UNIVERSITÄT BASEL

Thorsten Möller
Maarten Schenk - maarten.schenk@unibas.ch
Michael Plüss - m.pluess@unibas.ch

UNI BASEL

Programming Paradigms – Haskell FS 2019

Exercise 4 Due: 12.05.2019 23:59:59

Upload answers to the questions **and source code** before the delivery deadline via courses.cs.unibas.ch. In addition, running programs have to be demonstrated during the exercise slots until **Friday, 17.05.2019** the latest. Also note, of all mandatory exercises given throughout the course, you must score at least 2/3 of their total sum of points to get accepted in the final exam.

Modalities of work: The exercise can be completed in groups of at the most 2 people. Do not forget to provide the full name of all group members together with the submitted solution.

Question 1: Bite-sized Haskell Tasks (8 points)

For each of the following task descriptions write a Haskell function that completes the task.

We do want to keep the solutions to these problems bite-sized; so an additional restriction is that they must not make use of additional helper functions other than functions that are already predefined in Haskell.

a) Append and prepend the same value to a given list.

(1 points)

b) Generate a list of tuples (n, s) where $0 \leq n \leq 100$ and $n \mod 2 = 0$, and where $s = \sum_{i=1}^{n} i$; i.e., the output should be the list `[(0,0),(2,3),(4,10),...,(100,5050)]`.

(1 points)

c) Swap the values at two specifiable positions in a list.

(2 points)

d) Count the number of prime factors of an integer value.

(1 points)

e) Calculate the euclidean norm (l^2-norm) of a list of floats. For reference: `https://en.wikipedia.org/wiki/Norm_(mathematics)#Euclidean_norm`

(1 points)

f) Split a list into two lists, such that the first list's length is the third of the length of the original list, rounded up. Eg. [1,2,3,4,5] is split into [1,2] and [3,4,5].

(1 points)

g) Return, as a `Boolean`, whether the first and the second last element in a list are not the same.

(1 points)

Question 2: Lazy Evaluation (Call-By-Need) (3 points)

a) Using the concept of lazy evaluation, write a function that returns the infinite list of natural numbers.

(1 points)

b) Now create from the above infinite list a new one consisting of tuples. Calculate the tuples the following way: `[(1,1/1),(2,1/2),...,(i,1/i), ...]`.

(1 points)

c) As a last step, write a function using the above infinite list of tuples that takes a number and returns a list of all the tuples where the second element is greater or equal to the given number.

(1 points)

Hint: The functions `iterate`, `map` and `takeWhile` might be useful to solve this question.

Question 3: Guards and Pattern Matching (5 points)

You are tasked to implement a recursively defined function. The function takes three parameters. Here is the definition:

$$f(x,y,z) = \begin{cases} f(x-4,y-1,z-2)+|x+z| & \text{if } x>2 \text{ and } y>10 \text{ and } z>3 \\ z & \text{if } x+y+z \leq 0 \\ f(x,y+1,z-5)+x & \text{if } x \leq 2 \text{ and } y>10 \text{ and } z>3 \text{ and } x+y+z>0 \\ f(x-1,y-1,z-1)+3 & \text{if } (y \leq 10 \text{ or } z \leq 3) \text{ and } x+y+z>0 \end{cases}$$

a) Implement this function using **guards**. Notice that you can simplify the function. Think about the order and structure of the conditions. Implement a simplified version of the function.

Hint: The keyword `where` might help.

(3 points)

b) Write a function that takes a list of integers and returns a list of integers according to the following definition using **pattern matching**:

$$F(s) = \begin{cases} 42 & \text{if } s = () \\ (f(-5,y,-5)) & \text{if } s = (y) \\ (f(x,5,5))^\frown F(...) & \text{if } s = (x,...) \end{cases}$$

Where $^\frown$ is the list concatenation.

(2 points)

Question 4: Sorting (6 points)

a) Write a function `sorted` that takes a list of comparable values and returns whether the list is sorted or not.

(2 points)

b) Now, implement a function `cocktailsort`, which takes a list of comparable values and returns the sorted list, sorted using *cocktail sort* (`https://en.wikipedia.org/wiki/Cocktail_shaker_sort`).

The algorithm should stop as soon as the list is sorted.

(4 points)

Question 5: Recursion, Map and Fold (7 points)

The function `map` takes an unary function and a list of values and returns a list containing the result of applying the given function to each element in the original list individually.

Similarly, the function `filter` takes a predicate (i.e., a function that returns a Boolean) and a list of values and returns a list only containing the values that satisfy the predicate (i.e., for which the return value of the predicate is `True`).

In this question you will write your own implementations of `map` and `filter` and use them. For obvious reasons, you are not allowed to use the built-in Haskell `map` and `filter` function, but it may be helpful to look up their function signatures with the `:type` command when in the Haskell interpreter.

Hint: Recursion is a commonly used concept in Haskell and may prove very useful in solving the following tasks.

a) Write your own implementation of the map function.

(1 points)

b) Write your own implementation of the filter function.

(1 points)

c) Consider the following list of grades a musician got. The list of hours corresponds to how many hours of work this person put into preparation.

grades: [3.2, 5.6, 5.9, 4.1, 5.7, 1.8, 3.6, 4.1, 5.7, 5.0, 1.3, 4.5, 1.7, 3.7, 4.2, 4.1, 3.2, 5.4, 5.6, 5.6, 1.1, 4.6, 1.1, 3.7, 2.2, 5.2, 2.4, 5.4, 4.6, 3.5, 3.1, 5.3, 2.1, 4.3, 4.6, 5.0, 2.2, 5.7, 1.2, 1.9, 3.8, 3.9, 3.5, 4.4, 4.4, 4.7, 1.8, 4.9, 4.2, 4.9]

hours: [2, 11, 1, 9, 4, 7, 5, 10, 2, 19, 10, 3, 9, 1, 11, 3, 8, 2, 18, 10, 10, 18, 1, 3, 4, 19, 12, 6, 7, 3, 16, 13, 8, 1, 10, 2, 7, 1, 6, 3, 9, 19, 5, 4, 8, 1, 6, 15, 8, 4]

For each grade the musician prepared in another way and is now interested which method was the 'best'. To answer this we will look at the grades-hours ratio, to see how his practice method influences the return in grades.

Write a main function that uses your implementation of the map to calculate the ratio `grades/hours` of these grades-hours pairs. Then use your filter implementation to find the indexes of the times where the practice method did not pay off (ratio < 1), and separately the times where the method yielded a good result (ratio > 4). Print both the list of ratios as well as the two lists of indexes.

Hint: The functions `zip` and `zipWith` might be useful. To retrieve the index of the individuals in the original list it might be a good idea to pair each data point with its index and using list comprehension to extract only the indexes.

(4 points)

d) Write a function that calculates the average value in a list and use it to calculate the average grade, hours and ratio in the given data.

Hint: You might want to use the function `foldl` or `foldr`. Keep the differences of the two functions in mind.

(1 points)

Question 6: Currying (6 points)

In this exercise, you will concern yourself with the idea of *currying*, a central idea in the way Haskell works as a functional programming language.

a) Explain the concept of *currying*.

You should also include in your answer why it is possible to generate a curried version of any function, regardless of it's arity.

(3 points)

b) Explain how *currying* manifests itself in the way functions are declared and defined in Haskell.

Include in your answer what the Prelude functions `curry` and `uncurry` do (you might want to have a look at them before answering the first part of this question, as they could help your understanding).

(3 points)

Question 7: Steganography (8 points)

Steganography (`https://en.wikipedia.org/wiki/Steganography`) is the practice of concealing information within another kind of information. In this question you will write a Haskell program to uncover a single hidden message from the following three integer arrays:

X = [18, 68, 36, 36, 20, 67, 36, 20, 36, 35, 68, 20, 20, 36, 68, 33, 65, 20, 20, 35, 36, 17, 36, 65, 36, 17, 68, 20, 68, 33, 33, 19, 20, 35, 67, 33, 35, 18, 68, 20, 68, 36, 68, 19, 36, 65, 68, 36, 20, 68, 35, 20, 20, 35, 17, 36, 68, 17, 68, 36, 33]

Y = [19, 34, 66, 32, 34, 20, 67, 19, 65, 36, 35, 33, 34, 66, 19, 19, 17, 18, 34, 22, 35, 65, 34, 36, 19, 65, 18, 34, 64, 65, 17, 68, 19, 33, 68, 33, 64, 64, 18, 36, 67, 33, 18, 71, 16, 65, 32, 36, 16, 66, 36, 17, 35, 37, 65, 19, 66, 17, 64, 34, 33]

Z = [34, 65, 32, 67, 20, 66, 33, 66, 35, 18, 65, 16, 17, 32, 64, 36, 66, 33, 16, 35, 70, 18, 32, 35, 17, 66, 65, 16, 33, 34, 18, 67, 18, 36, 64, 34, 66, 66, 16, 33, 70, 20, 65, 20, 33, 34, 65, 64, 65, 20, 20, 32, 16, 65, 18, 33, 33, 66, 35, 17, 19]

The information is hidden as follows: each array $X = [X_1, ..., X_n]$, $Y = [Y_1, ..., Y_n]$ and $Z = [Z_1, ..., Z_n]$ contains information about the hidden message. The arrays are in order, that means X_i, Y_i and Z_i contain information about one part of the message. Find the correct list of integers calculated from X, Y, Z and use the ASCII encoding to generate a string. X, Y and Z encode the solution list $S = [S_1, ..., S_m]$ in base 5 where X encodes the highest digit and Z the lowest. A digit that is higher than the given base is still correct in this encoding. A 7 in the second digit is equivalent to $7 * 5^1$. You additionally get a decipher key that holds further instructions:

- Ignore any bits of X_i, Y_i, Z_i that are higher valued than the 4th bit. That means for example, that for any X_i only the lowest valued four bits $x_{i4}, x_{i3}, x_{i2}, x_{i1}$ matter. As such a 67 corresponds to a 3.
- **Important:** Remove the encodings where the unimportant/throwaway bits of Y_i and Z_i are the same.

Decoding a single character from hand with all these instructions looks like this:

$$X_i = 66, \quad Y_i = 7, \quad Z_i = 17 \quad \rightarrow \quad S_i \; = \; 2 * 5^2 + 7 * 5^1 + 1 * 5^0 \quad \rightarrow \; 'V'$$

a) Write a function `getAsciiChars` that takes a list of integers and returns a list of the corresponding ASCII characters. You might want to use the function `chr` from `Data.Char`.

(2 points)

b) Write a function `uncover` that extracts the hidden information from the three integer lists. This task is easiest with the `(.&.)` (bitwise-and) from `Data.Bits`. The data above should result in a recognizable string.

Hint: You need a bit mask to extract the important bits.

(6 points)