Project 1 specifications

September 2024

1 Overview and purpose

This project involves implementing a program that calculates your course grade. It begins by prompting you to input your grades for each project and then computes your final grade accordingly. Each project's maximum grade must be entered along with the grade achieved. Both values undergo validation: the maximum grade must be a positive integer, and the achieved grade must be between 0 and the maximum grade.

Similar calculations are commonplace in engineering applications:

1. Online banking: Withdrawals and transfers require positive amounts and must be to dollars and cents. They cannot exceed the current balances, possibly including an overdraft limit.
2. Mars Rover Operations: Rovers like NASA's Mars rovers (e.g., Curiosity, Perseverance) receive commands from Earth for movements, experiments, and data collection. Input validation ensures that commands for movement distances, instrument operations, and data transmission are accurate and within operational limits to avoid risks in the Martian environment.
3. Temperature Monitoring: In industrial settings or scientific experiments, sensors are used to measure temperatures. Software systems may need to verify that temperature inputs fall within specified operational ranges, ensuring accurate responses and calculations based on these measurements.
4. Autonomous Vehicles: Self-driving cars and autonomous drones rely heavily on accurate sensor data and input validation. Inputs from sensors (e.g., cameras, LIDAR, radar) are validated to ensure they are reliable and within expected ranges before decisions are made (e.g., steering, speed adjustments, obstacle avoidance).

Although these systems are significantly more complex, they all start with input validation followed by appropriate calculations—exactly what you'll practice in this initial project.

2 Project description

The course components consist of

1. A mid-term examination.
2. A final examination.
3. Five projects.

The maximum grade in each component will be given as an integer greater than zero. The grade achieved may be a real number but must be between zero and the maximum grade. For example, you may achieve a grade of 37.25 out of 40 on the mid-term examination and a grade of 75.4 out of 90 on the final examination.

2.1 Entering the grades

You will first ask the user for what the maximum grade in each component is and then what the achieved grade in that component was, in the order:

1. The final examination.
2. The mid-term examination.
3. The projects in the order 1, 2, 3, 4 and 5.

The maximum grade in each component must be a positive integer[1] The achieved grade in each component must be a value between 0 and the maximum grade inclusively[2] In each case, if the value is not as specified, you will simply ask for that value again, and again, and again, until the user enters an appropriate grade.

Your program will print a single integer as output: that integer being the calculated grade.

2.2 Supremacy of the final examination

Second, you will calculate each of your grades out of 100, so therefore, you will calculate

$$F_{100} = 100\frac{F}{F_{\max}}, M_{100} = 100\frac{M}{M_{\max}}, P_{1,100} = 100\frac{P_1}{P_{1,\max}}, \ldots, P_{5,100} = 100\frac{P_5}{P_{5,\max}}$$

Next, if any of these grades are less than F_{100}, you will replace that grade with the value of F_{100}.

2.3 Examination weighted average

Let $M_{\max}$ and $F_{\max}$ be the maximum grades in both the mid-term and final examinations and let M_{100} and F_{100} be your achieved grades in those two examinations. First, you will create a weighted average[3] of your

[1] 0 is neither positive nor negative, so the "positive integers" refer to $1, 2, 3, 4, \ldots$. If you want to include 0, you should refer to the "non-negative integers."

[2] To say that a value is "between a and b inclusively" says that the values a and b are allowed. If you were told a value was "between 2 and 10 exclusively," then the value could be 2.01 or 9.999, but the value cannot be exactly equal to either 2 or 10.

[3] The "average" of four numbers is $\frac{x_1+x_2+x_3+x_4}{4} = 0.25x_1 + 0.25x_2 + 0.25x_3 + 0.25x_4$. A "weighted average" is any such linear combination where the sum of the coefficients equals one. For example, $0.1x_1 + 0.2x_2 + 0.3x_3 + 0.4x_4$ and $0.7x_1 + 0.2x_2 + 0.05x_3 + 0.05x_4$ are both weighted averages of these four numbers. Your grade in any course will always be a weighted average of the various components: it would be peculiar if your final grade was just the average of your grade on each component: $\frac{\frac{F}{F_{\max}}+\frac{M}{M_{\max}}+\frac{P_1}{P_{1,\max}}+\frac{P_2}{P_{2,\max}}+\frac{P_3}{P_{3,\max}}+\frac{P_4}{P_{4,\max}}+\frac{P_5}{P_{5,\max}}}{7}$.

mid-term and final grades: $E_{100} = \frac{3}{4}F_{100} + \frac{1}{4}M_{100}$. Because both grades F_{100} and M_{100} out of 100, then so too must E_{100}.

2.4 Calculating the project grade

The project grade is the average of the four projects, so

$$\begin{aligned} P_{100} &= \frac{P_{1,100} + P_{1,100} + P_{1,100} + P_{1,100} + P_{1,100}}{5} \\ &= 0.2P_{1,100} + 0.2P_{2,100} + 0.2P_{3,100} + 0.2P_{4,100} + 0.2P_{5,100}. \end{aligned}$$

Because each project grade $P_{k,100}$ is out of 100, then so too must the average of these five projects.

2.5 Final grade calculation

Finally, your final grade calculation will depend on your examination grade E_{100}:

1. If $E_{100} \leq 40$, your final grade is E_{100}.
2. If $E_{100} \geq 60$, your final grade will be $\frac{2}{3}E_{100} + \frac{1}{3}P_{100}$. You may remember that $E_{100} = \frac{1}{4}M_{100} + \frac{3}{4}F_{100}$, so this calculation is equal to $\frac{2}{3}\left(\frac{3}{4}F_{100} + \frac{1}{4}M_{100}\right) + \frac{1}{3}P_{100} = \frac{1}{2}F_{100} + \frac{1}{6}M_{100} + \frac{1}{3}P_{100}$. You may remember that $\frac{1}{2} + \frac{1}{6} + \frac{1}{3} = 1$.
3. Otherwise, you will see that $40 < E_{100} < 60$. In this case, the weight of the projects increases linearly from 0 if your examination grade was 40 and $\frac{1}{3}$ if your examination grade was 60. Here is how we get there:
   - You will see that the formula $\frac{E_{100}-40}{20}$ has a value of 0 if $E_{100} = 40$ and a value of 1 if $E_{100} = 60$. This is a linear polynomial that interpolates $(40, 0)$ and $(60, 1)$.
   - Thus, $\frac{1}{3}\frac{E_{100}-40}{20}$ has a value of 0 if $E_{100} = 40$ and a value of $\frac{1}{3}$ if $E_{100} = 60$. This is a linear polynomial that interpolates $(40, 0)$ and $(60\frac{1}{3})$.
   - Thus, $1 - \frac{1}{3}\frac{E_{100}-40}{20}$ has a value of 1 if $E_{100} = 40$ and a value of $\frac{2}{3}$ if $E_{100} = 60$. This is a linear polynomial that interpolates $(40, 1)$ and $(60, \frac{2}{3})$.
   - Thus, the final grade will be $P_{100}\frac{1}{3}\frac{E_{100}-40}{20} + E_{100}\left(1 - \frac{1}{3}\frac{E_{100}-40}{20}\right)$. If you expand this out, you will get the formula on the course syllabus: $-\frac{1}{60}E^2 + \frac{5}{3}E + \frac{1}{60}P_{100}E_{100} - \frac{2}{3}P$.

In all cases, we will round the final grade to the closest integer, and if the final grade is exactly 0.5, we will round up, so 89.5 will be rounded up to 90. See the "tricks and tips" topic on rounding below. The final grade will be printed as the string `"Final grade: "` followed by the rounded final grade as an integer.

Notice: If your examination weighted average is equal to 40, both the first and third formulas give exactly the same result. Similarly, if your examination weighted same result. Similarly, if your final

3 Tricks and tips

In this section, we will reinforce some topics that are taught in class, but are especially necessary for this project.

3.1 The C math library

We will use the C mathematics library in the project. Just like you include the input/output stream library using

```
#include <iostream>
```

to include the C mathematics library, you must use

```
#include <iostream>
#include <cmath>
```

To test if a floating-point number is an integer, you will make the following comparison: suppose that `F_max` is of type `double` and is storing the maximum grade on the final examination. The following comparison will be `true` if that floating-point number is an integer, and false otherwise:

```
if ( F_max == std::round( F_max ) ) {
    // Do something if 'F_max' is an integer.
} else {
    // Do something else if 'F_max' is not
    // an integer (e.g., 54.5).
}
```

You are expected to round the final calculated grade to the nearest integer, and if a grade is exactly half way between two integers (e.g., 83.5 or 84.5), you must round up. Fortunately, this is the behavior of the `std::round(...)` function as described in the documentation. But this is not the complete story.

Floating-point numbers approximate real numbers in a computer, but they only store a finite and fixed number of digits beyond the decimal point. Thus, π can never be stored exactly using a local variable of type `double`. One consequence is that one programmer may perform one sequence of calculations and get the value 73.5000000000003 while another may perform the same calculations but in different orders and get 73.4999999999996. For most engineering applications, this difference is insignificant: the difference is approximately one part in one quadrillion. However, in calculating your final grade, we will assume that both of these actually should have resulted in the value of 73.5, and this final grade should be rounded up to 74. However, the second number will round down to 73 while the first will round up to 74. Thus, to avoid such rounding errors, you will perform the following computation:

```
final_grade = std::round( final_grade + 1e-12 );
```

This adds the value 10^{-12} before rounding, so that the two numbers above become 73.50000000000102 and 73.50000000000095, respectively, before they are rounded.

Important: There may be one student in one million whose actual grade does indeed result in 84.49999999999973, so adding 10^{-12} might unfairly give this student a final grade of 85 instead of 84, however, the vast majority of students where the calculation of such a grade results in a number so close to 84.5 actually do students

3.2 Printing of floating-point numbers

When `std::cout` prints a local variable of type `double`, even though the local variable stores approximately sixteen decimal digits of precision, the value by default is only printed to six decimal digits. Additionally, trailing zeros to the right of the decimal point are not printed. Also, if nothing is printed to the right of the decimal point, then the decimal point is not printed, either. The consequence of this is that if a local variable is storing a value very close to an integer, it is printed as an integer. Thus, if the value of `n` is any value between 0 and 100 inclusively, then `std::cout << n << std::endl;` must print what appears to be an integer to the screen.

3.3 Scope of local variables

Local variables are declared inside a program, but can only be accessed between where that variable is declared and the end of the block in which it is declared. Thus, the following code will not work:

```
// We cannot access 'P1max' in the condition
// of the while loop, as it has not yet
// been declared.
while ( P1max-is-not-a-positive-integer ) {
    double P1max{};

    std::cout << "Enter the maximum grade in project 1: ";
    std::cin  >> P1max;
}

// Error: we cannot access 'P1max', either,
// as it was declared inside the loop body.
std::cout << P1max << std::endl;
```

The local variable `P1max` must be declared before the loop body:

```
double P1max{};

while ( P1max-is-not-a-positive-integer ) {
    std::cout << "Enter the maximum grade in project 1: ";
    std::cin  >> P1max;
}
```

3.4 A do-while loop

You do not have to learn this to get 100% in the project, so you are welcome to skip it. It is simply time that prevents us from teaching the do-while loop. You will not be tested on the do-while loop on either examination.

In class, you are taught about the while loop:

```
while ( some-condition ) {
    // Do something...
}
```

This always tests the condition before the body of the loop is executed, and if the condition is ever `false`, the loop stops executing. When entering a grade, it is essential that the initial value of the grade causes the condition to fail:

```
double P1{-1.0};

while ( P1-is-between-0-and-P1max-inclusive ) {
    std::cout << "Enter your grade in project 1: ";
    std::cin  >> P1;
}
```

This works, but it may make the reader wonder "what is so significant about -1.0?" A do-while loop is like a while loop, but the body of the loop is always executed at least once before the test is made, and the structure reflects this:

```
double P1{-1.0};

do {
    std::cout << "Enter your grade in project 1: ";
    std::cin  >> P1;
} while ( P1-is-between-0-and-P1max-inclusive );
```

Please note: You will notice that there is a semicolon after the closing round parenthesis at the end of the do-while loop condition.

3.5 Integer versus floating-point division

When you do integer division, it discards the remainder, so in essence, it rounds towards zero. Floating-point division performs division as you learned in secondary school. The compiler, however, is very careful to do exactly what you tell it to do. For example, execute these statements:

```
double E_100{ 100.0 };
std::cout << (3/4*E_100) << std::endl;
std::cout << (1/3*5.4) << std::endl;
```

The output for both of these is `0`. This is different from the output of these statements:

```
double E_100{ 100.0 };
std::cout << (3.0/4.0*E_100) << std::endl;
std::cout << (0.75*E_100) << std::endl;
std::cout << (5.4/3) << std::endl;
```

Here, you get the expected answers: `75` and `1.8`. The reason is that the compiler sees, for example, `1/3*5.4` and goes through the code left to right: it sees the integer `1`, a division symbol, and then the integer `3`. It then deduces you want to perform an integer division, and the result of calculating `1/3` is `0`. Only then does the compiler see the multiplication symbol and the floating-point `5.4`, so it converts the result, the integer `0`, to the floating-point `0.0`. If you want the correct result printed, you should use:

```
double E_100{ 100.0 };
std::cout << (0.75*E_100) << std::endl;
```

```
std::cout << ((3.0/4.0)*E_100) << std::endl;
std::cout << (1.0/3.0*5.4) << std::endl;
// This is clearer:
std::cout << ((1.0/3.0)*5.4) << std::endl;
// This is better:
std::cout << (5.4/3.0) << std::endl;
```

3.6 Variable names

From the above discussion, it seems there should be three variables for each component, so for example

```
double P3max{};
double P3{};
// Input these values...
double P3_100{ P3/P3max };
```

The different names in this document make it clear that `P3` and `P3_100` are storing different values, each satisfying these two conditions, respectively:

```
(0.0 <= P3) && (P3 <= P3max)    (0.0 <= P3_100) && (P3_100 <= 100.0)
```

However, you could, instead, once all values are entered, perform a block transformation:

```
F /= F_max;
M /= M_max;
P1 /= P1_max;
// ...
// All grades are now out of 100
```

This is a personal choice, and you're welcome to take either approach. In this specification, we must differentiate them to make it clear what is being referred to: P_3, $P_{3,\max}$ and $P_{3,100}$.

If you're worried about memory, don't be: any decent compiler would not allocate any additional memory for `P3_100` as the compiler would determine that `P3` is never used again. Clarity is more important than optimization: the compiler is almost certainly a better optimizer, anyways.