universität innsbruck

Functional Programming | SS 2020 | LVA 703025

Exercise Sheet 3, 10 points | Deadline: Monday, March 23, 2020, 10am

- Watch the lecture of week 3 and study part 3 of the slides up to slide no. 25[1].
- Please write all the Haskell code into a single .hs-file and upload it in OLAT.
- Exercises 3.1 and 3.5 can be added as comments to the .hs-file.
- You can use the template .hs-file that is provided on the proseminar page[2].
- Your .hs-file should be compilable with ghci.
- Don't forget to mark your completed exercises in OLAT.

Exercise 3.1 *Parsing* **2 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 part 2 / 19 instead.

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

Exercise 3.2 *Recursion* **2 p.**

1. Implement two functions that calculate the sum of all integers from 0 to n for any integer $n \geq 0$. One should count downwards (i.e., one recursion is from $n, n-1, \dots 0$), whereas the other should count upwards from 0 up to n. For the upwards solution you might need to introduce an auxiliary function of type `Integer -> Integer -> Integer`. (1 point)
2. The Fibonacci sequence is $0, 1, 1, 2, 3, 5, 8, 13, \dots$. It is starting with 0 and 1, and each further number is calculated by the sum of the two previous numbers in the sequence. Implement a function `fib :: Integer -> Integer` that calculates the nth Fibonacci number for any integer $n \geq 0$, i.e., `fib 0` should evaluate to 0, `fib 1` to 1, etc. (1 point)

Exercise 3.3 *Prime Numbers* **4 p.**

Use functional composition and divide-and-conquer to write a Haskell function that determines if a number is a prime number.
A prime number is a natural number greater than 1 that is only divisible by 1 and itself. For example: 2 is divisible by 1 and 2 and therefore a prime number; the same holds for 3, it is divisible by 1 and 3; but 4 is divisible by 1, 2 and 4 and therefore not a prime number.

[1] http://cl-informatik.uibk.ac.at/teaching/ws19/fp/slides/03x1.pdf
[2] http://cl-informatik.uibk.ac.at/teaching/ss20/fp/index.php#exercises

1. Write a Haskell function `isDivisible :: Integer -> Integer -> Bool` that takes two values and determines whether one is divisible by the other. You are **not** allowed to use the built-in `mod` function. (2 points)

2. Write a function `isPrime :: Integer -> Bool` that takes a natural number and returns whether that number is a prime number or not. Make use of your `isDivisible` function written in the previous task. (2 points)

Exercise 3.4 *Polymorphism* 1 p.

Write a function `threeEqual` taking three arguments of the same, arbitrary type, which returns `True` if and only if all of them are equal. Also write down the type signature.
Here are two examples how `threeEqual` should work:

```
threeEqual 1 1 1 == True
threeEqual True False False == False
```

Exercise 3.5 *FAQ* 1 p.

Study the slides, exercise sheets and sample solutions up until week 3. Come up with a question regarding the material or functional programming in general. Add it as a comment to your .hs-file or ask it in the OLAT forum[3]. Feel free to add multiple questions. (Remember: There are no stupid questions.)

[3]Don't use an anonymous nick if you ask in the OLAT forum. Otherwise it is not possible to assign points.