Programming Design, Spring 2016
Homework 10

Instructor: Ling-Chieh Kung
Department of Information Management
National Taiwan University

Please upload one PDF file for Problem 1 and two CPP files for Problems 2 and 3 (optional) to PDOGS at http://pdogs.ntu.im/judge/. Each student must submit her/his individual work. No hard copy. No late submission. The due time of this homework is ***2:00 am, May 16, 2016***. Please answer in either English or Chinese. The maximum point of this homework is 120.

Before you start, please read Sections 22.1-22.6 and Chapters 9 and 10 of the textbook.[1]

The TA who generates the testing data and grades this homework is ***Parker Chiang***.

Problem 1

(20 points; 5 points each) Consider Problem 2 below.

(a) (10 points) Consider the class `Course`. Explain why a programmer should implement its copy constructor rather than using the default copy constructor. Then implement an appropriate copy constructor (if you did so in Problem 2, simply copy and paste).

(b) (5 points) Consider the class `CourseGrade`. One alternative design is to replace `Course* ptrCourse` by `int courseId`. As course IDs do not repeat, this still allow us to identify the correct course information given an ID. Then why do we use a pointer? Explain the pros and cons of replacing the address by the ID.

(c) (5 points) Continue from Part (b). Another alternative design is to replace `Course* ptrCourse` by `Course course`. Explain the pros and cons of replacing the address by an object.

Problem 2

(55 points) In an enrollment system, three types of information are records. First is a set of courses, in which each course has its course ID, number of credits, and course name. Second is a set of students, in which each student has her/his student ID and name. Finally, we have a set of course-student-grade tuples, in which each tuple records a grade earned by a student in a course. In this problem, let's try to use classes to construct a data structure storing and processing this information. To do so, we define the following classes (incompletely):

```cpp
class Course
{
private:
  int id;
  int credit;
  char* name;
};
class CourseGrade
{
private:
  Course* ptrCourse;
  int grade;
```

[1]The textbook is *C++ How to Program: Late Objects Version* by Deitel and Deitel, seventh edition.

```
};
class Student
{
private:
  int id;
  char* name;
  CourseGrade* cg[10];
};
```

Each `Course` object represents a course, and each `Student` object represents a student. You will need to create two arrays, one for `Course` and one for `Student`. As a students can take at most ten courses (to simply this problem), we add a pointer array `cg` for `CourseGrade` to each student. Each element in `cg` is a `CourseGrade` pointer pointing to a `CourseGrade` object, which contains a pointer `ptrCourse` pointing to a `Course` object in the course array and the grade earned by the student in that course. Let's assume that all the grades are available, i.e., there is no `CourseGrade` whose `grade` is missing.

Note 1. *In* `Student`*, why do we use* `CourseGrade*` *rather than simply* `CourseGrade`*? First, we want to save space: For students who do not take 10 courses, we do not need to create 10* `CourseGrade` *objects. Second, we want to delay the invocation of constructor for* `CourseGrade`*. Following the same idea, it is your discretion whether to create object arrays or pointer arrays. An object array is easy to implement and access, but a pointer array allows you to delay the invocation of constructors.*

You will be given the information of n courses, m students, and k course-student-grade tuples. Obviously, we have $k \leq nm$, and in practice $k \ll nm$. This explains why we do not create an $n \times m$ matrix to store all the grades information: The matrix will be sparse and waste a lot of space. With the current data structure, we only need $10m$ `CourseGrade` variables, which means $20m$ integers to store all grades. As long as $n > 20$, we may save space.

Given the information, your program should rank all students according to their average grades, from high to low, and print out their names accordingly. If multiple students have the same average grades, rank them in the alphabetical order (Amy gets a higher rank than Bob, etc.).[2]

Input/output formats

There are 15 input files. In each file, there are $k+3$ lines. The first line contains a positive integer $n \leq 1000$ and then n ID-credit-name tuples for courses. Each course tuple is given in a pair of parentheses, where two values are separated by a comma. There is no white space in a tuple. There is no white space between consecutive tuples. A course ID is a positive integer between 1 and 99999. A course name contains at most 50 English letters (therefore, no white space). Different courses have different IDs and names (case-insensitive, so there will not be two courses "Calculus" and "calculus"). A course credit is either 1, 2, 3, or 4. For example,

```
3 (10001,3,Programming)(10002,3,Calculus)(201,2,History)
```

lists the information of three courses. This line contains at most 5000 characters in total.

The second line contains a positive integer $m \leq 1000$ and then m ID-name tuple for students. Each student tuple is given in a pair of parentheses, where two values are separated by a comma. There is no white space in a tuple. There is no white space between consecutive tuples. A student ID is a positive integer between 1 and 99999. A student name contains at most 50 English letters (therefore, no white space). Different courses have different IDs and names (case-insensitive, so there will not be two students "ikuta" and "Ikuta"). For example,

```
4 (1,Amy)(2,John)(3,Tom)(4,Bob)
```

lists the information of four students. This line contains at most 5000 characters in total.

[2]Simply use `strcmp` or `strncmp` to get the alphabetical order!

Note 2. *Suppose that there is a line of input which consists of an integer, a white space, and then a string. To read this line into an integer and a C string, we probably write:*

```
int i = 0;
char s[100] = {0};
cin >> i;
cin.getline(s, 100);
cout << i << "---" << s << "\n";
```

If you try this, and type

```
123 Hi
```

what you will see is

```
123--- Hi
```

where in `s` *there is a weird white space. The reason is the following. After the execution of the* `cin >> i;` *statement, the input cursor stops at the first non-integer character, which is the white space between* `123` *and* `Hi`*. Then as we read things into a C string, the white space is treated as a part of the string and stored into* `s`*.*

To solve this problem, one way is to insert a statement `cin.ignore();` *in between* `cin >> i;` *and* `cin.getline(s, 100);`*. This statement asks the input cursor to skip one character, which is the white space we do not want. Then* `s` *will contain exactly the string we need.*

The third line contains an integer $k \leq 10m$. Then each of the fourth to $(k+3)$th line contains three integers, which are a course ID, a student ID, and a grade. They together represent a course-student-grade tuple. The course ID must be the ID of an existing course. The student ID must be the ID of an existing student. The grade must be within 0 and 100. Two consecutive values are separated by a comma. A student takes at least one course. No two tuples are about the same pair of course and student. For example,

```
7
10001,1,90
10001,4,80
10002,1,92
10001,3,77
10002,3,83
10001,2,20
201,1,88
```

means that Amy got 90 in Programming, John got 80 in Programming, etc.

Given the input, you need to store the information into objects of the three classes defined in this problem. After you store this information, you should calculate the average grades for each student (weighted by course credits, of course), rank students from high to low, and then print out their names, separated by white spaces.

In the example given above, Amy, John, Tom, and Bob have their average grades as 90.25, 20, 80, and 80, respectively. A student takes at least one course. If multiple students have the same average grades, rank them in the alphabetical order (Amy gets a higher rank than Bob, etc.). Therefore, the output should be

```
Amy Bob Tom John
```

What should be in your source file

Your .cpp source file should contain C++ codes that will both read testing data and complete the above task. For this problem, you are allowed to use only techniques covered so far. NO other techniques are allowed. Finally, you should write relevant comments for your codes.

Grading criteria

You must use the given classes to store the given input information. If you fail to do so, you will get no point. If you do, you will be graded according to the following rule:

- 45 points will be based on the correctness of your output. PDOGS will compile your program, feed testing data into your program, and check the correctness of your outputs. Each fully correct set of outputs gives you 2 points.
- 10 points will be based on how you write your program, including the logic and format. Please try to write a robust, efficient, and easy-to-read program.

Problem 3

(45 points) This problem is a complication of Problem 2. Now a course name and a student name may contain white spaces, and course-student-grade tuples may be given with IDs or names. The objective is also changed. Now a list of p student IDs will be given. The program should then $\binom{p}{2}$ pairs of students and find the number of dominant relationships. Let $x^i = (x^i_1, x^i_2, ..., x^i_n)$ be the grade vector of student i, where x^i_j is her/his grade in course j. If she/he did not take that course, let $x^i_j = -1$. We say student i_1 dominates student i_2 if $x^{i_1}_j > x^{i_2}_j$ for all j such that $x^{i_1}_j \neq -1$ and $x^{i_2}_j \neq -1$. We say there is a dominant relationship between students i_1 and i_2 if either i_1 dominates i_2 or i_2 dominates i_1.

In the example given in Problem 2, we have $x^{\text{Amy}} = (90, 92, 88)$, $x^{\text{John}} = (20, -1, -1)$, $x^{\text{Tom}} = (77, 83, -1)$, and $x^{\text{Bob}} = (80, -1, -1)$. Therefore, Amy dominates all the other three students, Bob dominates Tom and John, and Tom dominates John. Therefore, we have six dominate relationships among Amy, John, Tom, and Bob. Note that there may be no dominant relationship between two students. For example, if Mary got 80 in both Programming and Calculus, and Tom does not dominate Mary while Mary does not dominate Tom. Given a list of p student IDs, you should output the number of dominant relationships among these p students.

Input/output formats

There are 15 input files. In each file, there are $k + 4$ lines of input. The first $k + 3$ lines are given in the same way as in Problem 2, except that in the fourth to the $(k + 3)$th lines names may replace IDs. These given names and IDs are guaranteed to match given courses and students. The last line contains p students IDs. These must be non-repeating IDs of existing students. Two consecutive IDs are separated by a white space. Given the input, the output should be the number of dominant relationships among the p students. For example, if the output is

```
3 (10001,3,C Programming)(10002,3,Calculus)(20001,2,History)
5 (1,Amy Wang)(2,John Wong)(3,Tom Wing)(4,Bob Weng)(5, Mary Wung)
9
C Programming,1,90
10001,Bob Weng,80
10002,1,92
C Programming,3,77
10002,Tom Wing,83
10001,2,20
```

```
20001,Amy Wang,88
C Programming,Mary Wung,90
10002,5,80
5 3 4 1
```

then the output should be

```
4
```

Note that there is no dominant relationship between Tom and Mary and Amy and Mary.

What should be in your source file

Your .cpp source file should contain C++ codes that will both read testing data and complete the above task. For this problem, you are allowed to use any technique. You are not even required to use the classes defined in Problem 2. Finally, you should write relevant comments for your codes.

Grading criteria

45 points will be based on the correctness of your output. PDOGS will compile your program, feed testing data into your program, and check the correctness of your outputs. Each fully correct set of outputs gives you 3 points.