Project Programming Paradigms 2021

Task Sets & Deadlines:

- The project will be divided in **4 steps**, each with its own **task set**, submission assignment and number of points associated. The task sets will have progressively-increasing difficulty, and will reflect the lecture progress.
- Each step will have a **fixed, hard, deadline**. (There will be no soft deadlines!) However, you **can skip deadlines** and still receive some points for task sets. The penalty for missing **each** deadline will be **-0.5p** out of the final project score. Just submitting something for a deadline is not enough to avoid the penalty. You have to pass at **least 50%** of the tests, else it is considered that you did not meet the deadline.

Project description

The project consists of a **data analytics app** that will be used to get insights into an (anonymised) dataset containing semester and exam grades from a real lecture. In this project, we will create a Haskell app that will give us a few valuable stats for the course.

Dataset

The dataset consists of course grades from a real lecture (the names have been changed). The course points were divided into: lecture grades, homework points and exam grades. The dataset has the following problems:

- lecture grades are mapped against email addresses, whereas homework grades and exam grades are mapped against names.
- lecture grades also contains entries with no email address. These correspond to students which have not provided an email in a valid form.
- so, in order to have a unique key for each student, the table email to name student map was created using a form. However, it contains some typos.

- Homework grades
- Lecture grades
- Exam grades
- Email to name map

Task Set 1 (Introduction)

These first tasks will get you accustomed to the dataset, by implementing a few simple operations.

A table **value** (or cell) will have type `String`. A **row** is a list of values, and a **table** is a list of **rows**. Please use the following type synonyms in your code, to make function signatures more legible.

```
type Value = String
type Row = [Value]
```

```
  type Table = [Row]
  --- now, whenever you use the word 'Table', Haskell knows you are referring to
[[String]].
```

Task 1 - 0.15p

We will first like to compute everyone's final exam grade, based on the formula:
(Q1+Q2+Q3+Q4+Q5+Q6)/4+Ex.scris

. You will have to write a function which takes an `Exam grades` table and returns a table with columns `Nume, Punctaj Exam`.

(Remark: your implementation should work on **any** dataset having the same structure, not only on the provided one.)

Task 2 - 0.2p

Based on their exam grade, check:

- how **many** students have **passed** the course (have at least 2.5p),
- what is the **percentage** of students who have **passed** and
- what is the **average grade** for the exam.

For this you will have to implement 3 functions which take an *Exam grades* table and return an `Int`.

Another important number for us is the number of students who received at least 1.5 points from homework (columns *T1, T2, T3* from *Homework grades* table).

Task 3 - 0.15p

Since each questions in the interview exam was focused on a chapter, we want to find out which are the trickiest chapters. We want to compute the average score for each question Q. The result will be a table with columns `Q1, Q2, Q3, Q4, Q5, Q6`.

Task 4 - 0.15p

Similar to the previous task, we want to know how many answers were marked 0/1/2 on each question. The result will be a table with columns *Q, 0, 1, 2* and 6 rows (besides the columns names) for each question.

Task 5 - 0.2p

In the end, we would like to see the overall leaderboard. You'll have to sort the students by their final exam grades (ascending). If two students have the same final score, they will be listed in alphabetical name order. The function for this will take an *Exam grades* table and return a table with columns *Nume, Punctaj Exam*.

Task 6 - 0.15p

In the end, let's check if the written part in an online exam is harder than the interview one. So we will compute the difference between the two parts. The final list will sort ascending the students by their difference. If two students have the same difference, they will be listed in alphabetical name order. The function takes an *Exam grades* table and returns a table with columns *Nume, Punctaj interviu, Punctaj scris, Diferenta*.

Submit Task Set 1

Deadline: 11.04 at 23:50.

Submit a .zip archive with your implementation. Don't include Main.hs or Dataset.hs (these will be overwritten anyway). Also, don't name any of your modules Utils.hs.

Vmchecker

Testing and Skel

Using the provided skel, you'll have to implement the specified tasks. Each task will have its designated function with an already specified type that will receive input from our Checker. **Do not modify anything** (function names and their types). For easier automated testing, our checker will call your functions and will expect the specified type as result.

In order to get the checker and the skel:

```
git clone https://github.com/cs-pub-ro/PP-CB.git
cd PP-CB/2020-2021/project
```

For more information, checkout README.

Recommendations

- Don't try and implement everything in one function! Use auxiliary functions.
- Use Float for computations.
- To convert from Int to String, use the function `show`.
- To convert from Float to String, use the function `printf “%.2f”`.
- To convert from String to a Numerical type, use the function `read`.

```
show 12 -- "12"
printf "%.2f" 1.23456789 -- "1.23" (please import Text.Printf)
read "12" :: Int -- 12
read "12.2" :: Float -- 12.2
```

Task Set 2

Prerequisite

Implement a function which reads a string in CSV-format to a Table and a function which writes a string from a Table.

Note: you have to implement this in order to test the other tasks.

```
read_csv :: CSV -> Table
write_csv :: Table -> CSV
```

Task 1 - 0.1p

Write a function which takes a column name and a Table and returns the values from that column as a list.

```
        as_list :: String -> Table -> [String]
```

Task 2 - 0.1p

Write a function which takes a column name and a Table and returns the Table sorted by that column (if multiple entries have the same values, then it is sorted by the first column). Sorting is ascending for columns with numeric values and lexicographical for strings. If the value is missing (“”), then it is considered before all values (e.g. “” < “0”).

```
        tsort :: String -> Table -> Table
```

Task 3 - 0.1p

Implement a function which applies a function to all values in a table.

```
        vmap :: (Value -> Value) -> Table -> Table
        --- An example use of this would be:
        correct_exam_table = value_map (\x -> if x == "" then "0" else x)
exam_grades
```

Task 4 - 0.1p

Implement a function which applies a function to all entries (rows) in a table. The new column names are given. Addionally, you have to implement a function which takes a Row from hw_grades Table and returns a Row with 2 values: name, total points (sum of columns 2 - 8).

```
        rmap :: (Row -> Row) -> [String] -> Table -> Table
        get_hw_grade_total :: Row -> Row
```

Task 5 - 0.1p

Implement a function which takes Tables t1 and t2 and adds all rows from t2 at the end of t1, if column names coincide. If columns names are not the same, t1 remains unchanged.

```
        vunion :: Table -> Table -> Table
```

Task 6 - 0.1p

Implement a function which adds new columns to a table (simple horizontal union of 2 tables).

```
        hunion :: Table -> Table -> Table
```

Task 7 - 0.2p

Implement table join with respect to a key (a column name). If an entry from table 1 has the same value for the key as an entry from table 2, their contents must be merged. If there are other columns with the same name in both tables, then the value from t2 overrides the value from t1, unless it is null (“”).

```
        tjoin :: String -> Table -> Table -> Table
```

Task 8 - 0.1p

Write the implementation for the cartesian product of 2 tables. The new column names are given. To generate the entries in the resulting table you have to apply the given operation on each entry in t1 with each entry in t2.

```
cartesian :: (Row -> Row -> Row) -> [String] -> Table -> Table -> Table
```

Task 9 - 0.1p

Extract from the table only the specified columns.

```
projection :: [String] -> Table -> Table
```

Checker

The checker has been updated, to include the new tasks: Checker.

You can use git pull to get the latest updates. **I recommend making a backup of your file Tasks.hs, in case git pull overrides it (though it shouldn't).**

Submit

Deadline: 25.04, 23:50. **Vmchecker**: https://v2.vmchecker.grid.pub.ro/homepage/?course=5.