Lab 1: Why Databases?
Part II

Due dates:

- Lab 1-1: Friday, April 7, 11:59pm (soft), Monday, April 10, 10:00am (hard).
- Lab 1-2: Monday, April 10, 10:00am (hard).

Honor System

This part of the assignment is handed to each group separately, upon submission of the first part of the lab.

While each group will recieve the same assignment, I ask you to use the honor system and not directly convey the contents of the assignment to students from the teams that have not yet completed part I of the assignment. Please let other teams concentrate on finishing the first part, without worrying about the content of the second part.

Lab Assignment

The Task

In the second part of the assignment, you have to change your program to accommodate the changes in the input data format.

Recall that the original `students.txt` file had the following format:

```
StLastName, StFirstName, Grade, Classroom, Bus, GPA, TLastName, TFirstName
```

Generally speaking, if each teacher teaches in exactly one classroom, it is a bit of a waste of space to put teacher's name against the name of every student. Therefore, we now revise the format of the input file. Now, the original `students.txt` is replaced with a pair of files: `list.txt` and `teachers.txt`.

The format of `list.txt` is

```
StLastName, StFirstName, Grade, Classroom, Bus, GPA
```

A sample line from `list.txt` is

```
DROP, SHERMAN, 0, 104, 53, 2.67
```

("Sherman Drop is a kindergarden student assigned to classroom 104 who takes bus number 53 and whose GPA is 2.67.")

The format of `teachers.txt` is

```
TLastName, TFirstName, Classroom
```

A sample line from `teachers.txt` is

```
NIBLER,  JERLENE, 104
```

("Jerlene Nibler teaches in classroom 104.")

The data files are available on-line. The direct URLs for the files (they won't be linked to the web page until later) are:

`http://www.csc.calpoly.edu/∼dekhtyar/365-Spring2017/labs/lab1/list.txt`

`http://www.csc.calpoly.edu/∼dekhtyar/365-Spring2017/labs/lab1/teachers.txt`

The information in these files is exactly the same as in the original `students.txt` file, so answers to test searches in both versions of your program should be the same.

Your goal is to change the `schoolsearch` program you have developed for Part I of this assignment to handle the new input data formats. In addition, you are asked to design, and implement in the new program extra search facilities.

Specs

With the exception of the new functionality, described below, the new `schoolsearch` program shall have the same functionality as the `schoolsearch` program, working exactly the same way, i.e., it shall accept the same search commands and produce the same results.

The key difference is that now, instead of obtaining the data from a single file called `students.txt`, your program will read data from two files: `list.txt` and `teachers.txt`. Same assumptions about the location of the files (current directory) and error handling (minimal) apply.

Additional Functionality: Extended Search

You are asked to design and implement additional search functionality. In particular, current language of search instructions for the program allows one to search for information about a student, given student's last name; find the bus route of a student, given student's last name, list all students attending a class taught by a specified teacher; list all students in a specific grade and all students taking a specific bus route and find students with the highest and the lowest GPA in a given grade.

Three new searches need to be added to the program:

- **Requirement NR1.** Given a classroom number, list all students assigned to it.
- **Requirement NR2.** Given a classroom number, find the teacher (or teachers) teaching in it[1].
- **Requirement NR3.** Given a grade, find all teachers who teach it.
- **Requirement NR4.** Report the enrollments broken down by classroom (i.e., output a list of classrooms ordered by classroom number, with a total number of students in each of the classrooms).

Additional Functionality: Analytics

The final portion of the functionality for your new program is a bit of analytics.

We are interested in determining whether student performance somehow is in a relationship with some other factors. We use student GPA as a proxy for overall student perfromance. There are three observable factors that can possibly influence student performance, that are avilable to us in this data set. These factors are:

- The grade level of the student
- The teacher teaching the student
- The bus route the student is on[2]

Requirement NR5. Add to your program the commands that allow a data analyst to extract appropriate data to be able to analyze whether student GPAs are affected by the student's grades, student's teachers or the bus routes the students are on.

[1]In our test file, there in one teacher per classroom. However, your search algorithm cannot assume that it will always be the case.

[2]You might find it silly to consider that a bus route may somehow be a predictor of student performance in school, but in some places bus routes may be viewed as proxies for socio-economic status of the families of the students, which is a characteristic that is known to be in a relationship with student performance.

Note: I am being **vague** here **on purpose**. Part of the assignment is to figure out not just what the commands should look like, but also, what needs to be computed and how it needs to be reported.

The program you are creating is not supposed to perform any data analysis by itself, but the analytical commands you develop shall output the information that would allow a data analyst to study the questions of the relationship between a sepcific factor (from the list of three factors mentioned above) and the GPAs of students.

For both types of new commands (extended searches and analytics) you shall

- extend the language of search instructions to allow for these three searches to be specified by users;
- implement the functionality supporting each command.

Implementation notes

Obviously, there are two ways to do this task. One way is to start from scratch and build a brand new program. The other way is to adapt the first program to the new data format. The decision of what to do is left up to individual teams.

Tracing, Testing, and Deliverables

Data. The data files are available on-line. The direct URLs for the files (they won't be linked to the web page until later) are:

`http://www.csc.calpoly.edu/~dekhtyar/365-Spring2017/labs/lab1/list.txt`

`http://www.csc.calpoly.edu/~dekhtyar/365-Spring2017/labs/lab1/teachers.txt`

Tracing. For this assignment, your source code must incorporate *traceability information.* That is, for each component of your program, you must indicate *in a well-structured comment* which requirements from the specifications above (**R1**—**R13**, and **E1**) this component implements (i.e., *traces to*).

For example, if implementing in Java, a method `computeStudentInfo(..)` can be prefaced with a tracing comment as follows.

```
// Traceability: implements requirements R3, R4, R5

// ... add other comments here

public static <ReturnType> computStudentInfo(...) {
...
```

```
}
```

The new requirements for this part of the lab are **NR1** — **NR5**. Please include them in your traceability information.

Testing. Create a test suite for your program. The test suit must cover all requirements and test all possible *correct* behaviors of the program, as well as one or two incorrect behaviors (situations where Requirement **E1** is triggered). (Please note, commands that return no information, e.g., `G: 10` do not trigger Requirement **E1**, they simply return no answers. Such commands should also be a part of your test suit).

Each test case is a single command in the command language of the program. For purposes of submission, the test suite shall be represented as a single text file, which includes for each test case an annotation (rendered as a comment) specifying what it tests, and containing the tracing information to the requirement that is being tested.

For example, a simple test suite consisting for three commands can be rendered as follows:

```
// CSC 365. Spring 2017
// Alex Dekhtyar
// Lab 1-1 test suite

// TC-1
// Tests Requirements R3, R4
// short form command name, existing student
// expected output: HAVIR,BOBBIE,2,108,2.88,HAMER,GAVIN

S: HAVIR

// TC-2
// Tests Requirements R3, R4
// short form command name, non-existing student
// expected output: <empty line>

S: DEKHTYAR

//TC-3
// Tests Requirments R3, R13
// quit command
// expected output: program terminates

Q
```

For this part of the lab:

- keep your original test cases (the whole point for a collection of test cases is that they can be used to validate multiple versions of the same program).
- add new test cases that thoroughly test the new functionality covered in requirements **NR1—NR5**.
- place all test cases in one file as before.

Writeup. The write-up is a mandatory part of the lab. I recommend that at the beginning of work, you designate one student to prepare it. The write-up shall contain a brief outline of your team's implementation effort. I strongly recommend building your write-up as you go, rather then after the program has been completed. At the very least, the write-up shall contain the following:

- Your team name/number, list of team members;
- Initial decisions: programming language, environment;
- Notes on selected internal architecture: what data structures to use, for what purposes;
- Task log. For each task to be completed, list the name of the task, the student(s) performing it, start time, end time, total person-hours it took to complete. Choose the granularity wisely. You do not have to document every method or function.
- Notes on testing. When, who, how long, how many bugs found, how long it took to fix them.
- Final notes (anything else you want to share with me about your implementation)

For Part II writeup, also add the following information, where appropriate:

- Decisions you made on how to modify your Part 1 code to accommodate new input data. Which parts of the code were affected?
- Syntax and semantics of any additions to the query language, complete with examples.

Additionally, please include simple description of the new commands in the `README` file.

Deliverables. The full list of deliverables is specified below. **PLEASE**, make sure the names of each team member are on each deliverable file (except for program output). Also, **PLEASE** make sure that your files are named **EXACTLY AS SPECIFIED** in this document.

1. Your code. The file name shall be `schoolsearch.ext` where `ext` is the extension for the source code files of the programming language you have selected.

2. Any additional source code files. As needed.

3. Your test suite formatted as specified above. Name the test suite file `tests.txt`.

4. Output of running your program on the commands from **your** test suite. Call this file `tests.out`. You can simply paste the program output into a file, or use the Linux `script` command to record your session with your program.

5. Your writeup. The writeup shall be submitted in the form of a PDF document called `writeup1-1.pdf`.

6. `README`. Submit a `README` file specifying how to compile/run your program and what new commands it implements. If you want to, you can submit a `Makefile` or any other supplemental files that help compile/run your code.

7. `teachers.txt` and `list.txt`. Place these files in the submission directory.

Submission. Place all files you want to submit into one directory. `cd` that directory and either `zip` or `tar` and `gzip` the contents of this directory *while in the directory itself*[3].

Name your archive `lab1-2.zip` or `lab1-2.tar.gz`. When you believe you have satisfied all requirements of the lab, submit your archive using the following `handin command`:

```
$ handin dekhtyar lab01-2 <FILE>
```

Good Luck!

[3] I.e., DO NOT zip the directory from its parent.