[HW] Grade Calculator

This is an **individual assignment**.

- Do not share your code with other students.
- Do not show your code to other students.
- Do not look at the code of other students.
- Do not ask other students how they solved a problem.

HWs are **individual** practice assignments (LWs, on the other hand, are collaborative practice assignments). If you have questions about this assignment, talk to a Peer Teacher, a TA, or an instructor.

- Go to Gradescope to submit your code.
- Ask questions on Piazza
    - Look here to see if someone has already asked your question.
    - It is very rare that a student's question is unique. Ask here and everyone can benefit from the answer.
    - If you know the answer, go ahead and answer it yourself!
    - If posting to the discussion would be an academic integrity violation since you would end up posting some of your homework code, use the Canvas Inbox to send a message to All instructors and All TAs instead.

If you know about another student who is sharing their code with other students (or in any other way is violating the Aggie Code of Honor), you should report them to the instructor or report them to the AHSO.

Use the syllabus, starter code, and test cases to build a grade calculator for CSCE 120/121/709.

Objectives

- Compile and run a C++ program.
- Basic logic.
- Consume data as you get it.
    - You don't need to save it in an array, vector, list, etc.

Requirements

1. The specification for the homework is the grading section of the syllabus and the test cases.
2. Submit only grade_calculator.cpp

3. Do not include any libraries except for the ones provided by the starter code.

Starter Code

The starter code includes a skeleton grade_calculator.cpp. **Everything except the actual grade calculation is already implemented for you.**

Download the starter code

Test cases (the *exact same* as on Gradescope) are provided along with the starter code. Do not submit the test cases to Gradescope (they will be ignored and you will waste bandwidth).

Grade Input Format

Grades are specified as: `<category> <score>`.

<category> values are: exam, final-exam, hw, lw, reading (zybook), engagement<score> values for exam, final-exam, hw, reading and engagement are specified as percentages (i.e. numbers from 0.0 to 100.0, but sometimes more when there is an opportunity for extra score). Converting a fraction to a decimal percentage is easy: 123/164 = (do the division) 0.75 = (multiply by 100) 75%.

Note: Per the syllabus, the reading and engagement categories have 15% added to their total, up to a total maximum of 100% in each category. Each of the hw, reading, engagement and lw categories are running averages, so track the number of grades in each category which have been read in, and average them together. For example, two engagement grades of 100 and 0 should be averaged to 50%, after which the 15% is added for a total of 65%.

<score> values for lw are boolean-valued (0 or 1, for incomplete or complete).

This is an example of grade input:

```
exam 96.6
hw 99.3
engagement 30.0
engagement 22.5
exam 94.2
lw 1
hw 78.8
lw 1
engagement 28.9
```

```
lw 1
hw 60.1
reading 89.5
hw 87.0
engagement 13.2
lw 1
hw 61.6
reading 90
```

Getting Started

Read the code in grade_calculator.cpp. Don't worry if you don't understand everything there, you'll eventually learn to do everything that is there during the semester.

Take note of the **TODO(student)** comments.

Compiling and Executing the Program (testing)

Compile

```
$ g++ -std=c++17 -Wall -Wextra -pedantic -Weffc++ grade_calculator.cpp
```

If you are curious about what the options we are specifying in this command line mean, you can look at **Appendix 1** below.

Exec

```
$ ./a.out
enter grades as <category> <score>
<category> := exam | final-exam | hw | lw | reading | engagement
<score> := numeric value
enter an empty line to end input
```

If the text in the figure with the ':=' symbol looks alien to you, just ignore it. If you are curious, you can look at **Appendix 2**.

Input Redirection

Instead of typing grades in line-by-line, we can use a capability of Unix-based operating systems to redirect the contents of a file to the standard input stream of the program. For programs that have a lot of input, as this one does, it is inconvenient to have to type the input again every time we run the program. By using input redirection, we can put the grades in a file, but still use console input for the program by default (i.e. grades *can* go in a file, but they don't have to). Putting a '<' symbol after the name of the program we are executing tells the shell to redirect the contents of the file on the right to the standard input stream of the program on the left.

```
$ ./a.out < test_complete_11.txt
enter grades as <category> <score>
  <category> := exam | final-exam | hw | lw | reading |
engagement
     <score> := numeric value
enter an empty line to end input
summary:
      exam average: 63.6
        hw average: 77.36
        lw average: 100
           reading: 100
        engagement: 38.65
    ---------------
    weighted total: 73.3165
final letter grade: C
```

Additional information about input redirection is in **Appendix 3**.

Notes on Gradescope Submissions

- One of the benefits of Gradescope is that the test cases usually provide some kind of feedback.

Have Fun!

Tips

Start by handling HW grades. Use the test_#.txt test files to test your code locally before submitting to Gradescope.

Don't add extra cin statements. The starter code already handles all input and validation.

Appendices

Students are not required to read the Appendices.

Appendix 1: Compiler Flags

g++, the compiler we use in this course, is very flexible.

It is designed to issue warnings regarding several aspects of the program being compiled, including warnings about program statements that are valid, but that programmers commonly misuse by mistake. The command line we provided tells the compiler to provide warnings of several categories:

- -Wall: this option turns on warnings for dozens of situations, including a few that programmers disagree about their rationale. The name is misleading, as it does not turn on all possible warnings.
- -Wextra: this adds more than a dozen warnings that we consider useful for novice programmers.
- -pedantic: it turns on all warnings related to the strict use of the C++ official standard.
- -Weffc++: it turns on warnings related to the guidelines from the book Effective C++

The option '-std=c++17' specifies the language standard (i.e., its definition version) that the compiler should apply. Other possible standards are 'c++14' (default on some systems), 'c++11' and even 'c++20' which was just certified and ratified in December of 2020.

Appendix 2: Input Format: The Language of Grades

The text

```
<category> := exam | final-exam | hw | lw | reading | engagement

<score> := numeric value
```

expresses the formatting of valid input in a formal way. It is a precise, very concise way of expressing the syntactical rules of a ‘language’, i.e., the expected sequence of letters and symbols.

The course *CSCE 433: Formal Language and Automata*, is an elective course that covers this rich field of study. All of the CSCE 120/121 instructors find this course to be interesting, useful, and fun.

Appendix 3: Pipes: Another Input Redirection Tool

Another version of this "hack" that doesn't necessarily even require a file uses a "pipe" to redirect the standard output stream of one program to the standard input stream of another program. The cat program performs file concatenation. In this case, we (ab)use it to output the contents of a file by concatenating the file to standard output. The '|' symbol redirects the output of the program on the left to the input of the program on the right.

```
$ cat test_complete_11.txt | ./a.out
enter grades as <category> <score>
  <category> := exam | final-exam | hw | lw | reading |
engagement
     <score> := numeric value
enter an empty line to end input
summary:
      exam average: 63.6
        hw average: 77.36
        lw average: 100
           reading: 100
        engagement: 38.65
    ---------------
    weighted total: 73.3165
final letter grade: C
```

If we just want to provide data to standard input, but don't want or need to store it in a file, we can use echo to produce the output that we need. This is useful for quickly and easily changing small parts of long inputs since, instead of editing a file or retyping input interactively, we can use the terminal history (hit up in the terminal to see previous commands) and edit the input. The echo program copies its input to standard output.

The -e flag enables interpretation of backslash (\) escape sequences, e.g. \n for the newline character.

```
$ echo -e "hw 100\nhw 80\n" | ./a.out
enter grades as <category> <score>
  <category> := exam | final-exam | hw | lw | reading |
engagement
     <score> := numeric value
enter an empty line to end input
summary:
      exam average: 0
        hw average: 90
        lw average: 0
           reading: 0
        engagement: 0
    ---------------
    weighted total: 36
final letter grade: F
```

[Learn more about redirection in a shell script language such as Bash]

.