Programming in Haskell – Midterm Exam

UNIZG FER, 2017/2018

Note: Define each function with the exact name and the type specified. You can (and in most cases you should) define each function using a number of simpler functions. Provide a type signature above each function definition and comment the function above the type signature. Unless said otherwise, a function may not cause runtime errors and must be defined for all of its input values. Use the `error` function for cases in which a function should terminate with an error message.

Each problem is worth a certain number of points. The points are given at the beginning of each problem, and are scaled to 10 upon grading.

You need at least *5 points* (after scaling) to pass this midterm exam. You have one hour for solving this exam. You are to solve it on your laptop and may use the Internet, save for communicating with one another, or third parties.

1. *(3 pts)* Write a function `listOfDigits :: [Int] -> [Int]` which takes a list of nonnegative integers and produces a list of their consecutive digits.

   ```
   listOfDigits [123,375,0,42] ⇒ [1,2,3,3,7,5,0,4,2]
   take 20 (listOfDigits [1..]) ⇒ [1,2,3,4,5,6,7,8,9,1,0,1,1,1,2,1,3,1,4,1]
   ```

2. *(4 pts)* Implement the function `anagramsFor :: String -> [String] -> [String]` that, given a word and a list of possible anagrams, select the correct sublist.

   Given “listen” and a list of candidates like “enlists”, “google”, “inLets”, “banana” the program should return a list containing “inLets”.

   Note: The input is **not** case-sensitive.

   ```
   anagramsFor "listen" ["inlets", "banana", "letins"] ⇒ ["inlets", "letins"]
   anagramsFor "listen" ["enlists", "inLets", "banana"] ⇒ ["inLets"]
   ```

3. *(3 pts)* Define a triangle data type which stores length of its 3 sides as `Int`. To make sure proper triangle is created provide a “constructor” function `makeTriangle :: Int -> Int -> Int -> Maybe Triangle` which checks if triangle is valid and than returns just a new triangle or nothing otherwise.

   Triangle is valid if any side is less than the sum of the other two sides.

   Here is a definition of `Maybe` data type (which is already included in `Prelude`) just as a reminder and some examples:

   ```
   data Maybe a = Just a | Nothing


   makeTriangle 100 20 20 == Nothing ⇒ True
   makeTriangle 3 4 5 /= Nothing ⇒ True
   ```