プログラミング言語論 課題2 レポート

情報工学科 4年 7番 市古 空

0. はじめに

- 変数 table の型定義と一部コメントを抜粋。この先、変数 table を単に table と表記する。

```
table :: [(String, Int, String, String, Double, Double, String,
String, String, Int, Int, String)]
{-
  minority: factor. Does the instructor belong to a minority (non-
Caucasian)?
  age: the professor's age.
  gender: factor indicating instructor's gender.
  credits: factor. Is the course a single-credit elective (e.g., yoga,
aerobics, dance)?
  beauty: rating of the instructor's physical appearance by a panel of
six students, averaged across the six panelists, shifted to have a
mean of zero.
  eval: course overall teaching evaluation score, on a scale of 1
(very unsatisfactory) to 5 (excellent).
  division: factor. Is the course an upper or lower division course?
(Lower division courses are mainly large freshman and sophomore
courses)?
  native: factor. Is the instructor a native English speaker?
  tenure: factor. Is the instructor on tenure track?
  students: number of students that participated in the evaluation.
  allstudents: number of students enrolled in the course.
  prof: factor indicating instructor identifier.
-}
```

- 依存関係は下記のとおりである。

```
$ ghci --version
The Glorious Glasgow Haskell Compilation System, version 8.10.7
```

1. table.hsのうち，<該当関数>がどのような関数であると定義されているか，説明してください.

※ <該当関数> = get_first，get_the_lowest，get_the_highest，filter_by_not_minority_and_average

1.1 get_first

table の先頭要素を返す関数。
リストの先頭要素の取得には `GHC.List.head` 関数を利用している。

```
-- 実装
*Main> get_first = head table
-- 実行結果
*Main> get_first
("yes",36,"female","more",0.2899157,4.3,"upper","yes","yes",24,43,"1")
-- get_first の型情報
*Main> :type get_first
get_first
  :: (String, Int, String, String, Double, Double, String, String,
      String, Int, Int, String)
```

1.2 get_the_lowest

table の要素であるタプルの6番目要素（eval) が最小である要素を返す関数。

内部では「昇順ソート → 先頭要素の抽出」を行うことで機能を実現している。 リストのソートは `Data.List.sortBy`関数を利用しており、先頭要素の取得は `get_first` と同じである。
Haskell の構文として `$` による関数の右結合が行われているのも特徴的である。またここで使用されている `sortBy`, `comparing` は高階関数と呼ばれるものである。高階関数については 1.4 で詳しく説明する。

参考

```
-- 実装
*Main> get_the_lowest = head $ sortBy (comparing eval) table
-- 実行結果
*Main> get_the_lowest
("no",59,"male","more",-0.7377322,2.1,"upper","yes","yes",39,44,"2")
-- get_the_lowest の型情報
*Main> :type get_the_lowest
get_the_lowest
  :: (String, Int, String, String, Double, Double, String, String,
      String, Int, Int, String)
```

1.3 get_the_highest

table の要素であるタプルの6番目要素（eval) が最大である要素を返す関数。

前節の `get_the_lowest` の逆である。
内部では「降順ソート → 先頭要素の抽出」を行うことで機能を実現している。 リストのソート、先頭抽出は `get_the_lowest` と同じだが、降順ソートを実現するために `GHC.Base.flip` 関数を用いている。

参考

```
-- 実装
*Main> get_the_highest = head $ sortBy (flip $ comparing eval) table
-- 実行結果
*Main> get_the_highest
("no",47,"male","more",0.540917,5.0,"lower","yes","no",10,11,"10")
-- get_the_highest の型情報
*Main> :type get_the_highest
get_the_highest
  :: (String, Int, String, String, Double, Double, String, String,
      String, Int, Int, String)
```

1.4 filter_by_not_minority_and_average

table の要素であるタプルの1番目要素（minority）が `"no"` であるリスト要素の eval の平均値を返す関数。

内部では「minority の値でフィルター → eval の値で List を作成 → 作成した List の平均値を算出」することで機能を実現している。
List 要素のフィルターに `Data.List.filter` 関数を用い、新たな List の作成に `Data.List.map` 関数を、List の平均値計算に独自実装である `average` 関数を用いている。下記は `avarage` 関数の実装部分である。

```
-- 平均を求める関数
average :: (Real a) => [a] -> Double
average xs = (realToFrac $ sum xs) / fromIntegral (length xs)
```

また `filter` 関数の第一引数を見てみるとリスト要素を受け取り真偽値を返すラムダ式が記述されていることがわかる。これは Haskell のラムダ式は無名関数として扱えるためである。 このように引数や戻り値に関数自体を持つ関数を「高階関数」という。今回の table.hs だと `filter` 以外にも `map` や `sortBy`, `comparing` などの関数が高階関数である。

参考

```
-- 実装
*Main> filter_by_not_minority_and_average = average $ map eval $ filter
(\x -> minority x == "no") table
-- 実行結果
*Main> filter_by_not_minority_and_average
4.015288220551379
-- filter_by_not_minority_and_average の型情報
*Main> :type filter_by_not_minority_and_average
filter_by_not_minority_and_average :: Double
```

2. filter_by_not_minority_and_average関数を参考に，your_original_1, your_original_2関数を実装して，それらがどのような結果であるか説明してください.

2.1 your_original_1 について

```
-- 実装
*Main> your_original_1 = average $ map age $ filter (\x -> minority x ==
"no") table
-- 実行結果
*Main> your_original_1
48.769423558897245
-- your_original_1 の型情報
*Main> :type your_original_1
your_original_1 :: Double
```

table の要素であるタプルの1番目要素（minority）が `"no"` であるリスト要素の **age の平均値を返す**関数。

前章の filter_by_not_minority_and_average 関数を一部改変して実装した。具体的には `map` 関数の引数に渡すタプル要素抽出の関数を `eval` から `age` に置き換えた。これによって `average` 関数で age 値の List を渡すことができるため、「age の平均値を返す」機能が実現できる。

2.2 your_original_2 について

```
-- 実装
*Main> your_original_2 = average $ map eval $ filter (\x -> gender x ==
"male") table
-- 実行結果
*Main> your_original_2
4.069029850746268
-- your_original_2 の型情報
*Main> :type your_original_2
your_original_2 :: Double
```

table の要素であるタプルの**3番目要素（gender）が `"male"` である**リスト要素の eval の平均値を返す関数。

前章の filter_by_not_minority_and_average 関数を一部改変して実装した。具体的には `filter` 関数の引数にある真偽値を返す関数を更新した。

```
-- filter_by_not_minority_and_average
(\x -> minority x == "no")
-- your_original_2
(\x -> gender x == "male")
```

minority 関数を使っている箇所が gender 関数になり、期待値の設定も変わっていることがわかる。なお更新後も既存実装にならいラムダ式を用いて実装している。

この更新により「3番目要素（gender）が `"male"` であるリスト要素」という機能が実現できる。

2.3 your_original_1, 2 の実行結果を比較する

それぞれの関数の実行結果

```
*Main> your_original_1
48.769423558897245
*Main> your_original_2
4.069029850746268
```

具体的な実装や戻り値は異なるが、関数の持つ責務自体は同じである

それぞれの関数の実行結果をみると異なる値が返ってくる。 しかし詳細な実装や関数の型情報を見るとどちらも引数を取らず、「List を filter → ある種類の値で平均値を算出」という責務を持っていることがわかる。

このことから「具体的な実装や戻り値は異なるが、関数の持つ責務自体は同じ」であり、今回私が変更した map や filter の引数部分は your_original_1, 2 の引数として切り出すことで2つの関数の責務の重複を省くことができると考察できる。