Programming Assignment - University Grade Management System (UGMS)

LP002 Data Structures

November 23, 2020

Important Milestones:

- Group Registration Deadline: **23:59 30 Nov. 2020**
- Submission Deadline: **23:59 14 Dec. 2020**

1 Introduction

You are asked to design and implement a University Grade Management System (UGMS), which can store and maintain the grade information of students. In particular, your implemented UGMS should be able to read the score reports of students from the given files. Each file will store the scores of ONE course. You should design the proper class types and choose the proper objects to store those data in your system. Meanwhile, your system should be able to calculate the final grade and Grade point average (GPA) for each student according to the grading scheme given by the university. Moreover, your system should print out the final result for each course, including the grades, the statistics of the course etc. In addition, your system should be able to execute queries from users and display the query results as well. There are TWO to THREE members (at most **THREE members**) in a group. You should state clearly what the responsibility of each member is.

1.1 Main tasks

Your UGMS shall fulfill the following main tasks:

1. *Reading Data File*: You need to read several data files, each of which contains the scores of a course (refer to Section 3.1).
2. *Storing Data*: After reading data files, you shall process them so that they can be stored in your UGMS in an efficient way so that it can support future queries (refer to Section 3.2).
3. *Output of results*: After dealing with the queries, your UGMS shall print out the query results in either ascending order or descending order according to the fields specified by the users (refer to Section 3.3).
4. *Querying*: Your UGMS shall support various queries from the users (as given in Section 3.4).

Details about the main tasks will be addressed in Section 3.

2 Objective

After the completion of this assignment, you shall be able to design a small but complete software by applying the sorting algorithms (such as merge sort, quick sort and heap sort) and the searching algorithms (e.g., Binary Search Tree). Moreover, you need to apply the algorithm analysis approach to evaluate the performance of your software.

3 Specification

Your UGMS system shall provide users with searching and sorting on a large number of student score records after reading the data records from given data files.

3.1 Read records

When the program first starts, it should ask the user for the file name of the data file. The prompt should look like this:

```
Please enter the database filename:
```

After read a file for a course, your program should display the prompt again and ask the user to input another score report.

```
Do you have another score report?
Press "Y" for Yes and "N" for No.
```

If the user chooses ”Y”es, then your program should ask the user to input the file name of the data file. If the input is ”N”o, then your program shall complete the input process.

Note that it is just a suggested user interface (UI). You are free to choose other UIs as you like. Also you can implement a graphic UI by Java.

3.1.1 Data File Format

Each score report keeps the score records for one course. The file has the following format:

(a) For each course, the first line contains the course code and the credit of the course, delimited by a comma.

(b) The next line contains the number of students, says N ($N \geq 1$), taking the course.

(c) Each of the next N lines contains a comma separated list of student's `Student surname`, `Student given name` `Student ID` and his/her obtained `Score`, where N is ranging from 1 to 10,000.

(d) You may assume there is no space before and after the comma delimiter and there is no trailing space after each line of data.

(e) Note that you **CANNOT** change the input data file format.

The following is an example of data file:

```
MA113,4
5
Ho,Ivan,1109853A-I011-0018,95.12
Leong,Wai Kit,1109853A-I011-0567,49.62
Hua,Peng,1109853M-I011-0232,75.24
Zhu,Danyang,1109853U-I011-0002,85.0
Zhen,Ben,1109853Z-I011-1234,43.46
```

As shown in this example, there are 5 students taken the course MA113 with credit 4. Each student has one score.

Other assumptions:

- The maximum number of data files is 100.

3.2 Store records

After reading the score reports successfully, your program should be able to store both the course information and student scores. Your shall design appropriate data structures so that they can support efficient queries and sorting outputs as specified later. In particular, your program may store the course information as follows:

Student:
Student Surname
Student Given name
Student ID
Score
Grade
Credit
GPA

Course:
Course Name
Course Credit
No. of students in this course

Note:

- The above two tables are not class diagrams and they just show what information your program may store.
- You need to design a class to sore the `Student` record and the `Course` record.
- You should specify enough space for each field defined in your class (i.e., designing appropriate data structures).
  - Student ID contains a fixed-length string with 18 characters (including letters, numbers and hyphen '-').
  - Student Surname and first name are varied-length strings, each with 2 to 20 characters (letters only).

Table 1: Grading Scheme in MUST (Suggested)

Scores	Grade	Points
93-100	A+	4.0
88-92	A	3.7
83-87	A-	3.3
78-82	B+	3.0
72-77	B	2.7
68-71	B-	2.3
63-67	C+	2.0
58-62	C	1.7
53-57	C-	1.3
50-52	D	1.0
40-49	F	0.0
≤ 39	O	0.0

- You can assume that the input data file always has the correct format.
- `Credit` shall include a list (an array) of the credits corresponding to the courses that the student has taken.
- `Grade` can be obtained by the university grading scheme (given by Table 1), which shows the conversion scheme from the score to grade and that from the grade to the points in our university.[1]
- The calculation of `GPA` will be given by Section 3.4.1.

3.3 Print out results

Your program should output a report (in a particular format) for a course specified by the user. Your program should let the user choose the course before display the result.

```
Please choose the course code first: LP002
```

After the course code is chosen (input by the user), a sample report is shown below:

```
Course Code: LP002
Credit: 4
Number of Students: 8
Name                    ID                      Score      Grade
--------------------    ------------------      -----      -----
Chan, Henry             1109853A-I011-0224      86         A-
Choi, Randy             1109853A-I011-0018      60         C
Ip, John                1109853A-I011-0456      66         C+
Lau, Mary               1109853U-I011-0002      75         B
Lee, Vito               1109853J-I011-0116      85         A-
```

[1]Note that you might convert a real-number score to an integer so that it can match to Grading Scheme in Table 1 according to `https://en.wikipedia.org/wiki/Rounding`. For example, 92.21 is rounded to 92 (A) and 92.5 is rounded to 93 (A+).

```
Leung, Terry             1109853P-I011-0955      80          B+
Ngai, Raymond            1109853Z-I011-0066      89          A
Tung, Tracy              1109853Q-I011-0016      51          D
```

3.3.1 Sorting on the report

You are required to display the report according to the surnames of the students, the student ID, the scores and grades in either an ascending order or a descending order. For example as shown in the previous sample report, the students are listed in ascending order lexicographically according to their surnames.

Note:

- To specify which field to be sorted and the sorting ways (ascending or descending), you can let user input it as a parameter.
- Users can specify their sorting according to the following fields: (1) Surname; (2) Student ID; (3) Score; (4) Grade.

An example input is shown as follows:

```
Please choose the sorting field: 1
(1) Surname; (2) ID; (3) Score; (4) Grade
```

```
Please choose the sorting ways: A
(A)scending order; (D)escending order
```

3.3.2 Statistics Reports

You should also print out the statistics of the report. In particular, your report should output the average score, the highest score and the lowest score in a course. Besides, your program should output a report showing the number of students within each grade. The possible grades are “A”, “A-“, “B+”, “B”, “B-“, “C+”, “C”, “C-“, “D”, and “F”. The report may have a similar output format to the following suggested report:

```
Course Code: SE121
Credit: 4

The average score: 67.5
The highest score: 89
The lowest score: 51

Course Grade Statistics:
  A+: 0
   A: 1
  A-: 2
  B+: 1
   B: 1
  B-: 0
  C+: 1
   C: 1
  C-: 0
   D: 1
   F: 0
```

3.4 Query on Score System

Your system shall provide users with the basic query and advanced query on the student grade records.

3.4.1 Calculate the GPA of a student

Your program should be able to list the GPA of all the courses taken by a student. The GPA can be calculated by the following equation:

$$GPA = \frac{\sum_{i=1}^{n} C_i \times P_i}{\sum_{i=1}^{n} C_i}, \tag{1}$$

where C_i denotes the credit for a course and C_i denotes the points that the student got in the course i, and n denotes the number of courses taken by the student (note that n is not necessarily equal to N).

For example, a student with SID “1109853A-I011-0118” who took the courses LP004, MA203 and CS001 and had the following scores: 61.5, 69.3 and 65.41, respectively. Then, his/her CPA can be calculated by the following equation:

$$GPA = \frac{1.7 \times 6 + 2.3 \times 4 + 2.0 \times 3}{6 + 4 + 3} = 1.95, \tag{2}$$

where we assume $C_1 = 6$ for course LP004, $C_2 = 4$ for course MA203 and $C_3 = 3$ for course CS001.

Hints

- You shall first find all the scores for him/her.
- You then calculate GPA based on the courses that he or she has taken.

3.4.2 Search the grades of all courses taken by a student

Your system shall be able to find all the grades and CPA for a student. For example, the user wants to list all the grades for a particular student with student ID “1109853A-I011-0118”. Then, your system should display the grades for all courses taken by this student.

A sample output is shown below:

```
Please input the student ID: 1109853A-I011-0118
LP004 C
MA203 B-
CS010 C+
GPA: 1.95
```

3.4.3 Sorting on GPA

Your program should be able to list the GPAs of all students in your system. Besides, your system shall be able to display the GPAs of students according to either the ascending order or the descending order.

A sample output is shown below:

```
Student ID                   Student Name             GPA
------------------           --------------           ----
1109853A-I011-0118           Chan, Henry              1.95
1109853B-I011-0284           Pan, Jianqin             2.32
1109853Z-I011-0184           Jiang, Feifei            3.40
```

3.4.4 Advanced Queries

(1) Your program should support the query by student surnames

For example, the user may want to know the grades and GPAs for all the students with surname "Wong". Then, your program should output the results like this:

```
Please input the student surname: Wong

2 records found in UGMS:

Tom Wong 1109853B-I011-0145
LP004 B+
MA203 A-
CS010 C+
GPA: 2.91

Joan Wong 1109853B-I011-0486
LP004 C
MA203 C-
CS010 B+
GPA: 2.21
```

Note that your system may support the query by either students' full names (including both surnames and first names) or students' first names. However, you are not compulsory to complete this function though the query on the surname is a MUST.

(2) Your program should support the query by student names with wild cards (*Bonus*)

For example, the user may want to know the grades and GPAs for all the students with last name started with "W*", where "*" denotes any one or more than one characters. Then, your program should display the records for all the students with last names as "Wang", "Wan", "Weng", "Wong", "Woo" and "Wu", etc. The perfect implementation of this part will lead to at most **3 bonus points**.

(3) Ranging query on GPA (*Bonus*)

For example, the user may want to know all the students whose GPAs are greater than 3.3.

```
Please input your ranging query condition on GPA: > 3.3

There are 3 students satisfying the condition:
```

```
Raymond Wong 1109853A-I011-0210
GPA: 3.93

Sophia Hon 1109853B-I011-0286
GPA: 3.45

John Lui 1109853B-I011-0486
GPA: 3.31
```

Note:

- You can specify the sorting manner either by yourself or by users.
- The perfect implementation of this part will lead to at most **3 bonus points**.

3.5 User Interface (UI)

You can design the user interface (UI) either based on windows console with text only (CUI) or graphical interfaces (GUI). In general, the CUI is easy to implement and less time consuming. Thus, we recommend you design and implement your program based on CUIs. However, you can also choose designing and implementing your program based on GUIs, which may be more time consuming. The perfect implementation based GUIs will lead to at most **4 bonus points**.

3.6 Performance

Your system shall be efficient in both searching and sorting. In particular, the running time for a single query of your system shall be less than **20 seconds** given that the number of records N is less than 10,000. **Every extra second leads to subtracting one point in your score** (note that the time for reading records will not be counted in the performance). Therefore, do use the faster algorithms (e.g. quick sort, merge sort and binary search trees, etc.) to implement your system.

3.7 Programming Language and Libraries

In this assignment, Java is the ONLY language that you can use. You can use either Java Standard Libraries (i.e., Java SE 11 or above) or the textbook library including algs4.jar and stdlib.jar. Both algs4.jar and stdlib.jar can be downloaded from the textbook website (`http://algs4.cs.princeton.edu/`).

4 Important Milestones

4.1 Phase I: Sorting and Output

In Phase I, you need to complete most of tasks specified in Sections 3.1, 3.2 and 3.3. In particular, your program should be able to achieve the following goals.

- Given input files, your program should process student records.
- Design appropriate data structures to manage student records.

- Display student records according to ascending order or descending order on a particular field specified by users.

4.2 Phase II: Query

In Phase II, you need to complete the remaining tasks specified in Section 3.4. In particular, your program should be able to achieve the following goals.

- Calculate GPA of a student.
- Search grades of a student.
- Sort on GPA of a student.
- Advanced queries.

4.3 Final report

Please submit a final report with your completed program codes. The report shall include:

(i) **Design report**

- The distributions of your tasks: please explicitly specify who is responsible for which part of the whole assignment.
- System design report: the class diagram of your design and the description of each class.
- Algorithm design report: e.g., the flow-chart diagram of your core algorithms, e.g., sorting and searching algorithms.
- Data structures: The concrete design of each class.
- Algorithm analysis: Your estimated complexity of your algorithms, e.g., your sorting algorithm is about $N \log N$. You should give the proof if necessary.

You shall write this document in word processing tools (such as MS Word, OpenOffice, WPS, etc.). Note that for compatibility, please convert your document to Portable Document Format (**PDF**).

(ii) **Your System Application**

1. A soft copy of your source program including all the components.
2. readme file (readme.txt) which contains:
   (a) group information
   (b) file list
   (c) file description
   (d) methods of compilation
   (e) method of execution
   (f) known bugs of your system
3. The special features of your system
4. A brief user manual of your system

Note that you shall submit the Java source codes only (not including `*.class` files). Each of your program shall follow the programming style as specified in Appendix A.

(iii) Compilation methods of your program and user manual.

Submission rules:

Step 1 Please compress all the required files into one zip package, i.e., *.zip file.

Step 2 Go to `http://moodle.must.edu.mo` to submit your assignments.

5 Grading Policy

1. Correctness and Performance (60%): we will verify the correctness and performance (running time) of your program by testing it through two data sets: (a) the provided data set; (b) another data set (not released).
   - Phase I counts for 30%;
   - Phase II counts for 30%.
2. Report (30%): Please refer to Section 4 for more details about the report.
3. Programming Style (10%): Style and efficiency of your program (comments are expected in the source code).
4. Late submission will lead to the score penalty.

6 Plagiarism

Programming is a creative work and the academic regulations that apply to plagiarizing prose also apply to plagiarizing code. Rights, Rules, and Responsibilities defines plagiarism as ”the use of any outside source without proper acknowledgment.” It ranges from ”verbatim copying” (e.g., cutting-and-pasting code) to ”thorough paraphrasing” (e.g., changing variable names or rearranging code). We use sophisticated tools to detect plagiarism and our teaching staff takes the issue very seriously.

We refer alleged academic violations (including plagiarism and abetting plagiarism) to the Discipline Committee under Academic Office. If found responsible, the typical penalty is the failure for this course plus whatever penalty that is stated in University Student Handbook.

Appendix A: Programming and Documentation Guidelines

Header comments:

You must include the following information at the top of every source file (.java file):

1. Assignment number
2. Name:
3. Student ID
4. Course and section number
5. email address
6. A short description of the content in the file or a short description of the corresponding program.

Describing Functions / Methods:

For all methods, and functions defined in your source program, you must describe them. If a function/method accepts parameters, you must also describe what each parameter is for. If a function/method returns a value, you must also describe what the returned value represents.

Program Readability:

1. Indenting code blocks - Statements that belong to the same block should start at the same column. You can use 2 spaces, 4 spaces, 8 spaces or tab to indent your codes.
2. Use space and empty lines to properly separate codes
3. Remove all unused variables
4. Each line in the source file, including comments, should not be longer than 80 characters. If a line is more than 80 characters long, split it into multiple lines.

A good example:

```
/****************************************************************
 * Assignment 1
 * Name : Alex Fung
 * Student ID: 1009853X-I011-YYYY
 * Section:LP002
 * email: alexfung@fakemail.com
 *
 * Description:
 *  This file contains the ThreeSum class, which counts
 *  the number of tuples equal to 0.
****************************************************************/

public class ThreeSum
{
  /*************************************************************
   *
   * Description:
   *   Brute-force method to compute the the number of
   *   tuples equal to 0 by checking condition:
   *   a[i] + a[j] + a[k] == 0
  *
  * Parameters:
  *   a: the array
  *
  * Return:
  *   count: the number satisfying the condition
  *
  ***************************************************************/

  public static int count(int[] a)
  {
    int N = a.length; // The size of array
    int count = 0;    // the result
    for (int i = 0; i < N; i++)
      for (int j = i+1; j < N; j++)
        for (int k = j+1; k < N; k++)
          if (a[i] + a[j] + a[k] == 0)
            count++;
    return count;
  }

  /**************************************************************
  * A testing client reads an array from standard input
  * and prints the number of tuples equal to zero.
  **************************************************************/
  public static void main(String[] args)
  {
    int[] a = StdArrayIO.readInt1D();
    StdOut.println(count(a));
  }
}
```