CSCI 201 – Computer Science 1

Final Project: An interactive program that processes student data.
Part I. Due on Sunday April 14.

The Problem. In this assignment, we have to compute the letter grades for a course. The data is a collection of student records stored in a file. Each record consists of a name(up to 20 characters), ID (8 characters), the scores of 10 homework assignments, a score for class participation, and the scores of three midterm tests and the final test (all positive integers). The maximum score a student can earn on any of these is 100. A sample file with 2 records is shown below:

```
Pondicherry,John XPQ23456 91 48 76 0 23 91 91 84 95 93 81 76 84 91 73
Quincy,Jane XPL27856 81 58 76 40 73 49 91 84 89 93 90 79 94 81 73
```

Processing the data. When the data for each student is processed, the following information is computed (displayed here in a single line):

```
<name>, <id>, <hwork&particpation score> <test score> <total score> <letter grade>
```

The output for the above input would look like this:

```
Pondicherry,John  XPQ23456 38.65 39.7 78.35 C
Quincy,Jane       XPL27856 39.2  40   79.2  C
```

The rules for calculating scores and the letter grade are as follows:

- The aggregate score on the homework assignments and class participation (maximum of 50). The best nine homework scores and the class participation score are used to compute the score for the assignments and class participation. All of the assignments and the class participation have equal weight; the worst homework score is discarded and the other nine are added to the class participation score; this sum is scaled down so that the maximum is 50. (The sum of the best nine scores can be calculated as follows: *Keep track of both the sum and the minimum when you read from the file; at the end subtract the minimum score from the sum.*)
- The aggregate score on the tests (maximum 50).All the midterm scores are equally weighted, whereas the final exam carries twice the weight of each midterm (e.g., if midterm scores are 60, 70, 80, and final is 90, aggregate test score is 39 out of 50).
- The total score (homeworks & participation + tests, maximum 100)
- A letter grade (A ($\geq$ 90), B ($\geq$ 80), C ($\geq$ 70), D ($\geq$ 60) or F ($<$ 60)).

Other requirements. We require a system that is flexible and does some simple checks on the data.
Data Checks. The only data check is that all the raw scores should be between 0 and 100. If any of the scores is outside this range, the corresponding aggregate score and the overall score are recorded as -1.0 and the student is awarded a grade of Z. As an example, the data record

```
Pondi,Joe XPQ23456 91 48 76 0 23 91 91 84 95 93 81 76 84 101 73
```

generates the output line

```
Pondi,Joe XPQ23456 38.65 -1.0 -1.0 Z
```

(Here the first 10 scores are homework scores, the eleventh is the participation score and the last four are test scores. The lowest homework score, i.e., 0, is dropped. One of the test scores is 101, so that function returns -1, and as a result the total score becomes -1, and the grade becomes 'Z'.)

Functional Requirements. The program displays a menu from which the user can make one of the following choices:
(a) Load a data file containing information about the students and the scores
(b) Display the stored data(raw data).
(c) Process the records currently loaded as per the rules specified for the first project. (The output file shows the student's name, ID, the aggregate score on the homework assignments and class participation (maximum of 50), the aggregate score on the tests (maximum 50), the overall score (maximum 100), and a letter grade ($A(\geq 90), B(\geq 80), C(\geq 70), D(\geq 60)$ or $F(< 60)$). Data should be checked to ensure that it is in the specified range.)
(d) Search for a particular student, specified by name or ID
(e) Print the processed grades to a file in a tabular format as described for the first project
(f) Add a student to the classlist
(g) Change a student's test scores
(h) Save the (modified) raw data(name, id, test scores and homework scores) to a specified file and exit the program.

Part I of Project I is to be done in two stages.

Stage 1. In Stage 1, the following items must be completed:
Q 1 *The framework for the UI.* Create a program that displays the menu options, accepts the user choices and invokes the appropriate stub. See the file `FinalProject/Stage1/Payroll.cpp` for an example. The functions for each menu operation are dummy operations(stubs). Generate a Script file showing source, compilation and test.
Q 2 *Analyzing the problem:* We shall use an Object-Oriented approach to accomplish this. In the Object-Oriented approach, we first look for nouns. List all the nouns you can find in the above description. Identify all the nouns that will become classes, and explain why the other nouns will not become classes. Create a UML diagram for each class.

Stage 2 Populating the data collection. Study the files in
`/FinalProject/Stage2` as you work on this. Add the following features:
(a) *Facility for storing data.* Define the class student that can store all the data for a student: name, id, test scores, 10 homework scores(stored in an array), participation scores, aggregated test and homework scores and the letter grade.
(b) *Storing a collection of students.* Define the class studentList to store data on all the students. Use a `Vector` to store all the students.
(c) *A function in* `main` *to load the data file.* The involves the following steps:
(i) getting file name and opening the file
(ii) reading the data from each line into a student object
(iii) adding the student object to the list.
The first two steps involve getting user input and dealing with format of stored information and are therefore done in the menu function. The last step involves adding an item to the collection, and is done in the list class.

(d) *A function in* `main` *for displaying the data.* All the data will be displayed, one record per line. The main program calls the StudentList object to display all the students, and the display method in the list calls the display on each student object.

Q 3 Create design documents explaining (i) how the data file will be loaded, and (ii) how it will be displayed. **See the sample design document posted on D2L for the payroll example.** These documents should have 5 columns as in the payroll example.
Q 4 Generate a script file showing source, compilation and test. Your tests should try out different scenarios that can occur. Here are some scenarios to consider: What if the input file has too many students? What if the file has no students?

What to turn in (Part I) Create a folder `FinalProjectPart1`. Upload answers for all four questions listed above.