CSCE 314: Programming Languages

Homework Assignment 3

Objective

In this homework you'll practice writing functions in Haskell, especially with an emphasis on recursive functions.

Submission

- Like before, this homework is will be graded as a 0, 1, or 2 for each question. (The questions are: "Basic Haskell Drills", "Aaah!", "Water Gates", "Goldbach's other conjecture"). With 0 = inadequate or no attempt; 1 = solid effort showing a good attempt at all questions; 2 = faultless/exemplary submission. You may find it helpful to discuss your homework with the peer teachers, TAs, or instructors (say, at their office hours).
- Prepare a **single** Haskell file with a `.hs` extension.
- *Recommended*: Include as a comment at the top of your code: your name and UIN.
- The file should contain functions named precisely as described below. Those functions must have proper type signatures and should follow exactly the ones that have been explicitly given. In addition, you may include any number of additional helper functions you like; you may not include any extra modules, packages, or external libraries.
- Additional documentation in the form of comments throughout your code is strongly encouraged. If you have used outside resources, document these directly in your source file.
- Also, please use the class slack #hw-3 channel to communicate about/solicit help/ask questions clarifying aspects of the assignment.
- Submission will be via Canvas, due: 25 October 2021.

Instructions

- This assignment asks you to implement a set of Haskell functions. Each of those functions can be defined with just a few lines of code. If your implementations start to get much longer, it may be a sign that there are less complicated solutions that you may be missing.
- Also, you shouldn't need any functionality beyond items that are in the standard prelude. Appendix B in the textbook for a list of those functions.

Basic Haskell Drills

For each of the following problems, write: (a) A Haskell function type definition and (b) an actual Haskell function of the given name.

If the problem notes that it is recursive, use a recursive implementation (other questions could be recursive, also, and there could be non-recursive solutions).

1. `myReverse`: given a list, reverse it. (recursive)
2. `isElement`: judge if a value is an element of a list. If it is, return True, else return False. (recursive)
3. `duplicate`: duplicate the elements of a list. For example, `duplicate [1,2]` returns `[1,1,2,2]`. (recursive)
4. `removeDuplicate`: given a sorted list, remove the duplicated elements in this list. (recursive)

5. `rotate`: rotate a list n places to the left, where n is an integer. For example, `rotate "abcde" 2` returns `"cdeab"`.
6. `flatten`: flatten a list of lists into a single list formed by concatenation. For example, `flatten [[1,2],[3,4],[5,6]]` returns `[1,2,3,4,5,6]`. (recursive)
7. `isPalindrome`: given a list, judge if it is a palindrome.
8. `coprime`: given two positive integer numbers, determine whether they are coprime. Two numbers are coprime if their greatest common divisor equals 1. Hint: Euclid's algorithm. (recursive)

Aaah! (Easy)

When we go to see a doctor, the doctor always asks us to say "aaah". Sometimes, the doctor needs us to say "aaaaaah", but we are only able to say "aaah". In that case, the doctor is unable to diagnose our disease, because the 'a's in our "aaah" are fewer than his or her requirements. Now, write a Haskell function called `seeDoctor` to judge if the doctor can diagnose us with our "aah". The input of the function consists of two strings. The first string is the "*aaaah*" the doctor needs and the second string is the "*aah*" we are able to say. Output "True" if our "*aah*" meets the requirements of the doctor, and output "False" otherwise. The test should pass with a "True" only when lowercase 'a's and 'h's are used, and each string contains a certain number of 'a's followed by a single 'h'.

```
seeDoctor:: String ->  String ->  Bool
```

Water Gates (Medium)

There are *n* water gates in a reservoir that are initially closed. To adjust water in the reservoir, we open/close the water gates according to the following rule: on the first day, we open all the gates. Then, on the second day, we close every second gate. On the third day, we change the state of every third gate (open it if it's closed or close it if it's open) ... Following the same general pattern, for the *i*th day, we change the state of every *i*th gate. Finally, for the *n*th day, we change the state of the last gate. Our question is, how many gates are open after n days?

```
Given n = 4.
At first, the gates are [closed, closed, closed, closed].
After the first day, the gates are [open, open, open, open].
After the second day, the gates are [open, closed, open, closed].
After the third day, the gates are [open, closed, closed, closed].
After the fourth day, the gates are [open, closed, closed, open].
So you should return 2, because there are two gates open.
```

Write a Haskell function called `waterGate` that computes how many of *n* water gates are open after *n* days.

```
waterGate :: Int -> Int
```

Goldbach's other conjecture (Hard)

Goldbach's other conjecture: Christian Goldbach once proposed that every odd composite number can be written as the sum of a prime and twice a square. For example,

```
9 = 7 + 2×1^2

15 = 7 + 2×2^2

21 = 3 + 2×3^2

25 = 7 + 2×3^2

27 = 19 + 2×2^2
```

```
33 = 31 + 2×1^2
```

However, this conjecture turns out to be false. Write a Haskell function called `goldbachNum` to find the smallest odd composite that does not match Goldbach's other conjecture.

```haskell
goldbachNum :: Int
```

Acknowledgements

Basic Haskell Drills are adapted from Dr. Dylan Shell and Dr. Hyunyoung Lee's previous CSCE 314 materials and "H-99: Ninety-Nine Haskell Problems".
Aaah! problem is adapted from 2012 Nordic Collegiate Programming Contest, October 6, problem A.
Water Gates problem is adapted from the bulb switcher problem at Leetcode, problem 319.
Goldbach's other conjecture problem is from Project Euler, problem 46.