universität innsbruck

Functional Programming	WS 2020	LVA 703025
Exercise Sheet 11, 10 points		Deadline: Wednesday, January 20, 2020, 6am

- Please write all the Haskell code into a single .hs-file and upload it in OLAT.
- You can use the template .hs-file that is provided on the proseminar page.
- Your .hs-file should be compilable with ghci.
- Don't forget to mark your completed exercises in OLAT.

Exercise 11.1 *List Comprehensions* **4 p.**

1. Write a function `addPos :: [a] -> [(Int,a)]` which returns all elements of a list with their positions (counting from 1 onwards). For instance, `addPos "examp" = [(1,'e'),(2,'x'),(3,'a'),(4,'m'),(5,'p')]`.

   Define `AddPos` using `zip` or `zipWith`. Neither recursion nor the `(!!)` operator is allowed. (1 point)

2. Write a function `prodSumEven` which given a list of `Int`s $[x_1, \ldots, x_n]$, computes the sum $1x_2 + 3x_4 + 5x_6 + \ldots + (n-1)x_n$ (if n is even), or $1x_2 + 3x_4 + 5x_6 + \ldots + (n-2)x_{n-1}$ (if n is odd).

   For instance, `prodSumEven [3,17,8,9,5,7] = 1 * 17 + 3 * 9 + 5 * 7 = 79`.

   Define `prodSumEven` without using recursion. Instead, list comprehensions, and functions like `addPos` from part 1 of this exercise and the Prelude function `sum` might be useful. (2 points)

3. Generalize the above function to `prodSum :: (Int -> Bool) -> [Int] -> Int` which computes the sum as in part 2 with positions selected by a argument function (`Int -> Bool`),

   For instance, `prodSumEven [3,17,8,9,5,7] = prodSum even [3,17,8,9,5,7]`

   or `prodSum (\x -> x `mod` 3 == 0) [1, 3, 4, 6, 10, 11] = 4 * (3 - 1) + 11 * (6 - 1) = 63`.

   (1 point)

Exercise 11.2 *foldr and foldl* **3 p.**

In this exercise the goal is to provide more insight about `foldl` and `foldr`.

1. Consider the list `xs` of integers `[2,2,2,2]` without syntactic sugar `2 : (2 : (2 : (2 : [])))`.
   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. Re-implement the Prelude function `filter` twice, once using `foldr` and once using `foldl`. The function `filterr f xs` has to be implemented using `foldr` and the function `filterl f xs` using `foldl`. (1 point)

   Examples:

```
filterr even [1,2,3,4] == [2,4]
filterl odd [1,2,3,4] == [1,3]
filterr f xs == filterl f xs
```

3. Use `foldr` or `foldl` to write a function `findFirst f xs` that returns the first element in `xs` for which `f :: a -> Bool` returns `True`. Use `Maybe` as return type, i.e., if a match is found return `Just x` and otherwise `Nothing`.

   Write a second function `findLast f xs` that returns the last element in `xs` for which `f :: a -> Bool` returns `True`. Again you have to use `foldr` or `foldl`. (1 point)

   Examples:

```
findFirst even [1,2,3,4] == Just 2
findFirst even [1,3] == Nothing
findLast even [1,2,3,4] == Just 4
findLast even [1,3] == Nothing
```

Exercise 11.3 *Calendars* 3 p.

In the lecture part 4, you have seen the implementation of showing `month` in `Calendar.hs`. In this Exercise you will expand the program so that it can display the calendar of a whole year. You can use the `Calendar_template.hs` file for this exercise.

1. Implement a function `monthWithName :: Month -> Year -> Picture` which print the month as follows:

```
       October

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 29
30 31
```

   Similar to the `Calendar.hs`, you can inspect your output using the function

```
showMonthWithName :: Month -> Year -> String
showMonthWithName m y = showPic $ monthWithName m y
```

   display final string via `putStr :: String -> IO ()` to properly print newlines and drop double quotes. (1 point)

2. Write a function `year :: Year -> Picture` that returns the calendar of an entire year. Use the function `monthWithName` from the previous exercise and the functions defined in `Calendar.hs`. Make sure your output includes proper padding between months, and ensures that all months are properly aligned. You can inspect your result using the provided function

```
showYear :: Year -> String
showYear y = showPic $ year y
```

   For example:

```
*Main> putStr $ showYear 1995
                              1995

       January                February                March

Mo Tu We Th Fr Sa Su   Mo Tu We Th Fr Sa Su   Mo Tu We Th Fr Sa Su
                   1            1  2  3  4  5            1  2  3  4  5
 2  3  4  5  6  7  8    6  7  8  9 10 11 12    6  7  8  9 10 11 12
 9 10 11 12 13 14 15   13 14 15 16 17 18 19   13 14 15 16 17 18 19
16 17 18 19 20 21 22   20 21 22 23 24 25 26   20 21 22 23 24 25 26
23 24 25 26 27 28 29   27 28                 27 28 29 30 31
30 31
```

```
       April                May                  June

Mo Tu We Th Fr Sa Su   Mo Tu We Th Fr Sa Su   Mo Tu We Th Fr Sa Su
                1  2    1  2  3  4  5  6  7             1  2  3  4
 3  4  5  6  7  8  9    8  9 10 11 12 13 14    5  6  7  8  9 10 11
10 11 12 13 14 15 16   15 16 17 18 19 20 21   12 13 14 15 16 17 18
17 18 19 20 21 22 23   22 23 24 25 26 27 28   19 20 21 22 23 24 25
24 25 26 27 28 29 30   29 30 31               26 27 28 29 30

       July                 August              September

Mo Tu We Th Fr Sa Su   Mo Tu We Th Fr Sa Su   Mo Tu We Th Fr Sa Su
                1  2       1  2  3  4  5  6                1  2  3
 3  4  5  6  7  8  9    7  8  9 10 11 12 13    4  5  6  7  8  9 10
10 11 12 13 14 15 16   14 15 16 17 18 19 20   11 12 13 14 15 16 17
17 18 19 20 21 22 23   21 22 23 24 25 26 27   18 19 20 21 22 23 24
24 25 26 27 28 29 30   28 29 30 31            25 26 27 28 29 30
31

      October              November             December

Mo Tu We Th Fr Sa Su   Mo Tu We Th Fr Sa Su   Mo Tu We Th Fr Sa Su
                   1          1  2  3  4  5                1  2  3
 2  3  4  5  6  7  8    6  7  8  9 10 11 12    4  5  6  7  8  9 10
 9 10 11 12 13 14 15   13 14 15 16 17 18 19   11 12 13 14 15 16 17
16 17 18 19 20 21 22   20 21 22 23 24 25 26   18 19 20 21 22 23 24
23 24 25 26 27 28 29   27 28 29 30            25 26 27 28 29 30 31
30 31
```

Hint: your output does not necessarily need to be identical to the example. Nevertheless, you should make sure your output is well aligned and keeps each month clearly separated. (2 points)