Lab 1, Part a: Why Databases?

Assignment Preparation

You may work individually or in teams of up to two people.

The Task

The full lab assignment consists of two parts. The first part of the assignment is described below. The second part will be distributed after you have submitted the first part.

You are given a list of students at a local elementary school, together with their class assignment. The list is stored in a file `students.txt`. Each line of the file stores information about exactly one student. The format of each line is:

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

Here, `StLastName` and `StFirstName` identify the student, `Grade` specifies the student's grade, `Classroom` specifies the classroom in which the student studies, and `TLastName` and `TFirstName` identify the student's teacher. `Bus` is the school bus route that the student rides to school and `GPA` is the GPA of the student[1]. `Bus`, `Grade` and `Classroom` are integers (Kindergarten is 0; a value of 0 for `Bus` indicates that the student does not ride the bus to school), while all other fields are strings.

The `students.txt` file is available on the course Canvas page.

Here is a sample line from the file:

```
DROP, SHERMAN, 0, 104, 51, 2.65, NIBLER,  JERLENE
```

This line is to be read: "Sherman Drop, who takes bus route 51, is a kindergarten student assigned to the class of Jerlene Nibler in the classroom 104. Sherman has a GPA of 2.65."

Your goal is to write a program that searches the `students.txt` file and outputs the results of the search. You must implement the following search queries:

- Given a student's last name, find the student's grade, classroom and teacher (if there is more than one student with the same last name, find this information for all students);
- Given a student's last name, find the bus route the student takes (if there is more than one student with the same last name, find this information for all matching students);
- Given a teacher, find the list of students in his/her class;
- Given a bus route, find all students who take it;
- Find all students at a specified grade level.
- For a given grade level find the average GPA of students in that grade.
- In a given grade level find the student with the highest (lowest) GPA.

[1] Elementary schools these days don't do GPAs, but we will suspend disbelief for the purposes of this assignment.

Detailed Specifications

You are to write a program called `schoolsearch`, which implements the requested functionality. The program may be written in any programming language with which you are comfortable (Java, C, C++, Perl, Python, R, etc...). The program must satisfy the following requirements:

R1. The program shall run from the command line on a standard Linux or BSD machine. If any specific packages are required, please note these in your write-up.

R2. The program shall be invoked without command-line parameters. Upon start, the program shall provide the user with a visible prompt, read search instructions entered by the user, print the result, and wait for the next user instruction until the termination command is received.

R3. The following search commands shall be implemented.

- `S[tudent]: <lastname> [B[us]]`
- `T[eacher]: <lastname>`
- `B[us]: <number>`
- `G[rade]: <number> [H[igh]|L[ow]]`
- `A[verage]: <number>`
- `I[nfo]`
- `Q[uit]`

Notes: For simplicity, all instructions are *case sensitive*. In the specifications above, square brackets (`[]`) indicate optional input (e.g., the `Student` instruction can be abbreviated to `S`), while items in angle brackets (`<>`) denote values provided by the user.

R4. `S[tudent]: <lastname>`

When this instruction is issued, your program shall perform the following:

- search the contents of the `students.txt` file for the entry (or entries) for students with the given last name.
- For each entry found, print the last name, first name, grade and classroom assignment for each student found and the name of their teacher (last and first name).

R5. `S[tudent]: <lastname>` B[us]

When the `S[tudent]` instruction is issued with the `B[us]` option, your program shall perform the following:

- search the contents of the `students.txt` file for the entry (or entries) for students with the given last name.
- For each entry found, print the last name, first name and the bus route the student takes.

R6. `T[eacher]: <lastname>`

When this instruction is issued, your program shall perform the following:

- Search the contents of the `students.txt` file for the entries where the last name of the teacher matches the name provided in the instruction.
- For each entry found, print the last and the first name of the student.

R7. `G[rade]: <Number>`

When this instruction is issued your program shall perform the following actions:

- Search the contents of the `students.txt` file for the entries where the student's grade matches the number provided in the instruction.
- For each entry, output the name (last and first) of the student.

R8. `B[us]: <Number>`

When this instruction is issued your program shall perform the following actions:

- Search the contents of the `students.txt` file for the entries where the bus route number matches the number provided in the instruction.
- For each such entry, output the first and the last name of the student and their grade and classroom.

R9. `G[rade]: <Number> H[igh]` or
`G[rade]: <Number> L[ow]`

When this instruction is issued your program shall perform the following actions:

- Search the contents of the `students.txt` file for the entries where the student's grade matches the number provided in the instruction.
- If the `H[igh]` keyword is used in the command, find the entry in the `students.txt` file for the given grade with the *highest* GPA. Report the contents of this entry (name of the student, GPA, teacher, bus route).
- If the `L[ow]` keyword is used in the command, find the entry in the `students.txt` file for the given grade with the *lowest* GPA. Report the contents of this entry (name of the student, GPA, teacher, bus route).

R10. `A[verage]: <Number>`

When this instruction is issued your program shall perform the following actions:

- Search the contents of the `students.txt` file for the entries where the student's grade matches the number provided in the instruction.

- Compute the average GPA score for the entries found. Output the grade level (the number provided in command) and the average GPA score computed.

R11. `I[nfo]`

When this instruction is issued your program shall perform the following actions:

- For each grade (from 0 to 6) compute the total number of students in that grade.
- Report the number of students in each grade in the format

  `<Grade>: <Number of Students>`

  sorted in ascending order by grade.

R12. `Q[uit]`

When this instruction is issued your program shall quit the current session.

R13. The program shall assume that the file `students.txt` is located in the directory from which the program is run, and shall attempt to read the information from that file.

E1. Your program can have minimal error-checking. Exceptions or any other error-causing situations must be handled gracefully (without core dumps or runtime error messages), but the program does not need to attempt to recover from most situations. If the `students.txt` is not available, or has the wrong format - exit the program. If an unrecognized search instruction is provided, go back to the prompt.

Implementation Notes

Implementation details are left up to individual teams. Depending on the language you choose, different facilities may be available to you. Simple scripting languages make parsing easier and have some convenient data structures (e.g., associative arrays) which may be helpful. Compiled programming languages provide access to large libraries of powerful data structures and programs may work faster, but parsing is not as convenient.

The precise format you use for output is up to you. However, it is essential that your program uses *consistent* formatting throughout.

Testing and Deliverables

Data. The `students.txt` file is available on the course Canvas page.

Testing. Create a test script for your program. Cover all requirements and test correct behaviors of the program, as well as *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 suite).

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 (as a comment) specifying what it tests.

For example, here is a sample for three commands, each of which tests multiple requirements:

```
// CSC 365
// Lab 1-a test suite

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

S: HAVIR

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

S: NEMO

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

Q
```

Your program *does not need to have the capability to parse this test file* (ie. piped in from the command line or otherwise read in.) You may wish to add this feature for your convenience, but this **is not** a requirement.

Writeup. The write-up is a mandatory part of the lab. Your write-up shall contain the following:

- List of team members;
- Initial decisions: programming language, development environment;
- Notes on your chosen internal architecture: what data structures you used, 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)

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).

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. If you want to, you can submit a `Makefile` or any other supplemental files that help compile/run your code.
7. `students.txt`. Place `students.txt` file in the submission directory.

Submission. Place all files you wish to submit into a single directory. The directory name should include the CalPoly usernames of all team members, along with the string "lab-1a" (for example: `mustang1_mustang2_lab-1a`). Create an archive with the same name as your directory and submit via Canvas.

After submitting part a, the second part of the assignment will be available to you on Canvas.