universität innsbruck

Functional Programming	WS 2020	LVA 703025
Exercise Sheet 10, 10 points		Deadline: Wednesday, January 13, 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 10.1 *Lists* **3 p.**

1. Write a function `removeFirst x xs` that removes the first occurrence of `x` in `xs`. If `x` is no element of `xs`, return the unmodified list.

   Examples:

   ```
   removeFirst 'a' "banana" == "bnana"
   removeFirst 5 [1,2,3,4] == [1,2,3,4]
   ```

   (1 point)

2. Two lists are permutations of each other, if they contain exactly the same elements – also with the same number of occurrences of each element – but where the order of the elements is irrelevant.

   Write a function `isPermutation xs ys` that returns `True` if its arguments are permutations of each other.

   Examples:

   ```
   isPermutation [1,2,1] [2,1,1] == True
   isPermutation [1,2,1] [2,2,1] == False
   ```

   Hint: `removeFirst` from the previous exercise could be useful.

   (1 point)

3. Write a function `hasDuplicates xs` that returns `True` if `xs` contains any duplicate elements.

   Examples:

   ```
   hasDuplicates [1,2,1] == True
   hasDuplicates [1,2,3] == False
   ```

   (1 point)

Exercise 10.2 *Type Classes* **2 p.**

1. Suppose that we have the following definition of the `member` function in Haskell:

   ```
   member :: Eq a => a -> [a] -> Bool
   member x [] = False
   member x (y:ys) | x == y    = True
                   | otherwise = member x ys
   ```

   Circle each type declaration that is a correct type for `member`.

   (a) `member :: Integer -> Integer -> Bool`

   (b) `member :: (Ord a) => a -> [a] -> Bool`

(c) `member :: (Integer -> Integer) -> [Integer -> Integer] -> Bool`

(d) `member :: (Eq a) => a -> [a] -> Bool`

(e) `member :: a -> [a] -> Bool`

(f) `member :: (Eq a) => [a] -> [[a]] -> Bool`

(g) `member :: Bool -> [Bool] -> Bool`

Hint: You likely get the most out of this exercise by only verifying your results with `ghci` instead of inferring them from it.

(2 points)

Exercise 10.3 *Perfect Numbers* **5 p.**

A perfect number[1] (`n :: Integer`) is a positive integer whose divisors (excluding `n` itself) sum to `n`. Take for example `6`. It's divisors (excluding `6`) are `[1,2,3]`. Adding these three gives `6`, therefore making `6` a perfect number. In this exercise we will try to find more perfect numbers by implementing the following functions.
Hint: List comprehension or using functions such as `map` and `filter` may be usefull for solving this exercise.

1. Implement the function `divisors :: Integer -> [Integer]`, which takes an integer `n` and returns the list of its divisors (excluding itself). (1 point)

   For example:

   ```
   divisors 1 == []
   divisors 3 == [1]
   divisors 6 == [1,2,3]
   ```

2. Using this implement a function `isPerfectNumber :: Integer -> Bool`, which checks if a given number is a perfect number. (1 point)

   For example:

   ```
   isPerfectNumber 6 == True
   isPerfectNumber 10 == False
   ```

3. Lastly you should implement a function `perfectNumbers :: Integer -> [Integer]`, which lists all perfect numbers up to the given integer. (2 points)

   For example:

   ```
   perfectNumbers 5 == []
   perfectNumbers 10 == [6]
   ```

4. Using your implementation find all perfect numbers up to 100, 1000 and 10000. Measure the time it takes to do so by using `:set +s` in ghci. (Depending on you implementation it may take a few seconds.) (1 point)

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