Functional and Logic Programming

Home Assignment 1

Due: Wednesday, 27.3.2019 - 23:55

Instructions

- Please create a source file called **_hw1.hs_** and put all the answers there.
  The file should start with a comment which contains your **full name** (in English) and **ID.**

  -- John Doe
  _-- 654321987_

- Make sure the file **is valid** by loading it into GHCi.
  A valid file will load without any errors or warnings.

- If you need a function but you don't know how to implement it - just write it's signature (name and type) and put `undefined` in the function's body.
  That way you'll be able to load the file even though it contains references to undefined names.

- When writing a function - write both the **type** and the **body** of the function.

- Be sure to write functions with **exactly the specified name** (and **type signature** - if it is provided) for each exercise.
  You may create additional auxiliary functions with whatever names and type signatures you wish.

- Try to write **small functions** which perform just **a single task**, and then **combine** them to create more complex functions.

Exercises

1. Write a function `isPali`, which takes a **String** and returns **True** if the **String** is a palindrome and **False** otherwise.

Example:

```
isPali ""          = True
isPali "x"         = True
isPali "radar"     = True
isPali "racecar"   = True
isPali "lambda"    = False
isPali "haskell"   = False
```

2. Write a function `prefix`, which takes 2 **Strings** and returns **True** if the first **String** is a prefix of the second **String**, and **False** otherwise.

Example:

```
prefix "pre" "prefix"   = True

prefix  ""   "word"     = True

prefix "abc"  "abcde"   = True

prefix "abc"  "abc"     = True

prefix "abcde" "abc"    = False

prefix "abc"   "def"    = False
```

3. a) Write a function `doubleList`, which takes an **Int** n and produces a list of the first n integers (0,1,2,3,..n) doubled in **ascending** order, using the operator **++**.
(You may assume the input n is always non-negative and that $0 \in \mathbb{N}$)

Example:
`doubleList` **0 = [0]**
`doubleList` **3 = [0, 2, 4, 6]**
`doubleList` **5 = [0, 2, 4, 6, 8, 10]**

b) Write a function `listDouble,` which takes an **Int** n and produces a list of the first n integers (0,1,2,3,..n) doubled in **descending** order.
(You may assume the input n is always non-negative and that $0 \in \mathbb{N}$)

Example:
`listDouble` **0 = [0]**
`listDouble` **3 = [6, 4,2, 0]**
`listDouble` **5 = [10,8,6,4,2,0]**

4. Write a function `myGcd` which calculates the Greatest Common Divisor of 2 numbers.
https://en.wikipedia.org/wiki/Greatest_common_divisor

NOTE: You may use auxiliary functions

Example:
`myGcd` **3 6= 3**
`myGcd` **3 7 = 1**
`myGcd` **6 8 = 2**
`myGcd` **121 121 = 121**
`myGcd` **0 8= 8**
`myGcd` **1 8= 1**

5. In this exercise we will implement a **credit card validation function** using a **checksum**. The checksum algorithm is as follows:

- 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, **[1, 3, 8, 6]** becomes **[2, 3, 16, 6]**.
- Add the digits of the doubled values and the undoubled digits from the original number.
  For example, **[2, 3, 16, 6]** becomes **2 + 3 + 1 + 6 + 6 = 18**.
- Calculate the remainder when the sum is divided by **10**. If the result equals **0**, then the number is valid.

Implement the following steps of the validation algorithm:

a) Create a function:

```
toDigits :: Integer -> [Integer]
```

which converts positive **Integer**s to a list of its digits.
(For **0** and negative numbers it should return the empty list)

Example:

```
toDigits 1234 = [1, 2, 3, 4]
toDigits 0    = []
toDigits (-17) = []
```

b) Create a function:

```
doubleIntermittent :: [Integer] -> [Integer]
```

that doubles every other number in a list *beginning from the right* - so it doubles the second-to-last number, the fourth-to-last number, etc.

Example:

```
doubleIntermittent [] = []
```

```
doubleIntermittent [1] = [1]
```

```
doubleIntermittent [1,2] = [2,2]
doubleIntermittent [1, 2, 3] = [1, 4, 3]
```

```
doubleIntermittent [8, 7, 6, 5] = [16, 7, 12, 5]
```

c) Create a function:

`sumDigitsInList` **:: [Integer] -> Integer**

that takes a list of numbers (not necessarily one-digit numbers) and returns the sum of all the digits of all the numbers in it.
If the list is empty it returns **0**.

Example:

`sumDigitsInList` **[16, 7, 12, 5] = 1 + 6 + 7 + 1 + 2 + 5 = 22**

d) Create a function:

`validate` **:: Integer -> Bool**

that indicates whether a given **Integer** is a valid credit card number by checking if the remainder of the given sum divided by **10** is equal to **0**.
This will use all the functions defined in the previous sections.

Example:

`validate` **4012888888881881 = True**
`validate` **4012888888881882 = False**

6.

a) Implement the function `mySum` which takes a list of Integers and returns the sum of the list's elements (not tail recursive).

`mySum` **:: [Int] -> Int**

Example:

`mySum` **[] = 0**
`mySum` **[1] = 1**
`mySum` **[1,2] = 3**

b) Implement the function `mySumTail` which takes a list of Integers and returns the sum of the list's elements (tail recursive).

`mySumTail` **:: [Int] -> Int**

NOTE: you can assume that the sum of the elements of an empty list is 0.

Example:

`mySumTail` **[] = 0**
`mySumTail` **[1] = 1**
`mySumTail` **[1,2] = 3**

7. A perfect number is a natural number which is equal to the sum of its natural devisors
(except the number itself).
Create a function:

`perfect` **:: Integer -> Bool**

which takes an Integer and checks if it is a perfect number.
You can assume that the input will be a positive number.

Example:

```
perfect    0 =   True
perfect    6 =   True  (because 6=0+1+2+3)
perfect   28 =   True
perfect  496 = True
perfect    1 =   False
perfect 8130 = False
```