UNIVERSITÄT BASEL

Thorsten Möller
Salome Müller - salo.mueller@stud.unibas.ch
Maarten Schenk - maarten.schenk@unibas.ch

UNI BASEL

Programming Paradigms – Haskell FS 2020

Exercise 4 Due: 17.05.2019 23:59:59

Upload answers to the questions **and source code** before the deadline via courses.cs.unibas.ch. Due to the measures taken to curb the Coronavirus pandemic, programs do **not** have to be demonstrated during the exercise slots. Instead, for **every** task and subtask you **must explain** your solution in detail. If a task involves writing or completing some code, you **must** provide code that is **commented in detail** (how it works, things that need to be done to make it working, things that must be satisfied, and so on).

Also note, of all mandatory exercises given throughout the course, you must score at least 2/3 of the total sum of their points to pass the lecture.

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 (7 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 $0 \leq n \leq 50$ and where $s = n^2$; i.e., the output should be the list `[(0,0),(1,1),(2,4),...,(50,2500)]`.

(1 points)

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

(2 points)

d) 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)

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

(1 points)

f) Return, as a `Boolean`, whether the first and the second last element in a list are 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+1)),(2,1/(1+2)),...,(i,1/(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: Pattern Matching and Guards (5 points)

The Ackermann function is the earliest-discovered example of a total computable function that is not primitive recursive (`https://en.wikipedia.org/wiki/Ackermann_function`).

$$A(m,n) = \begin{cases} n+1 & \text{if } m = 0 \\ A(m-1,1) & \text{if } m > 0 \text{ and } n = 0 \\ A(m-1, A(m,n-1)) & \text{if } m > 0 \text{ and } n > 0 \end{cases}$$

a) Implement the Ackermann function using guards.

(2 points)

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

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

Where $\frown$ is the list concatenation.

(3 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 `mergesort`, which takes a list of comparable values and returns the sorted list, sorted using *merge sort* (`https://en.wikipedia.org/wiki/Merge_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 weights (kg) and heights (cm).

Weights: [85, 67, 105, 61, 98, 56, 60, 58, 77, 94, 97, 52, 56, 91, 68, 69, 89, 70, 57, 108, 97, 91, 89, 59, 82, 76, 70, 88, 66, 69, 98, 60, 90, 72, 63, 108, 53, 73, 51, 103, 83, 83, 79, 72, 99, 93, 64, 62, 86, 95]

Heights: [155, 145, 139, 182, 172, 147, 191, 166, 197, 191, 173, 184, 186, 165, 130, 169, 182, 184, 137, 163, 134, 131, 176, 147, 180, 166, 148, 191, 169, 136, 169, 186, 178, 194, 197, 139, 194, 152, 194, 189, 196, 139, 142, 140, 193, 130, 138, 130, 194, 133]

Write a main function that uses your implementation of the map and filter functions to calculate the BMI of these weight-height pairs and find the indexes of all individuals that suffer from overweight (BMI > 25) and underweight (BMI < 18.5). Print both the list of calculated BMIs as well as the list of indexes of overweight/underweight individuals.

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 height, weight and BMI in the given data.

Hint: You might want to use the function `foldl` or `foldr`.

(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)