Lab 7

These exercises should help you get started with Haskell. You may need to refer to ch. 1 and 2 in the book for examples that will help you in solving the exercises.

1. List comprehensions

a) Outfits. A notion of basic wardrobe suggests that it is possible to create a large number of outfits using only a few "basic" clothing articles. Consider the person who owns 2 pairs of pants (jeans and khaki), 4 pairs of tops (white shirt, grey turtleneck, pink polo, green hoodie) and 3 pairs of shoes (brogues, converse, sandals). Write a list comprehension expression that generate a list of tuples of all possible outfits that can be constructed with these items.

b) You will now analyze a basic wardrobe from a). Assign resulting list from part a) to "variable" outfits. Type

```
length outfits
```

to see how many different outfits can be constructed from these few items.
Try adding more items to each list and observe the effect on the resulting number of outfits. If you only have budget for two more articles, does it make sense to buy more items of the same type or different type? (e.g. buy two more tops or by a top and a pair of shoes)?
Which two articles will yield the maximum increase in the number of outfits?
Now consider adding another category (say, scarves) with two elements in it (paisley scarf, knit loop). Update your expression to generate a list of 4-item outfits. How many outfits are generated now?
Type up your answers as comments below the code for outfit list generating expression.

c) There's a fancy event that people can attend solo or as a couple. Given a list l of honorifics accepted in some culture, write a Haskell expression (that includes list comprehension) that contains all possible combinations of prefixes for the guest(s). Be sure to accommodate both single and couple guests. E.g. if the original honorific list is ["Mr.", "Ms."], then the resulting list should be ["Mr.", "Ms.", "Mr. and Mr.", "Ms. and Mr.", "Mr. and Ms.", "Ms. and Ms"]. Order does not matter.
For testing, use l = ["Mr.", "Ms.", "Mrs.", "Dr.", "Prof.", "Rev."]

d) NATO phonetic alphabet is used to avoid errors in military communication
https://en.wikipedia.org/wiki/NATO_phonetic_alphabet
Each character is spelled as a corresponding word in the alphabet. Assume the alphabet is stored in a list `l` as list of tuples with 'Character',"Word" correspondence, and the word is stored in `word`. Write a list comprehension expression that translates `word` into its military spelling, i.e. where "DAY" becomes ["Delta", "Alpha", "Yankee"]. You can assume only uppercase characters are used to specify the word. For your convenience, NATO alphabet is provided.

2. Functions

a) Implement the `remainder` function that takes two integers as input and returns a remainder of dividing the first one by the second one.

b) Implement `isEven` function that takes one integer as input and returns True if it is even, False otherwise.

c) Implement the `merge` function that takes two sorted lists as parameters and returns the merged sorted list.

d) Write a function `removeMultiple` which takes a list `L` and a number `a` as arguments, and it returns a new list containing all the numbers in `L` that are not multiples of `a`. *(Hints: (1) It may be easier to create the output list with numbers in reverse order. That is fine as is; or you can use `reverse` afterwards. (2) You may want to use the `remainder` function from part a).*

3. List functions

Rewrite your expressions from Part 1 as functions. As a result, you should have 3 functions:

outfits

honorifics lst (takes in a list of possible choices)

nato wrd (takes in a word to be translated)

Manually test your functions by calling them with several different values.

Lab 7 continued

Validating Credit Card Numbers

Have you ever wondered how websites validate your credit card number when you shop online? They don't check a massive database of numbers, and they don't use magic. In fact, most credit providers rely on a checksum formula called the Luhn Algorithm for distinguishing valid numbers from random collections of digits (or typing mistakes).

In this assignment, you will implement the Luhn Algorithm. Pseudocode for the algorithm is provided below:

- Double the value of every second digit beginning from the right. That is, the last digit is unchanged; the second-to-last digit is doubled; the third-to-last digit is unchanged; and so on. For example, `[5,5,9,4]` becomes `[10,5,18,4]`.

- Add the digits of the doubled values and the undoubled digits from the original number. For example, `[10,5,18,4]` becomes `(1 + 0) + 5 + (1 + 8) + 4 = 19`

- Calculate the remainder when the sum is divided by 10. For the above example, the remainder would be `9`.

- If the result equals 0, then the number is valid.

The following series of exercises will lead you to a completed implementation of the algorithm. In your code, clearly mark each step with Haskell comments.

Exercise 1 We first need to be able to break up a number into its last digit and the rest of the number. Write these functions:

```
lastDigit :: Integer -> Integer
dropLastDigit :: Integer -> Integer
```

Example: **lastDigit 123 == 3**
Example: **lastDigit 0 == 0**
Example: **dropLastDigit 123 == 12**
Example: **dropLastDigit 5 == 0**

Note that some test cases for these functions have been provided in LAB8Tests.hs. To run the tests for this exercise, load LAB7Tests.hs in to GHCi and type **runTests ex1Tests**. The result is a list of failures; if the list is empty then all of the tests passed. You should add your own test cases for the remaining exercises.

Exercise 2 Now, we can break apart a number into its digits. It is actually easier to break a number in to a list of its digits in reverse order (can you figure out why?). Your task is to define the function

```
toRevDigits :: Integer -> [Integer]
```

toRevDigits should convert positive **Integer**s to a list of digits. (For **0** or negative inputs, **toRevDigits** should return the empty list.)

Example: **toRevDigits 1234 == [4,3,2,1]**
Example: **toRevDigits 0 == []**
Example: **toRevDigits (-17) == []**

It is easy to define a function that gets a list of digits in the proper order in terms of **toRevDigits**:

```
toDigits :: Integer -> [Integer]
toDigits n = reverse (toRevDigits n)
```

However, you will likely not need to use it for this lab (you're welcome to implement it from scratch as an auxiliary exercise).

Exercise 3 Once we have the digits in a list, we need to double every other one. Define a function

```
doubleEveryOther :: [Integer] -> [Integer]
```

Remember that the Luhn algorithm should double every other digit beginning from the right, that is, the second-to-last, fourth-to-last, . . . numbers are doubled. It's much easier to perform this operation on a list of digits that's in reverse order. Conveniently, the function you defined in the previous exercise gives you the digits in reverse order. The function **doubleEveryOther** should thus take in a list that is already in reverse order and double every other number starting with the second one.

Example: **doubleEveryOther [4, 9, 5, 5] = [4, 18, 5, 10]**
Example: **doubleEveryOther [0, 0] = [0, 0]**

Exercise 4 The output of **doubleEveryOther** has a mix of one-digit and two-digit numbers. Define the function **sumDigits :: [Integer] -> Integer**
to calculate the sum of all digits.

Example: **sumDigits [10, 5, 18, 4] = 1 + 0 + 5 + 1 + 8 + 4 = 19**

Exercise 5 Define the function
luhn :: Integer -> Bool
that indicates whether an **Integer** could be a valid credit card number. This should use all functions defined in the previous exercises. Remember that your implementation of **doubleEveryOther** gives you a list in reverse order. Do you need to account for this?

Example: **luhn 5594589764218858 = True**
Example: **luhn 1234567898765432 = False**

If you want more credit card numbers to test on you can get some at **http: //www.getcreditcardnumbers.com**. You can test on your own credit cards as well, but you probably don't want to submit test cases with your credit card information.

Submission

Upload your warmup.hs, LAB7.hs (with function definitions) and LAB7Tests.hs .