Due Date: Feb 28, 2020 @ 11:59:59

File I/O Introduction – Summarize Grades

This assignment will focus on File I/O operations. We're going to write a program that reads a file with student information and their homework scores. You will manipulate that data to compute the total score and percentage for each student and then write a summary file with the computed scores for each student. Along the way we're going to fix some style issues with the names.

You will submit your work to Gradescope and it will be both auto graded and human graded. The autograder will compile your code with g++-7.4.0. Be sure to watch Piazza for any clarifications or updates to the assignment.

Don't forget the Assignment 3 Canvas Quiz!
Your assignment submission is not complete without the quiz.

Your submission must consist of 3 files: **SummarizeGrades.h**, **SummarizeGrades.cpp** and **demo.cpp**. The first being the header file, the second being the implementation of the required functions and third being the **main()** function implementation.

For *SummarizeGrades.h:*

- Be sure to have include guards.
- Don't import anything that is unnecessary.
- Include file header comments that have your name, student id, a brief description of the functionality, and be sure to cite any resource (site or person) that you used in your implementation.
- Each function should have appropriate function header comments.
  - You can use javadoc style (see doxygen) or another style but be consistent.
  - Be complete: good description of the function, parameters, and return values.
- For this assignment, no classes and no inline methods.
- Structure Name has the following data members
  1. firstName (string)
  2. lastName (string)

For *SummarizeGrades.cpp:*

- All the function bodies should be here.
- Don't clutter your code with useless inline comments (it is a code smell [Inline Comments in the Code is a Smell, but Document the Why], unless it is the why).
- Follow normal programming practices for spacing, naming, etc.: Be reasonable and consistent.
- Be sure to avoid redundant code

Functions to Implement

*void **readGradeFIle**(const string inputFilepath, int *numberOfStudents, int *numberOfAssignments, map<int, Name> &studentNames, map<int, vector<int>> &studentScores)*

This function takes the path of the input file and references to some data structures that will hold the student names and scores for other functions to process. It also contains pointers to variables that will keep track of the number of students and assignments. Sample unformatted gradebook file is provided for reference.

The unformatted gradebook file has the following structure:

- The first line has the keyword *number_of_students* followed by an integer indicating the number of students.
- The second line has the keyword *number_of_assignments* followed by an integer indicating the total number of assignments for the course
- The third line contains three keywords - *student_number*, *first_name*, *last_name* and then integer values to indicate the maximum possible number of points for each assignment. Note all assignments have *10* possible points. There will be one '10' listed for each assignment.
- The fourth and all additional lines will have data for one student. First will be a 5-digit student number followed by their first and last name. Then scores for each of the assignments are listed.
- You may assume that there are no duplicate records and any file we use to test your work will follow this format exactly; however, the number of students and the number of assignments will vary.

Your code should do the following:

- Open the file for reading and process the file contents. Keywords can be ignored.
- Read the number of students and assignments and store those values in the *numberOfStudents* and *numberOfAssignments* variables respectively.
- Then process each of the student. Store the student name in the *studentNames* map with key as *student ID* and value as a *structure {first name, last name}*.
- Store the scores of each student in the *studentScores* map with key as *student ID* and value as a *vector of ints* to hold the scores.
- **Remember to close the file when finished.**

*void **formatCaseOfNames**(map<int, Name> &names)*

Some of the student names in the gradebook file are lower case, some are uppercase, and some are capitalized. This is unacceptable. This function takes the student names map and changes the first and last name strings in the struct (values in map) so that the first letter of each string is a capital letter and all other letters are lower case.
Hint: there are lots of ways to do this, but one simple way is to remember that characters are stored as integer values internally. You can cast these to integers, do math to switch the case and then cast them back to characters. Try printing out the integer values of some characters to test this idea.

*void **computeTotalAndPercent**(map<int, vector<int>> &scores, map<int, int> &total, map<int, float> &percent)*

This function takes the scores map and computes the total score (the sum of all points earned) as well as the final percentage as a float and stores these computed results in *total* and *percent* maps respectively where key is the student ID. Remember all assignments are worth *10* points each. Percentages should be computed as floats between 0 and 100.

*void **writeFormattedGrades**(const string outputFilepath, map<int, Name> &names, map<int, int> &total, map<int, float> &percent)*

Now that we've done all the hard work, it's time to write the results to a file. However, it's important that these results look good. Please see the *sample output file formatted_grades.txt* for an example. Each line should begin with the *student's last name* followed by a *comma, a space* and then their *first name*. After that write their total score and percentage to the file. Scores should be aligned so that the *ones* digit of every total falls in the ***22nd column***. You may assume that the name length (last name + first name) is always <= 15 characters and the total score will take max 3 characters. Percentages should be values between 0 and 100 written to **one decimal place of precision**. This means that even if the percentage is *100*, it will be reported as *100.0*. The decimal point goes in the ***28th column***. See the example output. We recommend using IO manipulators such as ***setw*** and ***setprecision***. Your output file should look **exactly** like the sample file. You can use *diff* (linux) utility to compare two files from the command line.
Remember to close the file when you are finished.

demo.cpp

You must create a program (place it in demo.cpp) that tests all the functions you implemented in SummarizeGrades.cpp. Your demo.cpp should have:

- A main function.

- There must be at least 1 call to each of the functions described above for SummarizeGrades.h.
- You may use the sample unformatted gradebook file provided to test your functions e.g. After calling *Read_Grade_File* function, you should print to console numberOfStudents, numberOfAssignments and the names and scores map to verify whether you read the data correctly.
- **This file won't be used as a part of auto grading and hence no fixed output is required**.

Makefile

Your submission must include a *Makefile*. The minimal requirements are:

- Compile using the demo executable with the *make* command.
- Use appropriate variables for the compiler and compiler options.
- It must contain a clean rule.
- The demo executable must be named something meaningful (not *a.out*).

README.md

Your submission must include a README.md file (see https://guides.github.com/features/mastering-markdown/). The file should give a brief description of the program, the organization of the code, how to compile the code, and how to run the program.

Submission

You will upload all the required files to Gradescope. Reminder to complete the Assignment 3 Canvas quiz. There are no limits on the number of submissions. See the syllabus for the details about late submissions.

Grading

For this assignment, half of the marks will come from the auto grader. For this assignment, none of the test details have been hidden, but there may be hidden tests in future assignments.

The other half of the marks will come from human grading of the submission. Your files will be checked for style and to ensure that they meet the requirements described above. In addition, all submitted files should contain a header (comments or otherwise) that, at a minimum, give a brief description of the file, your name and wisc id.

HAPPY CODING!