Prof. Dr. J. Giesl M. Hark

Notes:

- To solve the programming exercises you should use the Glasgow Haskell Compiler **GHC**, available for free at `https://www.haskell.org/ghc/`. You can use the command "`ghci`" to start an interactive interpreter shell.
- Please solve these exercises in **groups of four**!
- The solutions must be handed in **directly before (very latest: at the beginning of)** the exercise course on Wednesday, 15.05.2019, 14:30, in lecture hall **AH I**. Alternatively you can drop your solutions into a box which is located right next to Prof. Giesl's office (until 30 minutes before the exercise course starts). Also, please **print** the source code of your solutions for the programming exercises.
- In addition, please upload the source code of your solutions for the programming exercises in a single **ZIP**-archive via `RWTHmoodle` before the exercise course on Wednesday, 15.05.2019, 14:30. Please name your archive `Sheet_i_Mat1_Mat2_Mat3_Mat4.zip`, where `i` is the number of the sheet and `Mat_1...Mat_4` are the immatriculation numbers of the group members. It is sufficient if **one** of the group members uploads your solution. Files, which are not accepted by **GHCi**, will not be marked.
- Please write the **names** and **immatriculation numbers** of all students on your solution. Also please staple the individual sheets!

In all exercises, if *not stated otherwise*, also give the type declarations of the functions to implement. Moreover, if *not stated otherwise*, you can write auxiliary functions and always use functions from previous parts of the exercise, even if you did not manage to solve them.

Programming Exercise 1 (Lazy Evaluation): (2 + 2 = 4 points)

In this exercise we will consider the "Collatz conjecture"[1] from mathematics. In this exercise you are allowed to use any predefined function from Haskell's module `Prelude`.

Let the function $f : \mathbb{N} \to \mathbb{N}$ be defined as $n \mapsto \begin{cases} \frac{n}{2}, & n \mod 2 = 0 \\ 3 \cdot n + 1, & n \mod 2 = 1 \end{cases}$. For any $n, k \in \mathbb{N}$, let $f^k(n)$ denote $\underbrace{f(\ldots f}_{k \text{ times}}(n)\ldots)$. The *Collatz total stopping time* of a positive natural number n is defined to be the smallest $k \in \mathbb{N}$ such that $f^k(n) = 1$ and ∞ otherwise. For example, the Collatz total stopping time of 1 is 3 (since $f(1) = 4$, $f^2(1) = 2$, $f^3(1) = 1$) and the Collatz total stopping time of 3 is 7. The Collatz conjecture states that any positive natural number has a finite Collatz total stopping time. This conjecture is a famous open problem.

a) Implement a function `collatz :: Int -> [Int]` such that for any positive natural number `n` the expression `collatz n` is the infinite list with `k`th entry f^k(`n`). This function may behave arbitrarily on non-positive inputs.

Furthermore implement a function `total_stopping_time :: Int -> Int`. For any positive natural number `n` the function computes its Collatz total stopping time. This function may behave arbitrarily on non-positive inputs.

Hints:

- The predefined function `takeWhile :: (a -> Bool) -> [a] -> [a]` computes the shortest prefix of the list where the first argument of `takeWhile` is true for every list element.
- The predefined function `div :: Int -> Int -> Int` computes integer division, the predefined function `mod :: Int -> Int -> Int` implements the modulo operation from mathematics.

[1] `https://en.wikipedia.org/wiki/Collatz_conjecture`

b) Implement a function `check_collatz :: Int -> Bool`. For any positive natural number `n` the function returns `True` if the Collatz conjecture holds for the first `n` positive natural numbers. If the input `n` is not positive or the conjecture does not hold for the first `n` positive numbers, then the function may behave arbitrarily.

Programming Exercise 2 (List Comprehensions): (2 + 2 = 4 points)

In this exercise you *may not* write auxiliary functions and you *may not* use `where` or `let`.

a) Consider the following definition for an infinite list containing all primes from the lecture:

```
drop_mult :: Int -> [Int] -> [Int]
drop_mult x xs = [y | y <- xs, y `mod` x /= 0]

dropall :: [Int] -> [Int]
dropall (x:xs) = x : dropall (drop_mult x xs)

primes :: [Int]
primes = dropall [2 ..]
```

Write a function `goldbach :: Int -> [(Int,Int)]`[2] that returns a list of pairs indicating all possibilities to write a positive number as a sum of two odd primes. Each possibility up to permutation has to occur exactly once but the order of the pairs in the resulting list is irrelevant. For example, `goldbach 50 == [(3,47),(7,43),(13,37),(19,31)]`. Of course, `goldbach n == []`, whenever `n` is odd and positive! For non-positive numbers, the function may behave arbitrarily. A famous open problem is whether there exists an even positive number n with `goldbach n == []`.

You may only use *one* defining equation and the right-hand side *must* be a single list comprehension.

Hints:

- You may use the given functions `primes`, `drop_mult`, and `dropall` in addition to predefined functions in Haskell's module `Prelude` .
- Only test your implementation on relatively small numbers! Our implementation of `primes` is not the most efficient one.

b) Implement a function `range :: [a] -> Int -> Int -> [a]`. The expression `range xs m n` should yield the sublist of `xs` starting from the first element with an index `>= m` and ending with the last entry at an index `<= n`. The first entry has index `0`, the last one has `(length xs) - 1`. For example, `range [3,4,5] 1 2 == [4,5]`, `range [3,4,5] (-7) 2 == [3,4,5]` and `range [3,4,5] 10 7 == []`.

You may only use *one* defining equation and the right-hand side *must* be a single list comprehension.

- You may use any predefined functions in Haskell's module `Prelude` *except* `take`, `drop`, `span`, `break`, `splitAt`, or variants[3] of these functions.

Programming Exercise 3 (IO): (2 + 2 + 5 + 1 = 10 points)

In this exercise an interactive program simulating your personal library should be implemented. Consider the following example run. Here, inputs by the user are printed in **bold**.

```
*Main> main
Welcome to your Library
Would you like to put back or take a book?
 Enter Book: Title's name; Author's name
```

[2] https://en.wikipedia.org/wiki/Goldbach's_conjecture
[3] like `takeWhile`, etc.

```
Are you looking for an author?
 Enter Author: Author's name
Are you looking for a special book?
 Enter Title: Title's name.
>
```

df

```
There has been an error: df
Would you like to put back or take a book?
 Enter Book: Title's name; Author's name
Are you looking for an author?
 Enter Author: Author's name
Are you looking for a special book?
 Enter Title: Title's name.
>
```

Book: Harry Potter and the Philosopher's Stone; Joanne K. Rowling

```
Do you want to (p)ut the book back or do you want to (t)ake the book?
```

p

```
Done!
Would you like to put back or take a book?
 Enter Book: Title's name; Author's name
Are you looking for an author?
 Enter Author: Author's name
Are you looking for a special book?
 Enter Title: Title's name.
>
```

Book: Harry Potter and the Prisoner of Azkaban; Joanne K. Rowling

```
Do you want to (p)ut the book back or do you want to (t)ake the book?
```

z

```
Wrong input!
Would you like to put back or take a book?
 Enter Book: Title's name; Author's name
Are you looking for an author?
 Enter Author: Author's name
Are you looking for a special book?
 Enter Title: Title's name.
>
```

Book: Harry Potter and the Prisoner of Azkaban; Joanne K. Rowling

```
Do you want to (p)ut the book back or do you want to (t)ake the book?
```

p

```
Done!
Would you like to put back or take a book?
 Enter Book: Title's name; Author's name
Are you looking for an author?
 Enter Author: Author's name
Are you looking for a special book?
 Enter Title: Title's name.
>
```

Author: Joanne K. Rowling

```
You have the following books from Joanne K. Rowling
Book: Harry Potter and the Prisoner of Azkaban; Joanne K. Rowling,
 Book: Harry Potter and the Philosopher's Stone; Joanne K. Rowling,
Would you like to put back or take a book?
 Enter Book: Title's name; Author's name
Are you looking for an author?
 Enter Author: Author's name
Are you looking for a special book?
 Enter Title: Title's name.
>
```

Book: Harry Potter and the Prisoner of Azkaban; Joanne K. Rowling

```
Do you want to (p)ut the book back or do you want to (t)ake the book?
```

t

```
Done!
Would you like to put back or take a book?
 Enter Book: Title's name; Author's name
Are you looking for an author?
 Enter Author: Author's name
Are you looking for a special book?
 Enter Title: Title's name.
>
```

Author: Joanne K. Rowling

```
You have the following books from Joanne K. Rowling
Book: Harry Potter and the Philosopher's Stone; Joanne K. Rowling,
Would you like to put back or take a book?
 Enter Book: Title's name; Author's name
Are you looking for an author?
 Enter Author: Author's name
Are you looking for a special book?
 Enter Title: Title's name.
>
```

Title: Harry Potter and the Philosopher's Stone

```
You have the following books with the title: Harry Potter and the Philosopher's Stone
Book: Harry Potter and the Philosopher's Stone; Joanne K. Rowling,
Would you like to put back or take a book?
 Enter Book: Title's name; Author's name
Are you looking for an author?
 Enter Author: Author's name
Are you looking for a special book?
 Enter Title: Title's name.
>
```

Book: Harry Potter and the Chamber of Secrets; Joanne K. Rowling

```
Do you want to (p)ut the book back or do you want to (t)ake the book?
```

t

```
You do not have this book!
Would you like to put back or take a book?
 Enter Book: Title's name; Author's name
Are you looking for an author?
 Enter Author: Author's name
Are you looking for a special book?
 Enter Title: Title's name.
>
```

Exit

```
Bye!
*Main>
```

The library has a list `books` in which the books of the library are stored. A book is represented by a tuple `(title,author)::(String,String)`. After displaying the welcome message, the library is still empty. Then an interactive loop starts where the user is prompted for input and the library reacts to it. If the user

- enters `Book:title;author`, the user can choose to either put the book into the library (by entering the character `'p'`) or take it from the library (by entering the character `'t'`). If the book is to be taken but not contained in the library, an error message should be stated before starting from the beginning.
- enters `Author:author`, all the books from this author are printed on the screen. Then the loop starts again.
- enters `Title:title`, all the books with this title are printed on the screen. Then the loop starts again.
- enters `Exit`, the loop terminates.
- enters something else, an error message is shown and the loop starts again.

After the interactive loop terminates, a goodbye message is displayed and the whole program terminates.
A framework for the implementation is given in the file `library.hs`. Only edit the parts between the comments `replace with implementation:` and `end replace`. Do not change any other code.

Hints:

To output a line of text use `putStrLn :: String -> IO ()`, to read a line use `getLine :: IO String`. Furthermore you may use `elem :: Eq a => a -> [a] -> Bool` for checking membership in a list.

a) Implement the function `main` that first displays a welcome message, then evaluates `library []` and then displays a goodbye message.

b) Implement the function `getInput` that displays a prompt `>`, then reads a line from standard input, and finally returns a `LibraryInput` computed from the just-read string using the given function `parseLibraryInput`.

c) Implement the function `library`. It should first ask the user for input using `getInput` and then take an appropriate action, depending on the input as described above.

d) Is it possible in Haskell to write a function `main' :: Int` that behaves similar to the function `main` but returns the number of books stored in the library when the loop was exited? Explain your answer either on the paper you hand in or as a comment in your solution of the programming exercises. You must not refer to predefined Haskell functions which are not included in the module `Prelude`.

Exercise 4 (Definedness): (3 + 4 = 7 points)

a) Consider the following values of the domain $(\mathbb{Z}_\perp \times \mathbb{B}_\perp) \times \mathbb{Z}_\perp$:

- $y_1 = ((-1, \texttt{False}), 0)$
- $y_2 = ((-1, \perp), 2)$

Find all elements in the domain that are less defined than one of the values above, i.e., all x such that $x \sqsubseteq y_i \wedge x \neq y_i$ for some $i \in \{1, 2\}$.

Draw a directed graph whose nodes are labeled with y_1, y_2 and all these values x. Add arrows in such a way that there is a path from x' to y' if and only if x' is less defined than y'.

b) Consider the domain $D = \underbrace{\mathbb{Z}_\perp \times \cdots \times \mathbb{Z}_\perp}_{n \text{ times}}$ for $0 < n \in \mathbb{N}$. A chain $S \subseteq D$ is a totally ordered subset of D, i.e. if $x \neq y \in S$ then either $x \sqsubseteq y$ or $y \sqsubseteq x$. Determine

$$\sup\{|S| \mid S \subseteq D, S \text{ is a chain}\},$$

where $|S|$ is the number of elements in $|S|$. In other words, what is the maximal number of elements a chain in D can have? Please prove the correctness of your answer.