Functional Programming

Excercise Sheet 3

Emilie Hastrup-Kiil (379455), Julian Schacht (402403),
Niklas Gruhn (389343), Maximilian Loose (402372)

Excercise 1

a)

```
collatz :: Int -> [Int]
collatz x = iterate (\y -> if mod y 2 == 0 then div y 2 else 3*y+1) x


total_stopping_time :: Int -> Int
total_stopping_time 1 = 3
total_stopping_time x = length (takeWhile (/= 1) (collatz x))
```

b)

```
check_collatz :: Int -> Bool
check_collatz 1 = True
check_collatz n = (total_stopping_time n) <maxBound && check_collatz (n
   -1)
```

Excercise 2

a)

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

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

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

goldbach :: Int -> [(Int,Int)]
goldbach n = [(x,y)| not(odd n), x<-takeWhile (<n) primes, y<-filter
   (>=x) (takeWhile (<n) primes), odd x, odd y, x+y == n]
```

b)

```
range :: [a] -> Int -> Int -> [a]
range xs m n = [ x | (i, x) <- zip [0 ..] xs, i >= m, i <= n b]
```

Excercise 3

a)

```
main :: IO ()
main = do
  -- task a)
  putStrLn "Welcome to your library"
  library []
```

```
  putStrLn "Bye!"
  return ()
```

b)

```
getInput :: IO LibraryInput
getInput = do
  -- task b)
  putStrLn "Would you like to put back or take a book?\n Enter Book:
     Title's name; Author's name \nAre you looking for an author?\n
     Enter Author: Author's name \nAre you looking for a special book
     ?\n Enter Title: Title's name."
  putChar '>'
  input <- getLine
  return (parseLibraryInput input)
  return Exit
```

c)

```
library ::[(String,String)] -> IO ()
library books =  do
  -- task c)
  input <- getInput
  getLibraryAction books input
  return ()
```

```
getLibraryAction :: [(String,String)] -> LibraryInput -> IO()
getLibraryAction books Exit = do return ()
getLibraryAction books (Error e) = do putStrLn (show (Error e))
                library books
getLibraryAction books (Book (t,a)) = do putStrLn (show (Book (t,a)))
                putStrLn "Do you want to (p)ut the book back or do you
                   want to (t)ake the book?"
                input <- getLine
                evaluateAction input books (t,a)
getLibraryAction books (Author a)  = do putStrLn (show (Author a))
                putStrLn ("You have the following books from " ++ a)
                displayBooks_A a books
                library books
getLibraryAction books (Title t) = do putStrLn (show (Title t))
                putStrLn ("You have the following books with the title:
                    " ++ t)
                displayBooks_T t books
                library books
```

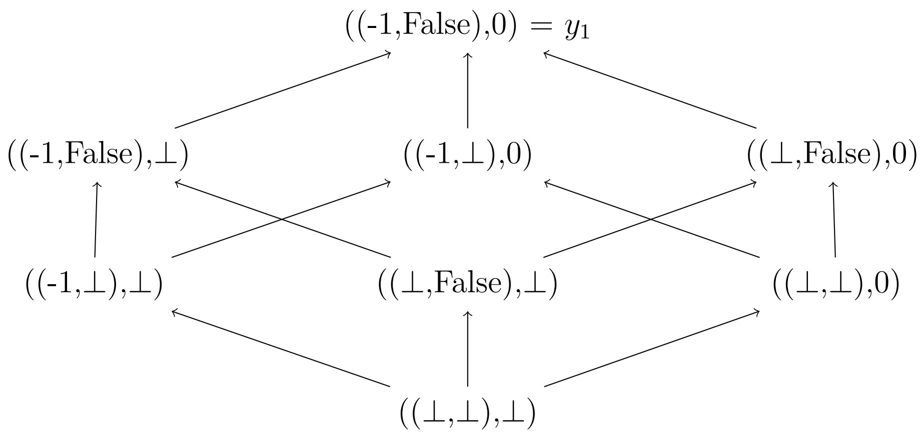
```
displayBooks_A :: String -> [(String,String)] -> IO()
displayBooks_A a [] = return()
displayBooks_A a ((title, author) :books) = if (a==author) then
   putStrLn (show (Book (title,author))) >> displayBooks_A a books
   else displayBooks_A a books

displayBooks_T :: String -> [(String,String)] -> IO()
displayBooks_T t [] = return()
displayBooks_T t ((title, author) :books) = if (t==title) then putStrLn
    (show (Book (title,author))) >> displayBooks_T t books else
   displayBooks_T t books
```

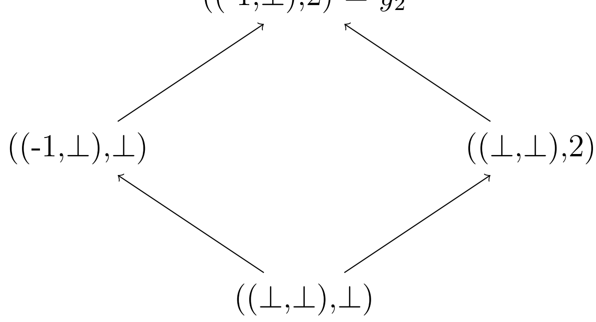
```
evaluateAction :: String -> [(String,String)] -> (String,String) -> IO
    ()
evaluateAction "t" books b  =  if (elem b books) then putStrLn "Done!"
   >> library (filter (/=b) books) else putStrLn "You do not have this
    book!" >> library books
evaluateAction "p" books b  =  putStrLn "Done!" >> library (b:books)
evaluateAction _   books _  =  putStrLn "Wrong input!" >> library books
```

d)
No, it is not possible to write a function `main :: Int` that behaves similar to the function `main :: IO ()` of the previous excercises. The number of books in the library is a value, which is encapsulated in an IO action. This value cannot be taken out of this action, since that would be against referential transparency. Furthermore, having the function `main` be of type `Int` would also consequently change the behaviour of the function as IO actions such as `putStrLn :: String -> IO ()` or let alone `>> :: IO () -> IO () -> IO()` would cause a type matching error.

Excercise 4
a)



All elements $x \in ((\mathbb{Z}_\perp \times \mathbb{B}_\perp) \times \mathbb{Z}_\perp$ that are less defined than y_1 are given by $((-1,\perp),0),((\perp,False),0),((-1,False),\perp),((\perp,False),\perp),((-1,\perp),\perp),((\perp,\perp),0)$ and the smallest Element $((\perp,\perp),\perp)$.



All elements $x \in ((\mathbb{Z}_\perp \times \mathbb{B}_\perp) \times \mathbb{Z}_\perp$ that are less defined than y_2 are given by $((\perp, \perp), 2), ((-1, \perp), \perp)$ and the smallest Element $((\perp, \perp), \perp)$.

b)
Given a domain $D = \underbrace{\mathbb{Z}_\perp \times ... \times \mathbb{Z}_\perp}_{n \text{ times}}$ for $0 < n \in \mathbb{N}$ and a chain $S \subseteq D$ then $\sup\{|S| \mid S \subseteq D, \text{S is a chain}\} = n + 1$.

We use induction on n to show that this holds for all $n \in \mathbb{N}, n > 0$.

Base case $n = 1$
Let $D = \mathbb{Z}_\perp$, then $\perp \in D$ and for all $x \in \mathbb{Z} : x \in \mathbb{Z}_\perp$. For $x, y \in S$ with $x, y \in \mathbb{Z}$ it follows from the definition of S that either $x \sqsubseteq y$ or $y \sqsubseteq x$. By the definition of $\sqsubseteq$ and x and y, this is only true iff $x = y$. As a result, $|\{x \in \mathbb{Z} \mid x \in S \text{ with } S \subseteq D, \text{S is a chain}\}| \leq 1$.
However, $\perp \sqsubseteq x$ for all $x \in \mathbb{Z}$ hence $\sup\{|S| \mid S \subseteq D, \text{S is a chain}\} = 1 + 1 = 2 = n + 1$.

Induction hypothesis
$\sup\{|S| \mid S \subseteq D, \text{S is a chain}\} = n+1$ holds for $n \in \mathbb{N}, 0 < n$ with $D = \underbrace{\mathbb{Z}_\perp \times ... \times \mathbb{Z}_\perp}_{n \text{ times}}$.

Induction step $n \to n + 1$
Let $D = \underbrace{\mathbb{Z}_\perp \times ... \times \mathbb{Z}_\perp}_{n+1 \text{ times}}$.
It follows from the hypothesis that if $D_n = \underbrace{\mathbb{Z}_\perp \times ... \times \mathbb{Z}_\perp}_{n \text{ times}}$ then $\sup\{|S_n| \mid S_n \subseteq D_n, S_n \text{ is a chain}\} = n + 1$.

Now we extend each n-tuple $(n_1, n_2, ..., n_n)$ in the chain S_n $(n_1, n_2, ..., n_n \in \mathbb{Z}_\perp)$ so that we get (n+1)-tuples $(n_1, n_2, ..., n_n, x) \in D, x \in \mathbb{Z}_\perp$.

Case 1: $x \in \mathbb{Z}$
For two tuples $t_i = (n_{i_1}, n_{i_2}, ..., n_{i_n}), t_j = (n_{j_1}, n_{j_2}, ..., n_{j_n}) \in S_n$ either $t_i \sqsubseteq t_j$ or $t_j \sqsubseteq t_i$ applies. Then also $(n_{i_1}, n_{i_2}, ..., n_{i_n}, x) \sqsubseteq (n_{j_1}, n_{j_2}, ..., n_{j_n}, x)$ or $(n_{j_1}, n_{j_2}, ..., n_{j_n}, x) \sqsubseteq (n_{i_1}, n_{i_2}, ..., n_{i_n}, x)$ since in particular $x \sqsubseteq x$ and consequently $(n_{j_1}, n_{j_2}, ..., n_{j_n}, x)$, $(n_{i_1}, n_{i_2}, ..., n_{i_n}, x) \in S$ with $\sup\{|S| \mid S \subseteq D, \text{S is a chain}\} \geq \sup\{|S_n| \mid S_n \subseteq D_n, S_n \text{ is a chain}\} = n + 1$.
If $(n_1, n_2, ..., n_n, x) \in S$ and $(n_1, n_2, ..., n_n, y) \in S$ then $x = y$ because neither $x \sqsubseteq y$ nor $y \sqsubseteq x$ applies. Similarily, If $t_i = (n_1, n_2, ..., n_n, x) \in S$ and $t_j = (n_1, n_2, ..., x, ..., n_n) \in S$ then neither $t_i \sqsubseteq t_j$ nor $t_j \sqsubseteq t_i$ applies, because $t_i \neq t_j$ and both are equally defined. This leads on to our second case.

Case 2: $(n_1, n_2, ..., n_n, x), n_1 = n_2 = ... = n_n = x = \perp$
Then clearly $(n_1, n_2, ..., n_n, x) \in S$ since it is the smallest element of D and therefore $(n_1, n_2, ..., n_n, x) \sqsubseteq y$ for all $y \in D$. Resulting from this, $\sup\{|S| \mid S \subseteq D, \text{S is a chain}\} = (n + 1) + 1 = n + 2$. The equation holds. $\square$