UNIVERSITÄT BASEL

Lecturer: Thorsten Möller - **thorsten.moeller@unibas.ch**
Tutors: Andreas Wassmer - **andreas.wassmer@unibas.ch**
Maximilian Barth - **m.barth@unibas.ch**

Programming Paradigms – Haskell FS 2022

Exercise 3 Due: 22.05.2022 23:55:00

Upload your answers to the questions **and source code** on Adam before the deadline.

Text : For answers to questions, observations and explanations, we suggest writing them in LaTeX. Please hand-in your answers as a **single PDF** file (independent of what tools you use, LaTeX, Markdown etc.).

Source-Code : For coding exercises, the source-code must be provided and has to be **commented in detail** (e.g. how it works, how it is executed, comments on conditions to be satisfied).

Upload : Please archive multiple files into a **single compressed zip-file**. If you upload an updated version of your solutions, the file name should contain a clear and intuitive versioning number. Only the latest version will be graded.

Requisit : In order to take the final exam, you must score at least $^2/_3$ of all available points throughout the mandatory exercises.

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)

Write a Haskell function for each of the following tasks.

We want to keep the solutions to these problems as compact as possible; so an additional restriction is that they must not make use of any additional helper functions other than those that are already pre-defined in Haskell.

a) Find the ith element of a list (you may not use `!!`).
You do not have to handle invalid indices.

(1 points)

b) Generate a list of tuples (n, s) where $n_i = 3i$ and $s_i = \sum_{j=1}^{n_i} j$ for $1 \leq i \leq 25$.
(e.g. the output should be the list `[(3,6),(6,21),(9,45),...]`)

(1 points)

c) Remove all instances of a given item from a list.

(1 points)

d) Return a list containg all divisors of a given positive integer.

(1 points)

e) Take two positive integers and return a list containing their shared prime factors.

(2 points)

f) Calculate and return the XOR string for two given binary strings.
(e.g. “0101” XOR “1011” = “1110”)

(1 points)

g) Take a list of integers and return the average value.

(1 points)

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

a) Write a function that returns the infinite list of odd positive integers.
How does Haskell handle infinite lists?

(1 points)

b) Write a function that takes the list from (a) as input and returns a new list containing tuples. Each tuple should contain the original value and its reciprocal.
(e.g. `[1, 3, 5, ...]` $\rightarrow$ `[(1, 1/1), (3, 1/3), (5, 1/5), ...]`)

(1 points)

c) Finally, write a function that takes the infinite list of tuples from (b) and a `Double` as arguments. This function should return the first n entries of the list, whose reciprocals (second value of the tuple) sum up to greater or equal to the given `Double`.
(e.g. `foo tupleLst 1.5` $\rightarrow$ `[(1, 1/1), (3, 1/3), (5, 1/5)]`,
since $\frac{1}{1} + \frac{1}{3} + \frac{1}{5} \geq 1.5$)

(1 points)

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

Question 3: Pattern Matching and Guards (5 points)

The Ackermann function[1] is an example of a total computable function that is not primitive recursive[2]. In this exercise we will use Ackermann's three-argument function:

$$\varphi(m,n,p) = \begin{cases} \varphi(m,n,0) = m+n \\ \varphi(m,0,1) = 0 \\ \varphi(m,0,2) = 1 \\ \varphi(m,0,p) = m, \quad \text{for } p > 2 \\ \varphi(m,n,p) = \varphi(m, \varphi(m, n-1, p), p-1), \quad \text{for } n, p > 0 \end{cases}$$

a) Implement Ackermann's three-argument function using guards.

(2 points)

b) Write the function `ackermannList`. This function should take a list of integers as input and apply the following function to it.

$$F(s) = \begin{cases} 13 & \text{if } s = () \\ (\varphi(x,0,3)) & \text{if } s = (x) \\ (\varphi(x,y,0) & \text{if } s = (x,y) \\ (\varphi(x,y,z) & \text{if } s = (x,y,z) \\ (\varphi(x,y,z))^\frown F(...) & \text{if } s = (x,y,z,...) \end{cases}$$

Where $^\frown$ is the list concatenation and s is the input list.

(3 points)

Question 4: Sorting (6 points)

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

(2 points)

b) Implement the function `mergesort`, which takes a list of comparable values and returns the sorted list. As the name already suggests, you should use **merge sort**[3].

(4 points)

[1]**https://en.wikipedia.org/wiki/Ackermann_function**
[2]**https://en.wikipedia.org/wiki/Primitive_recursive_function**
[3]**https://en.wikipedia.org/wiki/Merge_sort**

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

The function `map` takes an unary function and a list of values as argumnets. The given function is then applied to each element in the list.

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

In the following tasks, 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.

Hints: It might be helpful to look up their function signatures. You can do this with the `:type` command in the Haskell interpreter.

Recursion is a commonly used concept in Haskell and is very useful for 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 lists containing data about Marvel movies [4].

`audienceScore = [91, 70, 71, 76, 75, 91, 78, 75, 92, 92, 83, 85, 89, 86, 87, 87, 87, 79, 91, 81, 45, 90, 95, 91, 98, 78, 98]`

`movieDuration = [126, 112, 124, 113, 124, 143, 130, 111, 135, 121, 141, 117, 146, 115, 135, 133, 130, 134, 149, 118, 124, 181, 129, 133, 133, 157, 148]` (in Minutes)

`boxOffice = [186, 137.5, 170, 150, 140, 225, 200, 150, 170, 170, 365, 130, 250, 165, 200, 175, 180, 200, 300, 130, 175, 400, 160, 200, 150, 200, 200]` (in Millions of USD)

Write a function that calculates the samples mean $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$.
What are the mean values for each of the three lists?

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

(1 points)

[4] **https://www.kaggle.com/datasets/davidgdong/marvel-cinematic-universe-box-office-dataset**

d) Write a function that takes two lists and calculates the Pearson correlation coefficient[5]:

Given paired data $\{(x_1, y_1), ..., (x_n, y_n)\}$ consisting of n pairs, r_{xy} is defined as:

$$r_{xy} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

Use the sample mean function that you wrote in the previous sub-task in order to calculate $\bar{x}$ and $\bar{y}$.
Which two data sets have the highest Pearson correlation coefficient?

Hint: The functions `zip` and `zipWith` might be useful.

(4 points)

Question 6: Currying (6 points)

In the following tasks you are asked to answer questions about *currying*, a central concept of Haskell and other functional programming languages.

a) Explain the concept of *currying*. Is it possible to generate a curried version of any function? What are the benefits of currying a function?

(3 points)

b) Explain what the following Haskell code does. How is this related to *currying*?

```
foo = \x -> (\y -> x * y)
```

(2 points)

c) Provide the curried version of the following function:

```
foo :: (Bool, Int, Int) -> Int
foo (x, y, z)
  | x = y + z
  | otherwise = y - z
```

(1 points)

[5] https://en.wikipedia.org/wiki/Pearson_correlation_coefficient

Question 7: Steganography (8 points)

Steganography (**https://en.wikipedia.org/wiki/Steganography**) is the practice of concealing information within another kind of information. For this task you will write a Haskell program to uncover the hidden message from the following three integer lists:

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 list $X = [X_1, ..., X_n]$, $Y = [Y_1, ..., Y_n]$ and $Z = [Z_1, ..., Z_n]$ contains information about the hidden message. The arrays are in order, meaning that X_i, Y_i and Z_i contain information about the same 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 have a higher value than the 4th bit. For example: For any X_i only the value of he lowest four bits $x_{i4}, x_{i3}, x_{i2}, x_{i1}$ matter. This means that 67 corresponds to a 3.
- **Important:** Remove the encoding where the unimportant/throwaway bits of Y_i and Z_i are the same.

Decoding a single character by hand would look 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 the function `getAsciiChars` that takes a list of integers and returns a list of the corresponding ASCII characters. You may want to use the function `chr` from `Data.Char`.

(2 points)

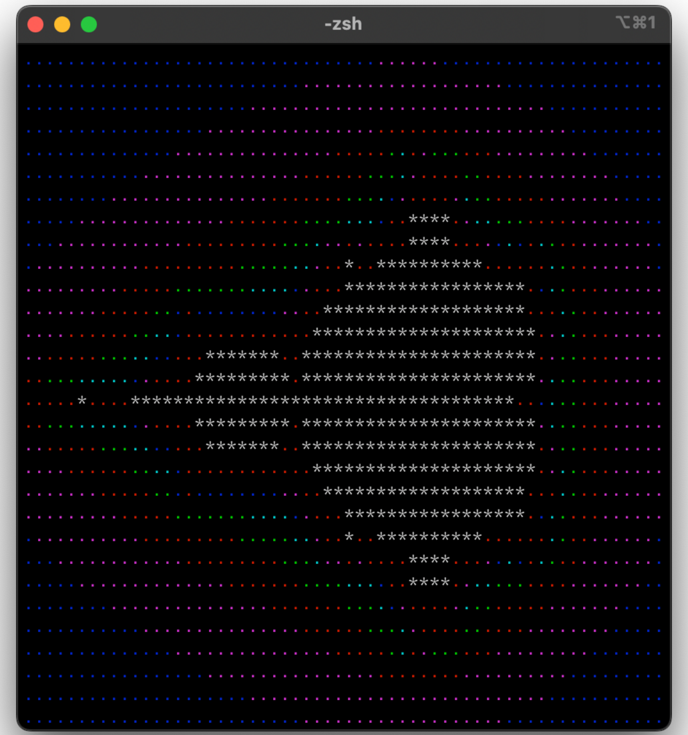
b) Write the function `uncover` that extracts the hidden information from the three integer lists. Use the bitwise-and `(.&.)` from `Data.bits`. The resulting string should make sense.

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

(6 points)

Question 8: Mandelbrot set (7 points)

Implement a Haskell program that computes the *Mandelbrot set*[6] over a range of complex numbers and print a visualization to the console. This will result in an ASCII art pattern as shown in Figure 1. Color the numbers that are not part of the set by counting how many iterations it takes for each of them to 'escape' (more details below).



```
............................................................
............................................................
............................................................
............................................................
............................................................
............................................................
............................................................
....................................****....................
....................................****....................
..............................*..***********................
..............................*****************.............
............................*******************.............
...........................*********************............
.................*******..**********************............
................*********.**********************............
.....*....************************************..............
................*********.**********************............
.................*******..**********************............
...........................*********************............
............................*******************.............
..............................*****************.............
..............................*..***********................
....................................****....................
....................................****....................
............................................................
............................................................
............................................................
............................................................
............................................................
............................................................
```

Figure 1: Mandelbrot set

Use the following definition to determine whether a given complex number is part of the Mandelbrot set:

$$z_{n+1} = z_n^2 + c$$

A complex number c is part of the Mandelbrot set if the absolute value of z_n remains bounded $\forall n > 0$, when starting with $z_0 = 0$. You can do this by checking if $|z_{100}| < 10$.

Draw the set from $-2 \leq x \leq 1$ with a step size of 0.05 and from $-1.5 \leq y \leq 1.5$ with a step size of 0.1. Transform each point (x, y) to $z_0 = x + yi$ and check whether it belongs to the Mandelbrot set (*) or if it escapes (.). Remember that the origin of the coordinate system should be in the center of the terminal.

[6]**https://en.wikipedia.org/wiki/Mandelbrot_set**

a) Write a function that prints the Mandelbrot set (as described above) to the console. **Hint:** The function `putStrLn`[7] might be useful.

(4 points)

b) Write a function that determines how many iterations it takes for a given complex number (that is not part of the Mandelbrot set) to escape. Do this by counting the number of iterations n until $|z_n| > 10^3$. Finally, use this function to add colors to your plot. You can color a string `s` by wrapping it as follows (Python code):

```
s = '\x1b[31m' + s + '\x1b[0m'
```

Where `31` defines the color. More available colors are:

color	number
black	30
red	31
green	32
yellow	33
blue	34
mangenta	35
cyan	36
white	37

E.g. `'\x1b[35m'` is mangenta.

We used the following mappings for Figure 1 (you may also use your own):

num iterations	color
4	blue
5	mangenta
6	red
7	green
8	cyan
9	blue
10	mangenta
otherwise	red

(3 points)

[7] **https://wiki.haskell.org/Introduction_to_Haskell_IO/Actions**