

Exam in Functional Programming SuSe 19

First Name: _____

Last Name: _____

Matriculation Number: _____

Course of Studies (please mark **exactly** one):

- | | |
|--|--|
| <ul style="list-style-type: none"> <input type="radio"/> Informatik Bachelor <input type="radio"/> Informatik Master <input type="radio"/> Other: _____ | <ul style="list-style-type: none"> <input type="radio"/> Mathematik Master <input type="radio"/> Software Systems Engineering Master |
|--|--|

	Available Points	Achieved Points
Exercise 1	40	
Exercise 2	40	
Exercise 3	22	
Exercise 4	18	
Sum	120	

Notes:

- **On all sheets** (including additional sheets) you must write **your first name, your last name, and your matriculation number**.
- Give your answers in readable and understandable form in either English or German.
- Use **permanent** pens. Do not use red or green pens and do not use pencils.
- Please write your answers on the exam sheets (also use the reverse sides).
- For each exercise part, give **at most one** solution. Cancel out everything else. Otherwise all solutions of the particular exercise part will be evaluated with **0 points**.
- If we observe any **attempt of deception**, the whole exam will be evaluated to **0 points**.
- At the end of the exam, hand in **all sheets together with the sheets containing the exam questions**.

I hereby confirm that I feel healthy
and able to participate in the exam.

Signature

Exercise 1 (Programming in Haskell):

(5 + 12 + 9 + 14 = 40 points)

We use the following data structure `Tree a` to represent binary trees in Haskell.

```
data Tree a = E | N a (Tree a) (Tree a)
```

For example, the tree from Fig. 1 can be represented by:

```
exTree :: Tree Int
exTree = N 16 (N 37 (N 19 E E) (N 21 E E)) (N 25 (N 2 E E) (N 12 E E))
```

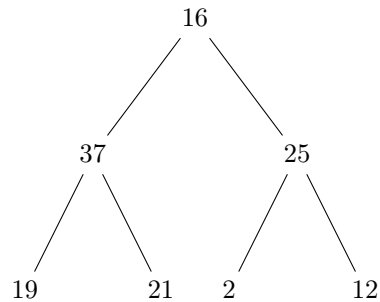


Figure 1: Tree of type `Tree Int`

In each of the following exercises you can use functions from the `Haskell Prelude` and from previous exercises even if you did not implement them. Moreover, you can always implement auxiliary functions.

- a) Declare `Tree a` as an instance of the type class `Ord` whenever `a` is an instance of `Ord`. In this instance declaration, you should provide an implementation of the function `<=` to compare two trees. For two non-empty trees `t1` and `t2`, we have `t1 <= t2` if and only if the value of `t1`'s root node is smaller or equal to the value of `t2`'s root node. For example, `(exTree <= N 19 E E) == True` but `(N 19 E E <= exTree) == False`, as `(19 <= 16) == False`. Furthermore, `(t <= E) == True` for any `t :: Tree a` but `(E <= t) == True` if and only if `t == E`.

Hints:

- You can assume that `Tree a` is an instance of `Eq` whenever `a` is an instance of `Eq`.

- b) A *heap* is a binary tree in which a value stored at a node is greater or equal to all values stored in its children. For example, the tree in Fig. 1 is not a heap, as $(16 \geq 37) == \text{False}$. However, the subtree of the tree in Fig. 1 with root 25 is a heap, as $(25 \geq 2) == \text{True}$ and $(25 \geq 12) == \text{True}$. Note that the root node of a heap contains the *maximal* value stored in the heap. The empty tree *E* is also a heap.

Write a Haskell-function `heapify` and also give its type declaration. Its only argument is an element `t :: Tree a` for a type `a` of the type class `Ord`. The call `heapify t` for some `t :: Tree a` results in a tree `h :: Tree a` which stores exactly the same values as `t` but is also a heap. If a value is stored n times in `t`, then it is also stored n times in `heapify t`.

For a tree `N v l r`, your implementation should check if `v` is greater or equal to the values stored in `l` and `r`, respectively. If yes, the result is `N v lh rh`, where `lh` is `l` transformed into a heap and `rh` is `r` transformed into a heap. If no, then swap `v` and the maximal value of the tree. Afterwards, call `heapify` on the subtree where `v` was swapped to in order to ensure that the result is a heap.

For example, `heapify exTree` results in the tree in Fig. 2.

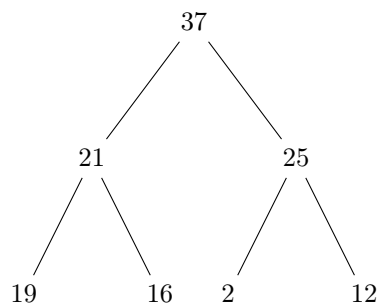


Figure 2: Heapified version of Fig. 1

Name:

Matriculation Number:

- c) For the following exercise, assume that there is a function `deleteRoot :: Ord a => Tree a -> Tree a`. For a given heap `h` with `h /= E`, the call `deleteRoot h` results in a heap storing exactly the same values as `h` *without* its root node. Again, if a value is stored n times in `h` without its root node, then the value is also stored n times in `deleteRoot h`. Furthermore, assume that there is a function `insert :: a -> Tree a -> Tree a` that inserts a value into a tree.

Write a Haskell-function `heapsort` and also give its type declaration. Its only argument is a list `xs :: [a]` for a type `a` of the type class `Ord`. The function `heapsort` sorts `xs` in *decreasing* order by first transforming it into a heap. Then, it extracts the maximal value of the heap by using `deleteRoot` and adds this value to the resulting list. This is repeated until the heap is empty.

Hints:

- You can use the functions `heapify`, `insert`, and `foldr` to transform a list into a heap.

- d) Implement a function `participants :: [(String, String)] -> IO ()`. This function simulates an interactive system for managing participants encoded as pairs (`surname`, `first_name`).

When `participants xs` is called, it asks the user for an input. If the input is

- "get", then the user can enter a value `k :: Int` and gets the k th name when sorting the participants w.r.t. surnames in decreasing order. Participants with the same surnames are sorted by their first names. The first participant has the index 0. If there are less than $k+1$ participants, an error message is prompted. Then `participants` calls itself again with the same argument.
- "add", then the user enters first the surname and then the first name of the participant to be added. Then `participants` prompts a success message and calls itself again with the list resulting from `xs` when adding the entered participant.
- In any other case the program terminates.

An example run should look as follows. Here, inputs of the user are written in *italics*.

```
*Main> participants []
Add participant or get kth?
add
Giesl
Juergen
Added participant
Add participant or get kth?
add
Mueller
Julia
Added participant
Add participant or get kth?
```

Name:

Matriculation Number:

```
get
```

```
2
```

```
Not enough participants
```

```
Add participant or get kth?
```

```
get
```

```
1
```

```
Giesl, Juergen
```

```
Add participant or get kth?
```

```
Stop
```

```
*Main>
```

Hints:

- The function `putStrLn::String -> IO()` prints a string and performs a line break.
- The function `getLine::IO String` reads a string from the keyboard.
- You can assume that there is a function `getInt::IO Int` that reads an integer from the keyboard.
- You can use the indexing operator `(!!)::[a] -> Int -> a`. Here, `xs!!k` gives the `k`th element of the list `xs` whenever `k` is between 0 and `(length xs)-1`.
- You can use the function `heapsort` to sort the list of participants in decreasing order. The reason is that for two pairs `(s1,s2),(t1,t2)::(String,String)` we have `(s1,s2) <= (t1,t2)` iff `s1 <= t1` or `s1 == t1` and `s2 <= t2`. Strings are compared as usual by `<=`.

Exercise 2 (Semantics):
 $((10 + 7) + (5 + 7 + 4) + 7 = 40$ points)

- a) i) Prove the following generalization of the Fixpoint Theorem. Let D be a domain, \sqsubseteq a complete partial order on D , $f : D \rightarrow D$ a continuous function w.r.t. \sqsubseteq , and $d \in D$. If $d \sqsubseteq f(d)$ then $p^* = \sqcup \{f^n(d) \mid n \in \mathbb{N}\}$ satisfies the following:

- The element p^* is a fixpoint of f .
- For every fixpoint p of f with $d \sqsubseteq p$ we have $p^* \sqsubseteq p$.

Hints:

- You do *not* have to show that $\sqcup \{f^n(d) \mid n \in \mathbb{N}\}$ exists, i.e., that $\{f^n(d) \mid n \in \mathbb{N}\}$ is indeed a chain.

Name:

Matriculation Number:

- ii) Let D be a domain, \sqsubseteq a complete partial order on D , and $f : D \rightarrow D$ a continuous function w.r.t. \sqsubseteq . By the Fixpoint Theorem, f has a least fixpoint $\text{lfp } f$. Disprove the following claim by giving a *counterexample*:

If $d \in D$ with $d \sqsubseteq f(d)$ then $d \sqsubseteq \text{lfp } f$.

Hints:

- You also have to show that your counterexample meets the requirements, i.e., that the chosen order is complete, that your function f is continuous, and that your chosen $d \in D$ satisfies $d \sqsubseteq f(d)$.
- You may use that any *flat* domain with the order $x \sqsubseteq y$ iff $x = y \vee x = \perp$ is complete.
- On *flat* domains, continuity and monotonicity are equivalent.

- b) i) Consider the following Haskell function f :

```
f :: (Int, Int) -> Int
f (n, 0) = 1
f (0, k) = 0
f (n, k) = f(n-1, k) + f(n-1, k-1)
```

Please give the Haskell declaration for the higher-order function \mathbf{ff} corresponding to f , i.e., the higher-order function \mathbf{ff} such that the least fixpoint of \mathbf{ff} is f . In addition to the function declaration, please also give the type declaration for \mathbf{ff} . You may use full Haskell for \mathbf{ff} .

Name:

Matriculation Number:

- ii) Let $\phi_{\mathbf{ff}}$ be the semantics of the function \mathbf{ff} . Give the definition of $\phi_{\mathbf{ff}}^m(\perp)$ in closed form for any $m \in \mathbb{N}$, i.e., give a non-recursive definition of the function that results from applying $\phi_{\mathbf{ff}}$ m -times to \perp . Here, you should assume that \mathbf{Int} can represent all integers, so no overflow can occur.

Hints:

- The binomial coefficient $\binom{n}{k}$ for $n \in \mathbb{N}$, $k \in \mathbb{Z}$ is defined as follows:

$$\binom{n}{k} = \begin{cases} \frac{n!}{k! \cdot (n-k)!}, & 0 \leq k \leq n \\ 0, & k < 0 \text{ or } k > n \end{cases}.$$

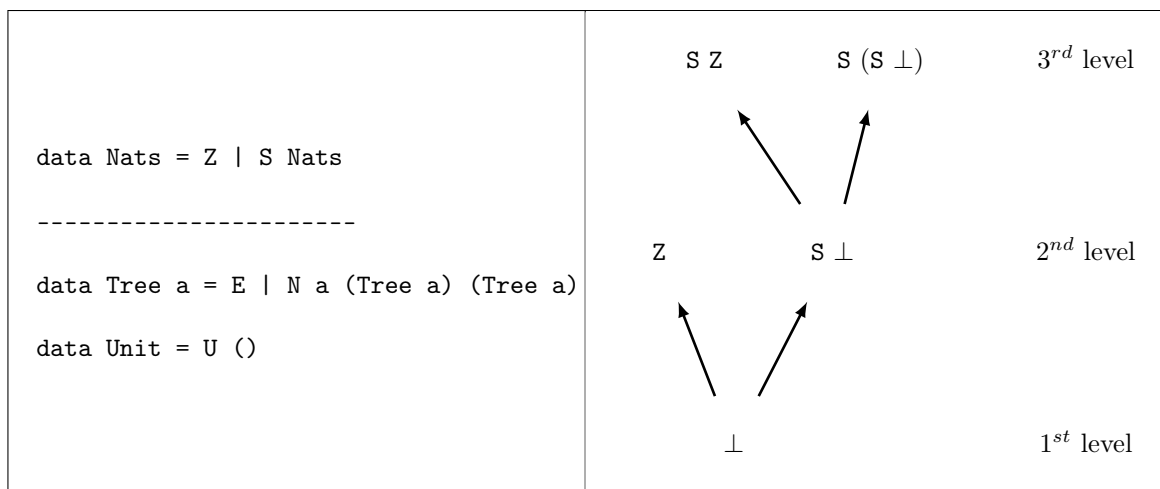
- For $n \in \mathbb{N}$, $k \in \mathbb{Z}$ we have $\binom{n+1}{k} = \binom{n}{k} + \binom{n}{k-1}$.

- iii) Give the definition of the least fixpoint of $\phi_{\mathbf{ff}}$ in closed form.

Name:

Matriculation Number:

- c) Consider the data type declarations on the left and, as an example, the graphical representation of the first three levels of the domain for **Nats** on the right:



Give a graphical representation of the first three levels of the domain for the type **Tree Unit**.

Exercise 3 (Lambda Calculus):

$((4 + 4) + 8 + 6 = 22 \text{ points})$

- a) Consider the following function that determines whether the first list is shorter or as long as than the second. Here, lists are represented by the data structure `List a` defined by `data List a = N | C a (List a)`.

```
cp :: List a -> List a -> Bool
cp N ys = True
cp (C x xs) N = False
cp (C x xs) (C y ys) = cp xs ys
```

- i) Please give an equivalent function in *simple Haskell*.
- ii) Implement the function `cp` in the *lambda calculus*, i.e., give a lambda term q such that, for all lists l_1, l_2 the term $q \ l_1 \ l_2$ can be reduced to `True` if and only if l_1 is at most as long as l_2 , and to `False` otherwise.

Hints:

- You do *not* have to use the transformation algorithms presented in the lecture. It is sufficient to just give an equivalent simple program and an equivalent lambda term.
- You can use infix notation for predefined functions like `(&&)` in both simple Haskell and the lambda calculus.

Name:

Matriculation Number:

b) Let

$$t = \lambda g n. \text{if } (n == 0) (\lambda z. \text{True}) (\lambda z. g \ 0)$$

and

$$\begin{aligned} \delta = \{ & \text{if True} \rightarrow \lambda x y. x, \\ & \text{if False} \rightarrow \lambda x y. y, \\ & \text{fix} \rightarrow \lambda f. f(\text{fix } f) \} \\ \cup \{ & x == y \rightarrow \text{True} \mid x, y \in \mathbb{Z}, x = y \} \\ \cup \{ & x == y \rightarrow \text{False} \mid x, y \in \mathbb{Z}, x \neq y \} \end{aligned}$$

Please reduce $\text{fix } t \ 1$ by WHNO-reduction with the $\rightarrow_{\beta\delta}$ -relation. List **all** intermediate steps until reaching weak head normal form, but please write “ t ” instead of

$$\lambda g n. \text{if } (n == 0) (\lambda z. \text{True}) (\lambda z. g \ 0)$$

whenever possible.

Name:

Matriculation Number:

- c) Consider the representation of natural numbers in the pure lambda calculus presented in the lecture (i.e., $n \in \mathbb{N}$ is represented by the term $\bar{n} = \lambda f x. f^n x$). Give a pure lambda term for the function `minus`, such that for $m, n \in \mathbb{N}$ the expression `minus \bar{m} \bar{n}` can be reduced to
$$\begin{cases} \overline{m - n}, & \text{if } n \leq m \\ \bar{0}, & \text{otherwise.} \end{cases}$$

Explain your solution shortly. You may give a reduction sequence as explanation.

Hints:

- You can assume that there is a pure lambda term `pred` such that `pred \bar{n}` can be reduced to
$$\begin{cases} \overline{n - 1}, & \text{if } n > 0 \\ \bar{0}, & \text{if } n = 0. \end{cases}$$

Name:

Matriculation Number:

Exercise 4 (Type Inference):
(18 points)

Using the initial type assumption $A_0 := \{f :: \forall a. a \rightarrow a, g :: \forall a. \text{Bool} \rightarrow a \rightarrow a\}$, infer the type of the expression $\lambda x. g (f x) x$ using the algorithm \mathcal{W} .

Hints:

- When writing $\mathcal{W}(A, t) = (\theta, t)$, you do not have to give the full substitution θ , but it is enough to give the parts of θ that concern the free variables in the type schemas of A .