universität innsbruck

Functional Programming | WS 2019/2020 | LVA 703024+703025

Exercises 10 | Deadline: January 08, 2020, 6am

- Please write all your Haskell functions from this exercise sheet into a single .hs-file for Exercise 10.1, and two files Year.hs and Picture.hs for Exercise 10.2, and upload them to OLAT.
- For Exercise 10.1 you can use a template .hs-file.
- For Exercise 10.2 there are two files available: `Calendar.hs` contains the Haskell-source from the lecture that need to be decomposed and modified; and `MCalendar.hs` is the testing file that should not be altered.
- All files are available from the proseminar homepage.
- All submitted files should compile with ghci.
- Once the files have been uploaded, they cannot be changed or resubmitted!

Exercise 10.1 *Lists* 5 p.

In this exercise the goal is to provide more insight about `foldl` and `foldr`. Consider the list `xs` of integers `[1,2,3,4]` without syntactic sugar `1 : (2 : (3 : (4 : [])))`.

1. Draw the abstract syntax tree of the latter representation of the list above and explain the difference between `foldl` and `foldr`.

   Substitute `+` for the `Cons` operator `:` and `0` for `[]` in the above list representation.
   Does this represent the calculation done with `foldl (+) 0 xs` or `foldr (+) 0 xs`?
   What happens if we take `(*)` instead of `(+)`? Justify your answers. (1 point)

2. The powerlist of a list `ys` contains every list, where arbitrary elements were removed from `ys`. For this exercise the only allowed functions are `foldl`, `foldr`, `(++)` and `map`. Give their most general type signatures.

   (a) Implement a function `powerlistFR` using `foldr`. (1 point)

   (b) Implement a function `powerlistFL` using `foldl`. (1 point)

   Examples:
   `powerlistFL [1,1,2] = [[1,1,2],[1,1],[1,2],[1],[1,2],[1],[2],[]]`
   `powerlistFR [1,2,3] = [[1,2,3],[1,2],[1,3],[1],[2,3],[2],[3],[]]`

   Note that the order of the lists in these examples is not relevant, i.e., your solution may return the elements of the powerlist in a different order than in the examples.

3. Insertion sort[1] is a sorting algorithm, which can be used to sort a list. It works by iteratively inserting elements at the correct position into a sorted list. Implement insertion sort using `foldr`. Give the most general type signature. Hint: You may find `takeWhile`, `dropWhile` and `span` useful.

   Example: `sortFold [3,1,2,1,6,12] = [1,1,2,3,6,12]` (2 points)

[1] https://en.wikipedia.org/wiki/Insertion_sort

Exercise 10.2 *Modules* 5 p.

In this exercise we will use modules to decompose the calendar program discussed in the lecture. The goal is to decompose `Calendar.hs` into 3 modules `Year`, `Picture`, and `MCalendar`, such that `Year` comprises *only* and *all* information about years, months, days, and calculations with them, and `Picture` comprises *only* and *all* information about pictures, rows, stacks and operations on them. `MCalendar` is given below and uses both: evaluating `monthInfo2` (from the module `Year`) on a given month and year, yields information (a quadruple), which is then turned into a picture by `info2Pic` (from the module `Picture`).
In as far as they are specified below, these 3 modules may *not* be modified.

1. Classify each type and function definition in `Calendar.hs`: say whether it should belong to the `Year` module, the `Picture` module, or whether it not does not clearly belong to one of them.

   Example: the `tile` function definition should belong to `Picture` as it transforms a list of pictures into a picture (no information on years, months), but `month` does not clearly belong to one of them (it involves months, years and pictures). (1 point)

2. Write a module `Year` of shape:

   ```
   module Year(Month,Year,monthInfo2) where

   ...

   monthInfo2 :: Month -> Year -> (Int, Int, String, Int)
   monthInfo2 m y = (fstdays y !! (m - 1), mlengths y !! (m - 1),header,7) where
     header = " Mo Tu We Th Fr Sa Su"
   ```

   The module should be supplemented with types and function definitions from `Calendar.hs` belonging to it (as in the first item), such that `monthInfo2` outputs a quadruple comprising the offset of the first day, the number of days in a month, the header containing the names of the days in the week, and the number of days in a week.

   Examples: `monthInfo2 11 2019` = `(4,30," Mo Tu We Th Fr Sa Su",7)` and
   `monthInfo2 2 2021` = `(0,28," Mo Tu We Th Fr Sa Su",7)`

   (2 points)

3. Write a module `Picture` of shape:

   ```
   module Picture(Picture,info2Pic,showPic) where

   ...

   info2Pic :: (Int,Int,String,Int) -> Picture
   info2Pic (offset,n,h,s) = ...
   ```

   The module should be supplemented with types and function definitions from Calendar.hs belonging to it (as in the first item), and a definition of `info2Pic` such that it generates a `Picture` of width 3·`s` from a quadruple `(offset,n,h,s)`. More precisely:

   - the string `h` is the first row of the Picture (so `h` should have length exactly 3·`s`);
   - the number of rows is such that all numbers 1 to `n` are in the picture and there are *no* trailing empty rows (this is different from the original program, where empty rows may be generated);
   - subsequent rows contain the numbers 1 to `n` in order (each number contributes a string of length 3), but in the beginning an `offset` number of entries are left blank (are filled with 3 spaces each).

   Examples: `info2Pic (2,10,"abcdefghi",3)` should yield:

   ```
   (5,9,["abcdefghi","        1","  2  3  4","  5  6  7","  8  9 10"])
   ```

   that is, 5 rows, where the first is the header `"abcdefghi"` and the others enumerate the numbers 1 to 10 with the first 2 entries left blank, and `info2Pic (4,7," Mo Tu We Th Fr Sa Su",7)` should yield:

   ```
   (3,21,[" Mo Tu We Th Fr Sa Su","              1  2  3","  4  5  6  7         "])
   ```

   (2 points)

You may test the combination of both modules by the module:

```haskell
module MCalendar where

import Picture
import Year

month :: Month -> Year -> Picture
month m y = info2Pic $ monthInfo2 m y

showMonth :: Month -> Year -> String
showMonth m y = showPic $ month m y
```

In particular, evaluating `showMonth` for February 2021 should yield a `Picture` having exactly 5 rows (a header and 4 weeks). That is, evaluating `putStr $ showMonth 2 2021` should yield:[2]

```
 Mo Tu We Th Fr Sa Su
  1  2  3  4  5  6  7
  8  9 10 11 12 13 14
 15 16 17 18 19 20 21
 22 23 24 25 26 27 28
```

[2] Here `putStr` is used to print formatted output, i.e. such that newline-symbols are printed as new lines.