Assignment 4

Transitioning to Haskell and SML

Deadline: Monday, November 30, 23:55

4.1 Submission instructions

1. Unzip the `A4.zip` folder. You should find 2 folders, each with one file inside:
   - `hs`, containing `Solutions.hs` - for the Haskell exercises
   - `sml`, containing `solutions.sml` - for the SML exercises
2. Edit the first line of each of the source files as described in the comments.
3. Edit the source files with your solutions.
4. When done, zip (not rar renamed as zip!) this `A4` folder and name the zip archive with the following format:

$$A4\_\langle FirstName\rangle\_\langle LastName\rangle\_\langle Group\rangle$$

   Examples of valid names:
   - `A4_John_Doe_30432.zip`
   - `A4_Ion_Popescu_30434.zip`
   - `A4_Gigel-Dorel_Petrescu_30431.zip`

   Examples of invalid names:
   - `Solutions.zip`
   - `A4.zip`
   - `Solutii_A4_Ion_Popescu.zip`

4.2 Assignment exercises

4.2.1 Haskell

Exercise 4.2.1 2p

Implement a function `strWords` that splits a string into a list of words. You can assume that the string contains only letters and spaces.

Haskell REPL

```
> strWords "Whats up doc"
["Whats", "up", "doc"]
> strWords "Beware the Jabberwock my son"
["Beware", "the", "Jabberwock", "my", "son"]
```

Implementation restrictions:
Aside from the functions that you define, you may only use the following library functions in your implementation: `take`, `drop`, `takeWhile`, `dropWhile`, `span`, `foldl`, `foldr`.

Grading:
2 points for the correct implementation

Exercise 4.2.2 2p

Implement a function `piglatinize` that converts strings to pig latin. The first consonant of each word is moved to the end of the word and "ay" is added, so "first" becomes "irst-fay." Words that start with a vowel ('a', 'e', 'i', 'o' or 'u') have "hay" added to the end instead ("apple" becomes "apple-hay"). You can assume that the input string contains only letters and spaces.

Haskell REPL

```
> piglatinize "Whats up doc"
"hats-Way up-hay oc-day"
> piglatinize "An apple a day keeps the doctor away"
"An-hay apple-hay a-hay ay-day eeps-kay he-tay octor-day away-hay"
```

Hint:
Use your `strWords` words function to split the string.

Implementation restrictions:
Aside from the functions that you define, you may only use the following library functions in your implementation: `elem`, `notElem`, `intersperse`, `intercalate`, `filter`, `map`

Grading:

- 1 point for defining a function that transforms a given word to pig latin.
- 1 point for defining a function that concatenates the transformed words (by placing spaces between them) into a string.

Exercise 4.2.3 — 2p

Write a function `apprPi :: Double` will be used to approximate $-\pi$ (minus pi) using the the `iter` function.

1. Implement the `nextPi :: Double -> Double` function that will be used by `iter` to generate the next approximation of pi, based on the following formula:

$$x_{n+1} = x_n + \frac{2 \cdot \cos(\frac{x_n}{2})}{2 \cdot \sin(\frac{x_n}{2}) - 1}, x_0 = 0$$

2. Implement the `apprPi` function that uses `iter` and `nextPi` to approximate $-\pi$ until two successive elements are equal.

Implementation restrictions:
Aside from the functions that you define, you may only use the following library functions in your implementation: `takeWhile`, `dropWhile`, `last`, `head`, `snd`, `fst`

Grading:

- 1 point for implementing the `nextPi` function
- 1 point for returning the correct result

Exercise 4.2.4 — 7p

Define a function `topWords n str`, with the signature `topWords :: Int -> String -> [(String, Int)]` that returns the top `n` words from the string `str`, by the number of occurrences. If there are multiple words with the same number of occurrences, you should break the ties sorting the words in lexicographic order. You can assume that the input string contains only letters and spaces.

Haskell REPL

```
> topWords 10 "I know that you know that I know"
[("know", 3), ("I", 2), ("that", 2), ("you", 1)]
> topWords 3 "I know that you know that I know"
[("know", 3), ("I", 2), ("that", 2)]
> topWords 6 "An apple a day keeps the doctor away"
[("An", 1), ("a", 1), ("apple", 1), ("away", 1), ("day", 1), ("doctor", 1)]
```

Solution option 1:
One possible solution is to define a function `update fn def k l` with the signature `udpate :: (Eq k) => (v -> v) -> v -> k -> [(k, v)] -> [(k, v)]` that looks up the key `k` in the association list `l`. If the key is found, then the update function `fn` is applied to the value associated to the key and the list with the update value is returned. If the key is not found, the key is inserted into the association list with the default value `def`.

Haskell REPL

```
> update (+1) 1 "a" [("a", 2), ("b", 3), ("c", 1)]
[("a", 3), ("b", 3), ("c", 1)]
> update (+1) 1 "d" [("a", 2), ("b", 3), ("c", 1)]
[("a", 3), ("b", 3), ("c", 1), ("d", 1)]
> update (+1) 1 "c" [("a", 2), ("b", 3), ("c", 1)]
[("a", 3), ("b", 3), ("c", 2)]
```

Solution option 2:

1. Implement a function `uniques :: (Eq a) => [a] -> [a]` that obtains the unique elements from a list.[a]

```
Haskell REPL

> uniques [1, 2, 1, 4, 3, 2]
[1, 2, 3, 4]
```

2. Implement a function `countOccurrences :: (Eq a) => a -> [a] -> Int` that counts how many times a given element occurs in a list.

```
Haskell REPL

> countOccurrences 1 [1, 2, 1, 4, 3, 2]
2
```

3. Implement a function `countWords :: String -> [(String, Int)]` that uses `uniques` and `countOccurencres` obtain a list of (word, count) tuples. **This list does not have to be sorted (yet)!**

Implementation restrictions:
Aside from the functions that you define, you may only use the following library functions in your implementation: `map`, `filter`, `head`, `tail`, `take`, `drop`, `takeWhile`, `dropWhile`, `foldl`, `foldr`, `span`, `sortOn`, `sortBy`

Grading:

- 5 points for obtaining the unsorted list of (word, count) tuples.
  - If you choose option 1 (the `update` function):
    - 2 points for handling the case when the value is missing from the association list
    - 3 points for handling the case when the value is already present in the association list
  - If you choose option 2 (`uniques`, `countOccurencres` and `countWords`):
    - 2 points for the `uniques` function
    - 2 points for the `countOccurrences` function
    - 1 point for the `countWords` function
- 2 points for obtaining the top n words
  - 1 point for sorting the list by word count[b]
  - 1 point for sorting the ties lexicographically

[a] You might want to take a look at the `quicksort` function. Specifically what happens if you change the comparison operators in the partition part from `(<=)` to `(<)` and `(>=)` to `(>)`.

[b] i.e. you still get one point if the output is only sorted by the number of occurrences

4.2.2 SML

Exercise 4.2.5 2p

Implement in SML a function `levenshtein: string -> string -> int` to calculate the Levenshtein distance of 2 string, in terms of the following operations: insertions, deletions and substitutions.

SML REPL

```
- levenshtein "kitten" "sitting";
val it = 3 : int;
- levenshtein "haskell" "sml";
val it = 5 : int;
```