << setprecision(2);` above the line of code where you actually display the real numbers.

2. Define three `int` variables, *exam1*, *exam2*, and exam3, for Homer's, Lisa's, and Ralph's exam scores.

3. Define three `double` variables, *exam_avg*, *exam_variance*, and *exam_stddev*.

4. Define an `int` constant named `NUM_STUDENTS` by writing `const int NUM_STUDENTS = 3;` and use this constant when calculating the exam average and the exam variance. Constants are normally defined outside of and above the *main*() function (which makes the constant global; the scope of a global constant is from the point where it is defined to the end of the source code file).

5. When calculating the exam average, you will have to use the `static_cast<double>` type cast operator, i.e.,

   ```
   exam_avg = (exam1 + exam2 + exam3) / static_cast<double>(NUM_STUDENTS);
   ```

   Why? Remember, in C++ if you divide two `int`s you will get an `int` as the result, e.g.,

   ```
   int a = 10, b = 5
   double c = (a + b) / 2;  // c is assigned 7 because both operands are ints
   cout << c << endl;       // Displays 7
   c = (a + b) / 2.0;       // c is assigned 7.5 because the right-hand side operand is a double
   cout << c << endl;       // Displays 7.5
   ```

6. When calculating the exam variance, use the *pow*() function from the C++ Standard Library, e.g.,

   ```
   exam_variance = (pow(exam1 - exam_avg, 2) + pow(exam2 - exam_avg, 2) + ...
   ```

   The description of the *pow*() function is in a header file named *cmath* which must be included with a `#include` directive.

7. The C++ Standard Library function for calculating the square root of a number is *sqrt*() and it is also described in the *cmath* header file. To call *sqrt*() write,

   ```
   exam_stddev = sqrt(exam_variance);
   ```

8. **Important note for Microsoft Visual Studio users.** Because of the way Microsoft has written their version of the C++ Standard Library, you will need to type cast the arguments to *pow*() and *sqrt*() to doubles in the function call, e.g.,

   ```
   exam_variance = (pow(static_cast<double>(exam1 - exam_avg), 2) + ...
   exam_stddev = sqrt(static_cast<double>(exam_variance));
   ```

4.2 Additional Programming Requirements

1. Put a **comment header block** at the top of your source code file that looks something like the following. Change the source code file name, author information, lab number, lab date/time, and lab TA appropriately

   ```
   //*****************************************************************************************
   // FILE:        Lab03.cpp
   //
   // DESCRIPTION: Reads three exam scores for three students, calculates and displays the average
   //              exam score and the standard deviation of the exam scores.
   //
   // AUTHORS:     your-name>           (your-email-address)
   //              your-partner's-name> (your-partners-email-address)
   //
   // COURSE:      CSE100 Principles of Programming with C++, Fall 2015
   //
   // LAB INFO:    Lab 3   Date/Time: your-lab-date-and-time   Lab TA: your-lab-ta
   //*****************************************************************************************
   ```

2. Carefully **format** your code and follow the **indentation** of the text as shown in the example programs of the textbook.

5 What to Submit for Grading and by When

Upload the *Lab03.cpp* C++ source code file to Blackboard using the lab submission link by the deadline. If your program does not compile or run correctly, upload what you have completed for grading anyway (you will generally receive some partial credit for effort). The deadline for the complete lab project is **4:00am Sat 26 Sep**. Consult the online syllabus for the late and academic integrity policies.

6 Grading Rubric

1. Lab Exercise Program (0 to 5 pts)

a. If the student did not submit a source code file or if he/she simply uploaded the "template file" given to them, assign **+0 pt**.
b. If the submitted program does, or does not, compile and the student completed less than 50% of the required code correctly, assign **+2 pts**.
c. If the submitted program does not compile and the student completed more than 50% of the required code correctly, assign **+3 pts.**
d. If the submitted program compiles and the student completed more than 50% of the required code correctly, assign **+4 pts**.
e. If the submitted program compiles and is implemented perfectly, or close to perfect with only one or two minor mistakes, assign **+5 pts**.

2. Deadline was 4:00am Sat 26 Sep

1. Assign 20% bonus calculated on the earned pts for a submission prior to 4:00am Thu 24 Sep.
2. Assign 10% bonus calculated on the earned pts for a submission between 4:00am Thu 24 Sep and 4:00am Fri 25 Sep.
3. Deduct 0.5 pt for a submission between 4:00am Sat 26 Sep and 4:00am Sun 27 Sep.
4. Deduct 1 pt for a submission after 4:00am Sun 27 Sep.