Assignment 5: Computing Course Grades

Due: 23:59, Fri 11 Dec 2015 Full marks: 100

Introduction

In a university course, there are three different types of students and their grading criteria are not the same. *Normal students* are assessed using six assignments (assignments 1–6, in equal weight). They fail if their average assignment score is below 50. *Graduate students* are assessed using six assignments (72%) and also a quiz (28%). They fail if the total score is below 60 *or* the quiz score is below 40. *Part-time graduate students* are like graduate students, but they fail only if the total score is below 55 *or* the quiz score is below 35. In this assignment, you will apply inheritance and polymorphism to write three classes `Student`, `GradStudent`, and `PTGradStudent` to model normal students, graduate students, and part-time graduate students respectively, and a client program to read in the assignments and quiz (if any) scores of the students from a data file and determine whether each student passes or fails the course.

Program Specification

You have to write your program in four source files (1) `Student.cpp`, (2) `GradStudent.cpp`, (3) `PTGradStudent.cpp`, and (4) `calGrade.cpp`. The first three files are for implementing the classes `Student`, `GradStudent`, and `PTGradStudent` respectively, organized in an inheritance hierarchy. The fourth is a client program of these classes. You are given the interface of the three classes in the header files `Student.h`, `GradStudent.h`, and `PTGradStudent.h`. Do *not* modify their contents. You are recommended to write the files in order (1, 2, 3, 4). *Implement the member functions of each class and test them individually one by one. Your four files will be graded separately*, so you should not mix the functionalities of them. Also, you *cannot declare any global variables* in *all* your source files (except `const` ones).

Class `Student` (`Student.cpp`)

The `Student` class models normal students. Its interface is given as follows.

```
class Student {
public:
    Student(string sid = "0000000000");
    string getID() const;
    double getAssignmentScore(int i) const;
    void setAssignmentScore(int i, double score);
    virtual double computeTotalScore() const;
    virtual string computeCourseGrade() const;
private:
    string id;
    double assignment[6];
};
```

Private Data Members

`string id;`
The student ID of the student, which is a string containing 10 digits.

`double assignment[6];`
An array of `double` for storing the six assignment scores of the student. Score of assignment 1 is stored in `assignment[0]`; score of assignment 2 is stored in `assignment[1]`; etc.

Public Constructor and Member Functions

`Student(string sid = "0000000000");`
The parameter `sid` is supposedly the student ID of the student for initializing the data member `id`. However, `sid` can be a string with any possible contents. You have to remove all non-digit characters from it. Then if it has not enough digits (< 10), add enough '0's to the front to make it 10-digit. If it has more than 10 digits, trim it to retain only the *rightmost* 10 digits. (E.g., suppose `sid` is "ab3453sdc58964569", then `id` should be initialized as "5358964569".) Besides, the scores of all six assignments should be initialized as 0.

`string getID() const;`
Returns the student ID of the student.

`double getAssignmentScore(int i) const;`
Returns the score of assignment `i` of the student. You can assume that `i` is always 1–6.

`void setAssignmentScore(int i, double score);`
Sets the score of assignment `i` to `score`. However, if `score` is less than 0 or greater than 100, then set it to 0 or 100 respectively. If `i` is not 1–6, then this member function does not set anything.

`virtual double computeTotalScore() const;`
Returns the total score of the student, which is the average of his/her assignment scores. (E.g., suppose the student scores 83, 82, 77, 65, 93, and 89 in the six assignments, then the member function should return 81.5.

`virtual string computeCourseGrade() const;`
Returns the string "Fail" if the total score of the student is smaller than 50; returns the string "Pass" otherwise.

Class `GradStudent` (`GradStudent.cpp`)

The `GradStudent` class models graduate students and is a subclass of `Student`. Its interface is given as follows.

```
class GradStudent : public Student {
public:
    GradStudent(string sid = "0000000000");
    double getQuizScore() const;
    void setQuizScore(double q);
    virtual double computeTotalScore() const;
    virtual string computeCourseGrade() const;
```

```
private:
    double quiz;
};
```

Private Data Members

`double quiz;`
The quiz score of the graduate student.

Public Constructor and Member Functions

`GradStudent(string sid = "0000000000");`
The parameter `sid` is supposedly the student ID of the graduate student. This constructor calls the constructor of its superclass `Student` and initializes the quiz score as 0.

`double getQuizScore() const;`
Returns the quiz score of the graduate student.

`void setQuizScore(double q);`
Sets the quiz score to `q`. However, if `q` is less than 0 or greater than 100, then set it to 0 or 100 respectively.

`virtual double computeTotalScore() const;`
This member function *overrides* the one in its superclass. It returns the total score of the graduate student, which is 72% of the average of his/her assignment scores plus 28% of his/her quiz score. (E.g., suppose the student scores 85, 70, 84, 93, 70, and 81 in the six assignments and 90 in the quiz, then the member function should return 83.16.

`virtual string computeCourseGrade() const;`
This member function *overrides* the one in its superclass. It returns the string "Fail" if the total score of the graduate student is smaller than 60 or the quiz score is smaller than 40; returns the string "Pass" otherwise.

Class `PTGradStudent` (`PTGradStudent.cpp`)

The `PTGradStudent` class models part-time graduate students and is a subclass of `GradStudent`. Its interface is given as follows.

```
class PTGradStudent : public GradStudent {
public:
    PTGradStudent(string sid = "0000000000");
    virtual string computeCourseGrade() const;
};
```

Public Constructor and Member Functions

`GradStudent(string sid = "0000000000");`
The parameter `sid` is supposedly the student ID of the part-time graduate student. This constructor simply calls the constructor of its superclass `GradStudent`.

`virtual string computeCourseGrade() const;`

This member function *overrides* the one in its superclass. It returns the string "Fail" if the total score of the part-time graduate student is smaller than 55 or the quiz score is smaller than 35; returns the string "Pass" otherwise.

Client Program (`calGrade.cpp`)

Your main program is a client of the three classes `Student`, `GradStudent`, and `PTGradStudent` with the following flow.

1. The program starts with opening the data file **data.txt** for reading the students' scores for the course. The data file has the following file format:
   - The first line contains a positive integer n denoting the number of students in a course. You can assume that n is at most 200.
   - Subsequently, there are n lines of data. Each line of data denotes the information of a student and contains eight or nine space-delimited fields, depending on the type of the student.
   - The first field denotes the student type, which must be either "N", "G", or "PT", denoting normal students, graduate students, or part-time graduate students respectively. The second field is the student ID.
   - The third to eighth fields are the scores of assignments 1–6. The ninth field exists for graduate students only (including part-time), denoting the quiz score. These fields must always be numbers (can be non-integers).

   You can see some sample **data.txt** in the next section (Program Output).
2. In case the file **data.txt** cannot be opened successfully, print a warning message *"Cannot open data file. Program exit!"* and then terminate program execution with the function call `exit(1)`.
3. Create an array of 200 pointers to (super)class `Student` as follows:

   ```
   Students *students[200];  // Array of superclass pointers
   ```

   Note that we conservatively create the array big enough for the largest case. The actual number of students is read from the data file.
4. Read the data from **data.txt**. Create objects of `Student`, `GradStudent`, or `PTGradStudent` accordingly, and set up the pointers `students[i]` to aim at these objects one by one. To create the objects and set up the pointers, you can write either one of the followings:
   - `students[i] = new Student(…);`
   - `students[i] = new GradStudent(…);`
   - `students[i] = new PTGradStudent(…);`

   Note that (1) the constructors are responsible for handling the case where a student ID is not 10-digit long; and (2) the `new` operator returns a pointer which can be assigned to `students[i]`.
5. Set the assignments' and quiz scores (if applicable) for the objects accordingly. The `setAssignmentScore(…)` and `setQuizScore(…)` member functions are responsible for handling the case where the scores are outside the range 0–100.
6. Then, the program should call the `virtual` member functions `computeTotalScore()` and `computeCourseGrade()` of the objects to compute and print out the total scores and pass/fail status of all the students in the following format:
   - There are n+1 lines of output, where n is the number of students. The first line is a header line and the subsequent n lines are the results of the n students.

- In each of the subsequent n lines, there are three fields separated by spaces. The first field is the 10-digit student ID. The second field is the total score of the student in *two decimal places*. The third field is either "Pass" or "Fail", denoting whether the student passes the course or not. You can use `cout << fixed << setprecision(2);` to control printing the total scores in two decimal places.

You can see some sample output of the client program in the next section (Program Output).

Program Output

This client program has no user input. All data should be obtained from the file **data.txt**. The following shows some sample program output given some sample data files. You can try the provided sample program with some other data files as well. *Your program output should be exactly the same as the sample program* (i.e., same text, same symbols, same letter case, same number of spaces, etc.). Otherwise, it will be considered as *wrong*, even if you have computed the correct result. Note that *all printout should be printed from the client program* (`calGrade.cpp`), not the classes.

Sample Data File data.txt	Sample Program Output
5 N 46709394 87 78 90 92 64 70.5 PT 5201314 90 95 87 50 59 64 89 G 25277177 77 66 39 53 69 80 43 N 25266366 47 46 40 67 57 42.1 N 1234567890 100 95 80 91 92 82	Results of 5 students: 0046709394 80.25 Pass 0005201314 78.32 Pass 0025277177 58.12 Fail 0025266366 49.85 Fail 1234567890 90.00 Pass
10 N 519a2b932838 93.5 54 45.5 63.5 82.5 60.5 PT 29761ay21 59.5 80.5 77 71 58 65.5 60.5 G 13443579c07w9d6 34 44.5 33 60.5 47 81.5 28.5 N 6b6296 39.5 64 86 90.5 48.5 65 G 8Z8636 44.5 60 35.5 90 53 75 52 N 252x1P26 62.5 28.5 29 70.5 83.5 48 PT 13165 100 26.5 43.5 30 35.5 -4 27.5 N 20970 37.5 43 65 68.5 82 68.5 N R36T8X31 100 71 51 90.5 80.5 92 N 509dC0 101 97.5 77.5 53.5 95 31.5	Results of 10 students: 5192932838 66.58 Pass 0002976121 66.32 Pass 4435790796 44.04 Fail 0000066296 65.58 Pass 0000088636 57.52 Fail 0000252126 53.67 Pass 0000013165 35.96 Fail 0000020970 60.75 Pass 0000036831 80.83 Pass 0000005090 75.83 Pass

Submission and Marking

- Your program file names should be `Student.cpp`, `GradStudent.cpp`, `PTGradStudent.cpp`, and `calGrade.cpp`. Submit the four files in Blackboard (https://elearn.cuhk.edu.hk/). You do not have to submit the `.h` files.
- Insert your name, student ID, and e-mail address as comments at the beginning of all your source files.
- Besides the above information, your program should further *include suitable comments as documentation*.
- You can submit your assignment multiple times. Only the latest submission counts.
- Your program should be *free of compilation errors and warnings*.
- *Plagiarism is strictly monitored and heavily punished if proven.* Lending your work to others is subjected to the same penalty as the copier.