universität innsbruck

| Functional Programming | WS 2019/2020 | LVA 703024+703025 |
|---|---|---|

Exercises 3 — Deadline: October 23, 2019, 6am

- Please write all your Haskell functions from this exercise sheet into a single .hs-file and upload it in OLAT.
- You can use a template .hs-file that is provided on the proseminar page.
- The file should compile with ghci.
- Once the file has been uploaded, it cannot be changed or resubmitted!

## 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* 4 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 $n, n-1, \ldots 0$), 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 $0, 1, 1, 2, 3, 5, 8, 13, \ldots$. 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 $n$th Fibonacci number for any integer $n \geq 0$, i.e., `fib 0` should evaluate to 0, `fib 1` to 1, etc. (1 point)
3. Try your function in GHCI with large numbers, e.g. `fib 1000`. If the evaluation does not stop within a minute, then abort it and try to find the maximum value of `n` that the function can calculate within one minute. (1 point)
4. Optional question (without points): Can you figure out why it takes so long?
5. Define a more efficient function to calculate the $n$th element of the Fibonacci sequence with large values of $n$. It should count upwards as in the sum example (part 1). In particular the auxiliary function will be invoked as follows:

```
aux 0 1 ...   -- ... means that further arguments are possible
aux 1 1 ...
aux 1 2 ...
aux 2 3 ...
aux 3 5 ...
```

(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. 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)