CS 352 HW 2: Haskell

Due Friday, February 14, 10pm

Question 1:

Define a Haskell function, `computeGpas`, that takes one `String` argument containing a series of lines, each of which give the results of a student's grade in a particular section of a course. It should return a `String` that displays a list of students, listed in alphabetical order by last name (ties broken by first name).

The starter file for this assignment (and the only one you should need to modify) is `hw2.hs` (included in `cs352_hw2.zip`). It contains stub implementations of a set of functions that can be used to solve this problem; you should not change the signature for any of these stub functions.

Input Format

The input consists of a sequence of lines, where each line consists of seven space-separated words that give the grade for a student in a particular course. For example, the line below indicates:

```
PHL 220 A 3 Wai Choi B
```

- *Subject* (e.g. PHL): one or more uppercase letters
- *Course number* (e.g. 220): a non-negative integer
- *Section* (e.g. A): a single uppercase letter
- *Number of credits* (e.g. 3): a non-negative integer
- *First name* (e.g. Wai): a sequence of letters and `-` (hyphen) or `'` (apostrophe) characters; you may assume that no student has a first name longer than 15 characters.
- *Last name* (e.g. Choi): same formatting and length restrictions as first name
- *Grade* (e.g. B): Must be a valid grade; A/B/C/D/F, where A-D have an optional trailing `+` (plus) or `-` (minus). A is 4.0, B is 3.0, C is 2.0, D is 1.0, F is 0.0, `+` or `-` add or subtract 0.3, respectively.

You may assume that all input given to your program will be correctly formatted.

Output Format

The output should consist of a header line (“`NAME    GPA     #`”) and then a list of students containing their names (in the format last name, comma, first name), overall GPA (rounded to two decimal places), and number of credits attempted. The columns must line up with each other and the header line, with two spaces between each.

Building Starter Code

You will need the Glasgow Haskell Compiler (`ghc`) installed on your system (it is present on both the Windows and Linux Engineering images). The provided starter code in `hw2.hs` should compile and run using the following commands.

```
ghc --make run_hw2.hs -o hw2
./hw2
```

`hw2` takes two optional command-line arguments:

- If there are no command-line arguments, it reads from standard input and writes to standard output (Ctrl-D on Linux or Ctrl-Z on Windows will end standard input).
- If there is one command line argument, it reads from that file, and writes to standard output.
- If there are two command line arguments, it reads from the first file and writes to the second.

To ensure that your submitted code is compatible with the auto-grading scripts, I have also provided a test file, `test_hw2.hs`. This test file will print varying output depending on how much of your code is complete, but should always build and run without crashing. You can build it similarly to the homework code:

```
ghc --make test_hw2.hs -o test_hw2
./test_hw2
```

Example

Input in `black`, output in `blue`:

```
PHL 220 A 3 Wai Choi B
CS 273 A 1 Paddy O'Brien F
CS 273 A 1 Jean-Luc Picard B-
CS 203 B 3 Paddy O'Brien C-
CS 203 B 3 Jean-Luc Picard C
PHL 220 A 3 Ada Lovelace A
PHL 220 A 3 Grace Hopper C
THE 105 A 3 Grace Hopper B+
THE 105 A 3 Paddy O'Brien D+
                   NAME  GPA   #
              Choi, Wai  3.00  3
          Hopper, Grace  2.65  6
          Lovelace, Ada  4.00  3
         O'Brien, Paddy  1.29  7
       Picard, Jean-Luc  2.18  4
```

Instructions

Complete `hw2.hs`. You should <u>not</u> change the type definitions for `ClassRecord` or `StudentRecord`, any function’s type signature, the exported identifier list, or the definition of `main`; you may add helper functions, rewrite the argument lists of any of the other functions to pattern-match on their types, and replace the dummy implementations of the functions mentioned below.

`main` is provided for you to read input from standard input or a file, and write it standard output or a file. Example input:

```
"CS 203 B 3 Jean-Luc Picard C\nPHL 220 A 3 Ada Lovelace A\n
PHL 220 A 3 Grace Hopper C\nTHE 105 A 3 Grace Hopper B+"
```

[2] Write `splitLines`, which breaks the input string into a list of lines:

```
["CS 203 B 3 Jean-Luc Picard C",
 "PHL 220 A 3 Ada Lovelace A",
 "PHL 220 A 3 Grace Hopper C",
 "THE 105 A 3 Grace Hopper B+"]
```

[10] Write `parseRecord`, which converts a single line of input into a `ClassRecord` tuple.

- The course number and credits are parsed as `Int`.
- The first and last names are combined into a single `"Last, First"` string.
- The letter grade is converted into a `Double`-type numeric GPA.
- All other components stay as `String` type.

```
[("CS",203,"B",3,"Picard, Jean-Luc",2.0),
 ("PHL",220,"A",3,"Lovelace, Ada",4.0),
 ("PHL",220,"A",3,"Hopper, Grace",2.0),
 ("THE",105,"A",3,"Hopper, Grace",3.3)]
```

[10] Write `sortAndGroupRecords`, which sorts the list of class records by student name, then groups the sorted records by student name. The result should be a list of lists, where each inner list contains all the class records for a specific student, and the outer list is sorted by student name.

```
[[("PHL",220,"A",3,"Hopper, Grace",2.0),
  ("THE",105,"A",3,"Hopper, Grace",3.3)],
 [("PHL",220,"A",3,"Lovelace, Ada",4.0)],
 [("CS",203,"B",3,"Picard, Jean-Luc",2.0)]]
```

[10] Write `calcGpa`, which takes as input a list of student records (the inner lists from `sortAndGroupRecords`), and computes a single `StudentRecord` tuple with the name, GPA, and number of credits attempted for one student.

- GPA is a <u>weighted</u> average; a 4-credit course affects the average twice as much as a 2-credit course.
- You may assume each student attempts at least one credit.
- You may need to use the built-in `fromIntegral` function to convert `Int` to `Double`.

```
[("Hopper, Grace",2.65,6),
 ("Lovelace, Ada",4.0,3),
 ("Picard, Jean-Luc",2.0,3)]
```

[6] Write `formatRecord`, which turns a student record into a single line of output.

- All fields should be aligned to the same columns; name is right-aligned, GPA and number of credits are left-aligned
- There should be two spaces between columns
- Only two decimal places of GPA should be printed; `showFFloat` in the `Numeric` module may be useful.

```
["                Hopper, Grace  2.65  6",
 "                Lovelace, Ada  4.00  3",
 "             Picard, Jean-Luc  2.00  3"]
```

[10] Write `computeGpas` using the helper functions above such that the (unmodified) `main` function generates the correct output.

- Don't forget the header line for the output.

```
             NAME  GPA   #
    Hopper, Grace  2.65  6
    Lovelace, Ada  4.00  3
 Picard, Jean-Luc  2.00  3
```

I would advise completing and debugging one function at a time. Note that the `show` function can be used to convert intermediate results into strings for testing purposes in `computeGpas`.

Style

[6] Your Haskell code should not be significantly more complex than necessary, and should be structured in a way that facilitates understanding. Make use of whitespace and comments as appropriate to make your code clearly readable. Function and variable names should clearly represent their purpose or be sufficiently limited in scope to be clear from context.

Testing

[6] You should include two files with your submission, `input.txt` and `expect.txt`, which contain at least 6 lines of test input (in `input.txt`) and the output it should produce (in `expect.txt`).

Question 2:

[6] Select <u>two</u> of the options below and describe how you would modify your Haskell code to accomplish them. You do <u>not</u> need to modify your code, but your explanation should have sufficient detail that a classmate could accomplish the modification.

- Handle student names of arbitrary length so that the whitespace reflects the actual length of the longest name but the table columns still line up.

- Handle the “divide-by-zero” case where the GPA is undefined due to the student taking only zero-credit courses.
- Print an error message if a student was enrolled in the same course more than once.
- Print an error message if a course is listed with different numbers of credits for different students.
- Have your program not crash on input that doesn’t match the specification.

Submission Instructions

Your code and testing files for Question 1 should be put into a zip file; include all code that was provided in the `cs352_hw2.zip` starter file.

Your answer to Questions 2 should be put in a PDF file. Consult the instructor if you have trouble generating a PDF from your word processing software.

Both the zip and PDF files should be submitted on Moodle by the assignment due date.