universität innsbruck

Functional Programming	WS 2024/2025	LVA 703025
Exercise Sheet 7, 10 points		Deadline: Tuesday, November 26, 2024, 8pm

- Mark your completed exercises in the OLAT course of the PS.
- You can use a template .hs file that is provided on the proseminar page.
- Upload your modified `Template_07.hs` file in OLAT.
- Do not change the first lines of `Template_07.hs`, in particular do not add any `import` instructions.
- Your .hs file must be compilable with ghci.

Exercise 1 *Partial Application* — 2 p.

Consider the following functions:

```
div1 = (2 /)
div2 = (/ 2)
div3 x = x / 2
div4 x y = x / y
div5 x = (\ y -> y / x)
```

1. Explain what `div1` and `div2` and give the most general type signature for both functions (do not use GHCi to find the type signatures). Give an example that shows the difference between `div1` and `div2` and explain why they are different. (0.5 points)
2. We say that a Haskell function `f` is equal to a Haskell function `g`, whenever `f x1 .. xN = g x1 .. xN` for all inputs `x1`, . . . , `xN`. Based on this definition, which of the following pairs of functions are equal? Justify your answer.
   (a) `div2` and `div3` (0.5 points)
   (b) `div2` and `div4 2` (0.5 points)
   (c) `div4` and `flip div5` (0.5 points)

Exercise 2 *Lists* — 3 p.

The following tasks should be solved with the help of the common functions on lists, e.g. `filter`, `map`, `reverse`, `sum`. You are **not** allowed to use list comprehensions (which will be explained in lecture 8) or explicit recursion on lists. Use higher-order functions and λ-abstractions if appropriate.

1. Define a function `count :: (Eq a) => a -> [a] -> Int`, that given an element `x` and a list `xs`, counts the number of occurences of `x` in `xs`. (1 point)
   Examples:
   ```
   count 2 [1,2,2,3,4,2] == 3
   count 5 [1,2,2,3,4,2] == 0
   ```
2. Define a function `filterCount :: (Eq a) => Int -> [a] -> [a]`, that given a number `n` and a list `xs`, removes all elements that occur less than `n` times. (1 point)
   Examples:
   ```
   filterCount 2 [1,2,2,5,4,3,4] == [2,2,4,4]
   filterCount 3 [1,2,2,5,4,3,4] == []
   ```

3. Define a function `oddSquares :: [Int] -> [Int]`, that given a list of integers squares every odd number and removes every even number. (1 point)
   Examples:
   `oddSquares [1,2,3,4,5] == [1,9,25]`
   `oddSquares [2,4,6] == []`

Exercise 3 *Higher-Order Functions* **5 p.**

1. Define a higher-order function `sequence :: (a -> a) -> (a -> Bool) -> a -> [a]`. The first argument of `sequence` is a function which takes an element of the sequence and calculates the next element from it. The second argument is a function which takes an element and indicates whether the sequence should stop with the current element. The third argument of `sequence` should be the first element of the sequence.

   Examples:

   ```
   sequence (+ 1) (== 6) 0 == [0, 1, 2, 3, 4, 5]
   sequence succ (> 'z') 'a' == "abcdefghijklmnopqrstuvwxyz"
   ```

   (1 point)

2. Define a function `collatz :: Integer -> [Integer]` which returns the Collatz sequence starting with the given integer x. This sequence is defined as follows.

   - If the current sequence element is smaller or equal to 1, the sequence stops.
   - If the current sequence element is an even number y, the next element is $\frac{y}{2}$.
   - If the current sequence element is an odd number y, the next element is $3y + 1$.

   Use the function `sequence` from the previous task. Do not define named functions for the arguments of `sequence` but use λ-abstractions or sections.

   Examples:

   ```
   collatz 8 = [8,4,2]
   collatz 7 = [7,22,11,34,17,52,26,13,40,20,10,5,16,8,4,2]
   ```

   Remark. The second example shows that numbers in a Collatz sequence can look quite unpredictable. It is a long open problem, whether the Collatz sequence will stop for every possible starting value.

   (1 point)

3. Define a function `collatzLength :: Integer -> Integer -> [Int]` where `collatzLength a b` should return the list `[length (collatz a), ..., length (collatz b)]`. You can assume that `a <= b`.

   You are not allowed to define programs with recursion for this task. A succinct implementation easily fits into a single line.

   Examples:

   ```
   collatzLength 1 20 = [0,1,7,2,5,8,16,3,19,6,14,9,9,17,17,4,12,20,20,7]
   ```

   Hint: the `sequence` function might be useful, as well as other higher-order functions that have presented in lecture 7. (1 point)

4. Define a function `fastestSequence :: [(a -> a, a -> Bool, a)] -> [a]` with the following behavior of `fastestSequence xs`.

   - You may assume that `xs` is not empty.
   - Each triple `(next, abort, start)` in the list consists of parameters for the `sequence` function.
   - The output should be the sequence of those parameters that need the least amount of steps to finish. (If there are two parameter sets that require the same number of steps, then take the one which occurs earlier in the list)
   - You should not fully evaluate all sequences.

Example:

```
fastestSequence [
  ( (+ 5), (> 200), 20),
  ( (^ 2), (> 10000000000), 2),
  ( (\ x -> x), (== 43), 42)
 ]
```

results in `[2,4,16,256,65536,4294967296]` since this is the shortest of the three sequences.

Hint: do not invoke `sequence` and `length`; instead use `filter` and `map`.

Note: you just implemented a kind of parallel evaluation mechanism.

(2 points)