universität innsbruck

Functional Programming	SS 2020	LVA 703025
Exercise Sheet 9, 10 points		Deadline: Monday, May 18, 2020, 10am

- Watch the lecture of week 9 and study part 6 of the slides up to slide no. 21[1].
- This week you have to write your own .hs-files from scratch. The finished files must all be uploaded to OLAT. There is no need to zip them. Exercise 9.1 can be uploaded as `.txt` file.
- You can use the template `.hs`-files that are provided on the proseminar page[2].
- Your .hs-files should be compilable with ghci.
- Don't forget to mark your completed exercises in OLAT.

Exercise 9.1 *Function Composition and Function Application*[3] 3 p.

1. Assume we have functions `f, g, h, i :: Int -> Int` and a value `x :: Int`. Which of the following expressions are type-correct?
   (a) `i $ h $ g $ f $ x`
   (b) `f $ g $ h $ i $ x`
   (c) `f . g . h . i $ x`
   (d) `f . g . h . i . x`
   (e) `\ x -> i . h . g . f`

   (1 point)

2. Which of the type-correct expressions from the previous exercise are equivalent to the Haskell expression `f (g (h (i (x))))`? (1 point)

3. Consider the following functions:
   ```
   not0 :: (Eq a, Num a) => a -> Bool
   not0 = (/=0)

   foo x = not0 (head (tail (tail x)))
   ```
   Rewrite function `foo` so that you can drop all parentheses using `(.)` and `$`.

   (1 point)

Exercise 9.2 *Modules* 7 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] `http://cl-informatik.uibk.ac.at/teaching/ws19/fp/slides/07x1.pdf`
[2] `http://cl-informatik.uibk.ac.at/teaching/ss20/fp/index.php#exercises`
[3] See part 4 of the slides `http://cl-informatik.uibk.ac.at/teaching/ws19/fp/slides/04x1.pdf`

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)`

   (3 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         "])
```

   (3 points)

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

```
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:[4]

```
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
```

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