universität innsbruck

Functional Programming	WS 2020	LVA 703025
Exercise Sheet 3, 10 points		Deadline: Wednesday, October 28, 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 3.1 *Parsing* **3 p.**

Consider the following Haskell expressions which contain superfluous parentheses.
Your task is to drop as many parentheses as possible without changing the meaning of the expressions, i.e., the abstract syntax tree that is obtained from parsing the expressions before and after dropping the parentheses must be identical.
Note: you cannot enter the expressions in GHCi, since they are not type-correct. Use the table of precedences on slide 2/18 instead.

1. `((f (g (3) (h 5 7)) 10) !! (((y - 7) > (5 || x))))` (1 point)
2. `((1 >>= (2 >> f (x && y))) . (((y ++ True) : 'a')))` (1 point)
3. `(((3 > 2) == ((3 ^ 5) < 42)) >>= (h && (x == y)))` (1 point)

Exercise 3.2 *Algorithm Design* **4 p.**

Use the design principles of divide-and-conquer and functional decomposition to write Haskell functions that determine if a number is even or odd.

1. Write a Haskell function `evenNat :: Integer -> Bool`, which takes a *non-negative* integer as input and determines whether it is even. (2 points)
2. Write `even :: Integer -> Bool`, which takes any integer and determines whether it is even. (1 point)
3. Write `odd :: Integer -> Bool`, which takes any integer and determines whether it is odd. (1 point)

In all of these programs you are not allowed to use the built-in multiplication-, division-, or modulo-functions.
Note: Haskell already ships its own version of `even` and `odd`. To test your program hide these by adding `import Prelude hiding (even, odd)` at the top of your program (as seen in the template .hs-file).

Exercise 3.3 *Enumerations* **3 p.**

The Austrian government currently gives every district (Bezirk) in Austria a color on the *corona-ampel*[1]. There are four possible colors (Green, Yellow, Orange and Red), of which Green is the least severe and Red is the most severe.

1. Define an enumeration type `Color`, and a function `severity :: Color -> Integer`, which maps each color to its severity on a scale from 0 (least severe) to 3 (most severe) using pattern matching on `Color`.

[1] https://corona-ampel.gv.at/

2. Define an enumeration type `District` for the 9 districts (Bezirke) in Tirol, and write a function `colorOf :: District -> Color`, which maps each district to its current color on the corona-ampel.

3. Using the functions `severity` and `hasColor` write a function `moreSevere :: District -> District -> Bool`, where `moreSevere x y` evaluates to `True` iff the district `x` is strictly more severe than district `y`.