Programming in Haskell – Homework Assignment 2

UNIZG FER, 2017/2018

Handed out: October 21, 2017. Due: October 28, 2017 at 23:00

1 Instructions

1. To submit your homework you need to have a folder named after your JMBAG. In that folder there should be two files, Homework.hs for homework tasks and Exercises.hs for all in class exercises (yes you need to submit those as well). You should ZIP that whole folder and submit it through Ferko.

   Example folder structure :

   - 0036461143
     - Homework.hs
     - Exercises.hs

   You can download homework template file from repository on FER web.

2. If you need some help with your homework or have any questions, ask them on our Google group.

3. Define each function with the exact name and type specified. You can (and in most cases you should) define each function using a number of simpler functions.

4. Unless said otherwise, a function may not cause runtime errors and must be defined for all of its input values (must be total). Use the `error` function for cases in which a function should terminate with an error message.

5. Problems marked with a star (⋆) are optional.

2 Grading

Each problem is worth a certain number of points. The points are given at the beginning of each problem or subproblem (if they are scored independently).

These points are scaled, together with a score for the in-class exercises, if any, to 10.

Problems marked with a star (⋆) are scored on top of the mandatory problems, before scaling. The score is capped at 10, but this allows for a perfect score even with some problems remaining unsolved.

3 Tasks

1. *(2 points)* Given a DNA strand, return its RNA complement (per RNA transcription).

   Both DNA and RNA strands are a sequence of nucleotides.

   The four nucleotides found in DNA are adenine (**A**), cytosine (**C**), guanine (**G**) and thymine (**T**).

   The four nucleotides found in RNA are adenine (A), cytosine (C), guanine (G) and uracil (U).

   Given a DNA strand, its transcribed RNA strand is formed by replacing each nucleotide with its complement:

   - `G -> C`
   - `C -> G`
   - `T -> A`
   - `A -> U`

   Implement function with type signature :

   ```
   toRNA :: String -> String
   ```

   Make sure you throw an error if the nucleotide doesn't exist.

2. *(3 points)* Extract the scrabble scores from a legacy system and convert them to the new system.

   The old system stored a list of letters per score:

   ```
   [(Int, String)]
   ```

   Example rendering of old data structure (you don't have to implement this):

   - 1 point: "A", "E", "I", "O", "U", "L", "N", "R", "S", "T",
   - 2 points: "D", "G",
   - 3 points: "B", "C", "M", "P",
   - ...

   The shiny new scrabble system instead stores the score per letter, which makes it much faster and easier to calculate the score for a word. It also stores the letters in lower-case regardless of the case of the input letters:

   ```
   [(Char, Int)]
   ```

   - "a" is worth 1 point.
   - "b" is worth 3 points.
   - "c" is worth 3 points.
   - ...

   Define a function which transforms data from the old system to the new one :

   ```
   transform ::  [(Int, String)] -> [(Char, Int)]
   ```

   Define a function that renders data from the new system

   ```
   render ::  [(Char, Int)] -> String
   ```

   in the following format :

   ```
   "a" is worth 1 point.
   "b" is worth 3 points.
   "c" is worth 3 points.
   ```

   Use backslash \to escape quotes and add newlines e.g.: `\"`, `\n`

3. *(3 points)* In this task you will have to implement a function which converts a number into it's English word counterpart.

```
numberToWords :: Int -> String
```

Since we haven't covered accumulators yet, adding "and" might be a bit more complicated, so instead of :

```
numberToWords 109324 ⇒ "one hundred and nine thousand three hundred
and twenty-four"
```

Your function should output :

```
numberToWords 109324 ⇒ "one hundred nine thousand three hundred twenty-four"
```

Also, you only need to implement conversion **for numbers up to millions**.

Here are some more examples :

```
numberToWords 1 ⇒ "one"
numberToWords 10 ⇒ "ten"
numberToWords 11 ⇒ "eleven"
numberToWords 100 ⇒ "one hundred"
numberToWords 115 ⇒ "one hundred fifteen"
numberToWords 1213 ⇒ "one thousand two hundred thirteen"
numberToWords 1005 ⇒ "one thousand five"
numberToWords 22213 ⇒ "twenty-two thousand two hundred thirteen"
...
numberToWords 1000000 ⇒ "one million"
numberToWords 1000001 ⇒ "one million one"
numberToWords 1002001 ⇒ "one million two thousand one"
```

4. *(3 points(⋆))* During last lecture we have mentioned `undefined` and how useful it is as a placeholder. It is a single value that can match any type. This might make it seem like some kind of special language construct but it is actually very simple to define such value.

   Try to define your own version of `undefined` :

   ```
   undefined' ::  a
   ```

   Hint : Hint