CS3490: Homework 1

1. Objectives

The main goal of this homework is to solidify your understanding of basic Haskell syntax. You should strive to master the following items by the time the assignment is due:

- The syntax and meaning of basic arithmetic and boolean operations:

  `+,-,*,/,^,**,mod,div,min,max,abs,exp,sqrt,...`

  `&&,||,not,==,/=,<=,<,>=,>...`

- Defining new functions from existing ones by combining them together
- Using recursion instead of loops to define functions on the integers, like factorial, sum-of-odd-numbers-up-to-n, etc.
- Defining functions of multiple variables
- **Declaring and understanding the types of functions**

  In particular, you should be able to understand how to declare a function that takes two numbers and returns a boolean, or takes a number and a list and produces a new list, or takes a list and produces a number, etc.
- Using conditionals (`if-then-else`), guards, and/or pattern-matching
- Using list comprehensions and ranges. For example, the set of all numbers from 1 to 50 divisible by 7: `[x | x <- [1..50], mod x 7 == 0]`
- A few basic functions used for processing lists:

  `(:) (++) null elem reverse length take drop head tail init last`

  Make sure you remember and understand the types of these functions!

  Also, make sure you can explain the difference between the operators `++` and `:`

Additional hints

- Most of the material required for these functions is covered in Chapter 2 of the tutorial. Some harder questions may require you to look in Chapter 4.
- *Do not fear compiler errors!* An error means the compiler has detected that you wrote something different than what you meant to write. It will usually try to explain what exactly the problem is.

2. Warm-up/Practice problems

The practice problems are grouped roughly by some shared themes.

2.1. Arithmetic operations

1. Implement the function $f(x) = 2x+1$ as a Haskell function `f :: Double -> Double`

2. Implement the function $g(x, y) = x^y + \sqrt{y}\sin(\pi x)$ as a Haskell function `g :: Double -> Double -> Double` that takes *two* arguments.

3. Implement a function `h1 :: Double -> Double` that behaves as follows:

$$h_1(x) = \begin{cases} 2x+1 & \text{when } x \geq 0 \\ x^2 & \text{when } x < 0 \end{cases}$$

4. Implement a function `h2 :: Double -> Double -> Double` that behaves like:

$$h_2(x, y) = \begin{cases} 0 & x^2 < 10 \\ y^2 & x^2 \geq 10, x + y < 100 \\ 2^x - y^2 & x^2 \geq 10, x + y \geq 100 \end{cases}$$

2.2. Recursion vs. list comprehension

For each function below, provide *two* implementations. One using list comprehension and/or Haskell's built-in `sum` function. The other, using recursion on the input number, outputing 0 when the input is 0 or less, and performing a recursive call on the previous number when the input is positive.

To differentiate between the two implementations, add an apostrophe to the name of the second function: `sumUp` and `sumUp'`, etc.

1. `sumUp :: Integer -> Integer` computes the sum of all numbers from 0 to the given integer, inclusive.

2. `sumSquares :: Integer -> Integer` computes the sum of the *squares* of all the numbers from 0 to the given integer, inclusive.

3. `sumOddSquares :: Integer -> Integer` computes the sum of squares of all *odd* numbers from 0 to the given integer, inclusive.

4. `sumOddSquaresRange :: Integer -> Integer -> Integer` computes the sum of squares of all odd numbers between the given two bounds. (Inclusive.)
   `sumOddSquaresRange 5 10 = 155`

2.3. Testing Collatz conjecture

1. The *Collatz function* is defined as following:

$$f(n) = \begin{cases} 1 & n = 0 \text{ or } n = 1 \\ f(n/2) & n > 1 \text{ is even} \\ f(3n+1) & n > 1 \text{ is odd} \end{cases}$$

   Write a Haskell function `collatz :: Integer -> Integer` so that `collatz n` equals $f(n)$ for every $n \geq 0$.

2. The *Collatz Conjecture* states that $f(n)$ always returns 1 when $n > 0$. No one knows if this is true. You will be very famous if you solve this problem, see `https://en.wikipedia.org/wiki/Collatz_conjecture`.

   Verify that the Collatz conjecture is true for $1 \leq n \leq 100$, by using list comprehension to construc a list `collatzCheck :: [Integer]` consisting of the values of $f(n)$ in that range.

   Hint. You can define a value of any type, including list, just like you would define a function, except there are no input arguments. For example, you can write

```
myNum :: Integer
myNum = 100
myList :: [Integer]
myList = [1,4,10,20]
collatzCheck :: [Integer]
collatzCheck = ?
```

2.4. List manipulation

Refer to the last bullet point on the first page of this document. The following questions can be solved by combining Haskell's built-in list functions in some ways.

1. `getSecond :: [String] -> String` returns the second element in a list of strings. It can crash if such element doesn't exist.

2. `makePalindrome :: String -> String` takes a string and appends it to its reverse: `makePalindrome "hello" = "helloolleh"`.

   Hint. In Haskell, strings are lists of characters: `String = [Char]`. Therefore, you can apply the same operations to pure strings that you can apply to arbitrary lists.

3. `skip3 : String -> String` removes the first three characters from a given string. It may error if the string has length less than 3.

4. `find7 : [Integer] -> Bool` returns `True` if the input list contains the number 7, and returns `False` otherwise.

3. Homework assignment

This final list contains the homework problems. Save your solutions to these in a separate file, and save the file as `homework1.hs`. *Do not include any personally identifying information in the file, such as your name, student id, etc.*

1. Write a function `radius :: Double -> Double -> Double` which takes two floating-point numbers, x and y, and returns the distance from the point (x, y) to the origin.
2. Write a function `sumEvens :: Integer -> Integer` which adds up all the even numbers from 1 to its input argument (inclusive, if applicable).
   `sumEvens 5 = 6`, `sumEvens 8 = 20`.
3. Using ranges and/or list comprehension, create a list of all numbers $1 \leq n \leq 200$ divisible by 17. Save this as a Haskell term `multiplesOfSeventeen :: [Integer]`.
4. Write a function `multiplyEnds :: [Integer] -> Integer` which multiplies the first and the last element of the given list. *If the list is empty, it should return 1.*
5. Write a function `getLengths :: [String] -> [Int]` which takes a list of strings and returns a list containing their *lengths*:
   `getLengths ["Hello","World"] = [5,5]`
6. Write a function `dropLastTwo :: [Integer] -> [Integer]` which returns all but the last two elements. (Assuming they exist.)
7. Write a function `findEmpty :: [String] -> Bool` that takes a list of strings, and determines whether it contains an empty string.
8. Write a function `checkPalindrome :: String -> Bool` that returns `True` if the string is the same when traversed in opposite direction. (*Hint.* You can use Haskell's equality operator `==` to check whether two strings are equal.)
9. Write a function `checkSize :: [Integer] -> Bool` which takes a list of integers, and returns `True` if the number of elements in the list is at least 3, and the first element in the list is at least 10. (*Hint.* Use the `length` function.)
10. Write a function `checkAnySize :: Integer -> [Integer] -> Bool`, which works similar to the previous function, but it now takes an additional integer parameter as the *first* argument, and checks whether both the length of the list and the first element in the list are at least as big as this parameter.

    `checkAnySize 4 [5,2,10] = False`, `checkAnySize 4 [5,2,10,0,-1] = True`.

Submission instructions

Save your file as `homework1.hs`. Submit it to asulearn by the deadline. *Your file must compile/load correctly.* If you can't get a function to work, provide a dummy value of the correct type that the function is supposed to return. For example,

```
dropLastTwo xs = []
```