CSCI 1300 Introduction to Computer Programming
Instructor: Knox
Assignment 6
Due Friday, March 10, by 12:30 pm

This assignment requires you to write two files: *main.cpp* and *Assignment6.cpp*. There are three functions that you are required to write for this assignment. All these should be written in a file called *Assignment6.cpp*. If you write more functions than the three that are required, these should be kept in Assignment6.cpp as well. Your main.cpp file should just implement your main() function. Linking the two files can be done with **`#include "Assignment6.cpp"`** in your main file.

Once you have your code running on your virtual machine (VM), you must zip the file containing your functions into a .zip file and submit that file to the autograder COG. You **must also** submit your code (both main.cpp and Assignment6.cpp in the same .zip file) to Moodle to get full credit for the assignment.

Submitting Your Code to the Autograder:
Before you submit your code to COG, make sure it runs on your computer. If it doesn't run on the VM, it won't run on COG. The computer science autograder, known as COG, can be found here: `https://web-cog-csci1300.cs.colorado.edu`

- Login to COG using your identikey and password.
- Select the CSCI1300 - Assignment # from the dropdown.
- Upload your .zip file and click Submit.

Your file within the .zip must be named Assignment6*.cpp* for the grading script to run. *Your main.cpp file is not needed in your submission to COG.* COG will run its tests and display the results in the window below the *Submit* button. If your code doesn't run correctly on COG, read the error messages carefully, correct the mistakes in your code, and upload a new file. You can modify your code and resubmit as many times as you need to, up until the assignment due date.

Submitting Your Code to Moodle:
You must also submit your code to Moodle to get full credit for the assignment, even if the computer science auto-grader gives you a perfect score.

Comments at the top of your source files should include your name, recitation TA, and the assignment and problem number. **Please also include comments in your code submission to describe what your code is doing. TAs will be checking that your code has comments.**

Upload one .zip file to Moodle containing <u>two files</u>: your own <u>main.cpp</u> *and* <u>Assignment6.cpp</u> *which was uploaded to COG.*

Interview Grading
For this assignment you'll need to schedule a slot for interview grading with your TA on Moodle.

Part 1. Count the number of lines in a file.

Write a function

```
int CountLines(string filename)
```

that takes in the filename as a parameter and returns the number of lines in the file. Lines are separated the newline character '\n'.

Part 2. Parse a file and store the values in a variable.

For this task you'll be given a file, containing a list of students and their scores in K subjects, where K is an arbitrary number. All of these values will be separated by commas. There will be one student record (Name + K scores) per line. The entire file will therefore be of this format

```
Name1, Score1, Score2, Score3 … ScoreK
Name2, Score1, Score2, Score3 … ScoreK
.
.
.
NameN, Score1, Score2, Score3 … ScoreK
```

Your task will be to read the file, parse it into its individual components and populate two arrays with the names of each student and their average score respectively.

You will need to write the function-

```
int ReadScores(string filename, string names[], float avg_scores[],
                            int array_size)
```

Where

- `filename` is the name of the input file as specified above
- `names[]` is an initially empty array of student names
- `avg_scores[]` is an initially empty array of student's average scores.
- `array_size` is the size of the arrays `avg_score` and `names`

Your function will return the number of lines read.

Task Details
You'll first need to initialize every element in the name array with the empty string **""**. Similarly every element in the **avg_score** array will be initialized with -1.

You will need to read each line in the file and parse the line of data to obtain all the values. Place the name into the names array and process all the scores to calculate an average to be placed into the corresponding position of the `avg_scores` array.

Part 3. Writing back to a file.

Given an array of **names** and associated averages in the **avg_score** array (as read from file in part 2), sort the data into ascending order by name and write the sorted data to the given filename.

You will need to the write the function

```
void WriteGrades( string filename, string names[], float avg_scores[],
int n_students)
```

Where

- `filename` is the name of the file that you shall be writing to
- `names[] i`s the array containing student names.
- `avg_scores[]` is the array containing the average scores for each student.
- `n_students` is the number of students.

Task Details

For every name in the names array, compute the grade from the corresponding average score. Remember the average score for the student in `names[i]` is stored in `avg_scores[i].`

For each student write a single line into the file with format given below. The format describes a three fields separated by commas. The nomenclature "<name>" means to place the name of the student in that location. Similarly, write the other two fields in place of the "<..>" in the format. (The < and > are not included in the output)

```
Format:  <Name>, <average score>, <grade>
```

Compute the grade using the following rubric:

For scores greater than and equal to 90
    Grade is A
For scores greater than and equal to 80 and less than 90
    Grades is B
For scores greater than and equal to 70 and less than 80
    Grades is C
For scores greater than and equal to 60 and less than 70
    Grades is D
For scores less than 60
    Grades is F