universität innsbruck

Functional Programming	WS 2019/2020	LVA 703024+703025
Exercises 9		Deadline: December 11, 2019, 6am

- Please write all your Haskell functions from this exercise sheet into a single .hs-file and upload it in OLAT.
- You can use a template .hs-file that is provided on the proseminar page.
- The file should compile with ghci.
- Once the file has been uploaded, it cannot be changed or resubmitted!

Exercise 9.1 *Lists* **6 p.**

For this exercise you mainly use slides part 4 and part 5 up to slide 24 (e.g. `filter`, `map`, `reverse`, `sum`). You are **not** allowed to use list comprehensions or `if then else` statements. Only in exercise 9.1.1 list recursion is required. Try to write your functions as compactly as possible with suitable applications of higher-order functions in combination with function composition (`.`), the application operator (`$`) and λ-abstraction (`\ vars -> exp`). Add type signatures to each function that are as general as possible (e.g. in exercises 9.1.4 - 9.1.6 the duration can be of any numeric type).

You are given a dataset of website visits. Here is an example of how such a dataset might look:

```
webData :: [String]
webData = ["Youtube", "Google", "Facebook", "Youtube", "Facebook",
           "Youtube", "Facebook", "Google", "Youtube"]
```

Each entry in the list represents a visit to a website. The goal is to create a ranking based on the number of times a website has been visited. For example, Google has been visited two times.

1. First you have to determine which websites occur in the dataset. Write a function `uniqueWebsites` that takes a dataset of the same form as `webData` and returns a list with the websites that occur in the dataset. For the example dataset a result could be:

   ```
   uniqueWebsites webData = ["Google", "Facebook", "Youtube"]
   ```

   (1 point)

2. The next step is to write a function `count` that takes the name of a website and a dataset and returns how often that website occurs in the dataset. For the example dataset and `website = "Google"` the result is: `count "Google" webData = 2`. (1 point)

3. The functions `uniqueWebsites` and `count` can now be combined to create a function that gives us a ranking of website visits. Write a function `result` that takes the dataset `webData` and returns an *ordered* list of tuples `[(count, website)]`, where the first tuple in the list is the website with the highest count and the last tuple the website with the lowest count (If two websites have equal count the ordering does not matter). For the example dataset the result is:

   ```
   result webData = [(4,"Youtube"),(3,"Facebook"),(2,"Google")]
   ```

   Hint: have a look at the functions `sortBy` and `compare`. (1 point)

Now you are given a more detailed dataset that gives besides each website visit also the duration of that visit:

```
webDataDuration :: [(String, Double)]
webDataDuration = [("Youtube", 2.5), ("Google", 23.2), ("Facebook", 23.2),
                   ("Youtube", 3.4), ("Facebook", 4.5), ("Youtube", 34.5),
                   ("Facebook", 33.2), ("Google", 34.3), ("Youtube", 12.4)]
```

Each entry in the list is a tuple representing a visit to a website and the duration of that visit in seconds. This time you have to rank the website based on the total duration of time a website has been visited. For example, Google has been visited for 57.5 seconds.

4. Write a new function `uniqueWebsites2`, similar to `uniqueWebsites`, for a dataset of the same form as `webDataDuration` that returns a list of websites that occur in the dataset. (1 point)

5. Write a new function `count2`, similar to `count`, which takes a `website` and a dataset of the same form as `webDataDuration` as inputs and returns the total time `website` has been visited. (1 point)

6. Write a new function `result2` that is a higher-order function version of the original `result` function. `result2` can be used for both datasets and rank each of them accordingly. The function `result2` will take three inputs, a count function, a getWebsites function and a dataset. For the example datasets the results are:

```
result2 count uniqueWebsites webData =
  [(4,"Youtube"),(3,"Facebook"),(2,"Google")]
result2 count2 uniqueWebsites2 webDataDuration =
  [(60.9,"Facebook"),(57.5,"Google"),(52.8,"Youtube")]
```

(1 point)

Exercise 9.2 *foldr-function* **2 p.**

Write the following functions using `foldr`. You may find lambda expressions useful.

1. Consider the prelude function `all` which defined in the slide Part 5, page 20. Define `all` without recursion, but via `foldr` as `all_fold`. (1 point)

2. Consider a function which converts a list of digits (represented as `Integer`s) into an `Integer`:

```
dig2int :: [Integer] -> Integer
dig2int [] = 0
dig2int (x:xs) = x + 10 * dig2int xs
```

   where the output of `dig2int [2, 1, 5]` is `512`. Define `dig2int` without recursion, but via `foldr` as `dig2int_fold`. (1 point)

Exercise 9.3 *Zip, List Comprehensions* **2 p.**

1. Write a function `number :: [a] -> [(Int,a)]` which numbers all elements of a list with their position. For instance, `number "hello" = [(0,'h'),(1,'e'),(2,'l'),(3,'l'),(4,'o')]`.

   Define `number` with the help of `zip` or `zipWith`. It is neither allowed to use recursion nor is the `(!!)` operator allowed. (1 point)

2. Write a function `evenProdSum` which given a list of `Int`s $[x_1, \ldots, x_n]$, computes the sum $2x_2 + 4x_4 + 6x_6 + \ldots + nx_n$ (if n is even), or $2x_2 + 4x_4 + 6x_6 + \ldots + (n-1)x_{n-1}$ (if n is odd).

   For instance, `evenProdSum [3,17,8,9,5,7] = 2 * 17 + 4 * 9 + 6 * 7 = 112`.

   Define `evenProdSum` without using recursion. Instead, list comprehensions, and functions like `number` from part 1 of this exercise and the Prelude function `sum` might be useful. (1 point)