Instructions

Homework 2

This assignment deals primarily with recursion, guards, pattern matching and fold.

The questions 1 to 10 all have to be solved with Haskell.

Hint: No question (except for #11) needs more than three lines of code.

1. Using recursion, write a function

   ```
   myFoldl :: (a -> b -> a) -> a -> [b] -> a
   ```

   which behaves just like the standard `foldl`.

   ```
   > myFoldl (+) 0 [1..5]
   15
   ```

   (10 points)

2. Using the standard `foldl` (not `myFoldl`), write a function

   ```
   myReverse :: [a] -> [a]
   ```

   which behaves just like the standard `reverse`.

   ```
   > myReverse [1..4]
   [4,3,2,1]
   ```

   (10 points)

3. Using the standard `foldl` (not `myFoldl`), write a function

   ```
   myFoldr :: (a -> b -> b) -> b -> [a] -> b
   ```

   which behaves just like the standard `foldr`.

   (10 points)

4. Using the standard `foldr` (not `myFoldr`), write a function

   ```
   myFoldl2 :: (a -> b -> a) -> a -> [b] -> a
   ```

   which behaves just like the standard `foldl`.

   (10 points)

5. Write a function

   ```
   isUpper :: Char -> Bool
   ```

which returns true if the provided character is in the range 'A' to 'Z'.

Hint:
`elem` checks whether an element is in a set.
Ranges (like `[1..3]`) also work on characters.

(5 points)

6. Using the standard `filter`, write a function

   ```
   onlyCapitals1 :: String -> String
   ```

   which returns only the capital letters of the provided string

   ```
   > onlyCapitals1 "Hello, World!"
   "HW"
   ```

   Hint: Use the `isUpper` function from question 5

   (5 points)

7. Using list comprehension, write a function

   ```
   onlyCapitals2 :: String -> String
   ```

   which returns only the capital letters of the provided string.

   Hint: Use the `isUpper` function from question 5

   (5 points)

8. Using recursion, write a function

   ```
   onlyCapitals3 :: String -> String
   ```

   which returns only the capital letters of the provided string.

   Hint: Use pattern matching, guards and the `isUpper` function from question 5

   (5 points)

9. Write a function

   ```
   divRemainder :: Int -> Int -> (Int, Int)
   ```

   which returns a tuple with the quotient and the remainder of an integer division of the provided two numbers:

   ```
   > divRemainder 12 4
   (3,0)
   > divRemainder 23 5
   (4,3)
   ```

Hint:
`div` performs integer division
`mod` returns the modulus of an integer division

(5 points)

10. Write a function

    ```
    digitSum :: Int -> Int
    ```

    which returns the sum of the digits of the given integer:

    ```
    > digitSum 123
    6
    > digitSum 23069
    20
    ```

    Hint: Use recursion, guards and what you learned in question 9.

    (15 points)

11. Write a function

    ```
    sayNum x
    ```

    which takes a string of digits and spells out the number as string in English.

    The number will be between 1 and 10^66 - 1, so you have to support names for big numbers from *thousand* and *million* up to *vigintillion*. A good reference for these names can be found on Wikipedia: http://en.wikipedia.org/wiki/Names_of_large_numbers.

    Note: For this question, you can choose **any** programming language you want. C, C#, C++, D, Erlang, F#, Go, Groovy, Haskell, Java, JavaScript, Lisp, Lua, Perl, PHP, Prolog, Python, R, Ruby, Scala, Scheme and Smalltalk are all valid choices. Some of these languages have built-in support for big integers which might help with this task. Please specify how to compile the file or which language implementation to run it with.

    ```
    > sayNum "5"
    "five "
    > sayNum "23"
    "twenty three "
    > sayNum "82379"
    "eighty two thousand three hundred seventy nine "
    > sayNum "93218065"
    "ninety three million two hundred eighteen thousand sixty five "
    ```

    Hint: Use what you learned in question 10.

    (20 points)