CPSC 449: Exercise 1

Fall 2022 (Revised July 22nd, 2022)

Due: Friday, September 23th (2022) at 11:59 PM midnight

For this exercise you are expected to develop the programs below. The tutorials will go over the starred questions to help get you going. You are expected to submit your programs – including the **starred** (*) questions – to gradescope using the template which is provided. You can submit as often as you like to gradescope before the deadline: the system will test and grade the programs you have completed.

I encourage you to discuss these programs with your classmates and in the tutorials: the aim is to complete as many of these exercises as you can so that you get used to thinking in Haskell.

Warning: the problems become progressively harder! You should comment your code indicating, in particular, how you arrived at a solution especially if it is not your own.

It is important that you understand the solutions and how to develop programs in Haskell as this comprehension will be tested in the laboratory written test and your ability to program will be further tested by the assignment.

Please name your functions according to what is prescribed below. Please, for this exercise, avoid using **`Prelude`** functions. If there are name conflicts with names defined in **`Prelude`**, then (a) explicitly import **`Prelude`**, and (b) use the **`hiding`** clause to hide the conflicting names when importing (see the grey box on page 53 of **[Thompson]**).

1. Write a function

   ```
   avgThree:: Int -> Int -> Int -> Float
   ```

   to take the average of three integers which returns a float.

2.* Write a function

   ```
   maxThree:: Int -> Int -> Int -> (Int,Int)
   ```

   which works out the maximum of three integers and returns also how many times that maximum occurs.

3. Write a function

   ```
   data SF a = SS a | FF

   invExp:: Integer -> SF Integer
   ```

which returns the largest number n such that 2^n is no greater than the given number (what happens if the given number is negative?).

4. Implement a function **`myLcm`** that takes two integers as arguments, and returns the least common multiple. One way to do this is to multiply the two numbers together and divide by the greatest common divisor (using the **Euclid's Algorithm**).

   ```
   myLcm :: Int -> Int -> Int
   ```

   You may not assume that the arguments are both positive: the least common multiple is always taken to be a positive number!

5.* The binomial coefficient $\binom{n}{k}$ is defined as follows for integers $n \geq 1$ and $0 \leq k \leq n$:

$$\text{binom } k\ n\ = \binom{n}{k} = \frac{n \times (n-1) \times \ldots \times (n-k+1)}{(1 \times 2 \times \ldots \times k)}$$

   Write a function:

   ```
   binom:: Integer -> Integer -> Integer
   ```

   to calculate the binomial coefficients.

6. Write a function

   ```
   grow :: String -> String
   ```

   which changes a string $a_1a_2a_3...$ to $a_1a_2a_2a_3a_3a_3...$ so `grow "now!" == "noowww!!!!"`.

7.* Write a function

   ```
   instrictorder:: [Int]-> Bool
   ```

   which tests whether the list of integers is strictly decreasing.

8. Write a function

   ```
   expensive:: [(String,Int)] -> Int -> [String]
   ```

   which given a list of items and their cost (here just an integer) returns a list of items whose cost is (strictly) above a given threshold cost.

9.* Write a function

   ```
   sortCheapest :: [(String,Int)] -> [(String,Int)]
   ```

   which, given a list of items with a cost, returns a list in cheapest first order.

10. Write a function

    ```
    divisors:: Integer -> [Integer]
    ```

    which calculates the list (in ascending order) of all prime divisors of an integer which are no bigger than the number. (Hint: What are the prime divisors of 0 which are no bigger than zero?)

11.* Write a function

    ```
    substring :: String -> String -> Bool
    ```

    which determines whether a given first string is a substring (or "infix") of a second string.

12. Write a function

    ```
    sublists:: [a] -> [[a]]
    ```

    which given a list of *any* type returns the list of all sublists of that list.