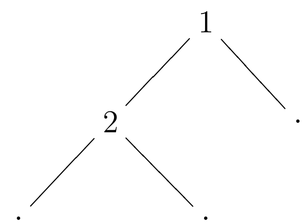
First name	Last name	Matriculation number

Exercise 1 (2+2+2 points)

The following data structure represents binary trees only containing values in the inner nodes:
`data Tree a = Leaf | Node (Tree a) a (Tree a)`

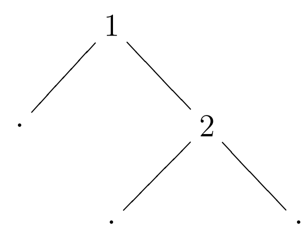
Consider the tree `t` of integers on the right-hand side. The representation of `t` as an object of type `Tree Int` in Haskell would be:



`Node (Node Leaf 2 Leaf) 1 Leaf`

Implement the following functions in Haskell.

(a) Please implement the function `swapTree` which returns the tree where all left children have been swapped with the corresponding right children. When computing `swapTree t` one would obtain the tree on the right-hand side. Apart from the function declaration, also give the most general type declaration for `swapTree`. You may not use any higher-order functions.




```
swapTree :: Tree a -> Tree a
swapTree Leaf = Leaf
swapTree (Node l x r) = Node (swapTree r) x (swapTree l)
```

First name	Last name	Matriculation number

(b) The function `foldTree` of type `(a -> b -> a -> a) -> a -> Tree b -> a` works as follows: `foldTree n l t` replaces all occurrences of the constructor `Node` in the tree `t` by `n` and it replaces all occurrences of the constructor `Leaf` in `t` by `l`. Suppose there is the function `add`:

```
add :: Int -> Int -> Int -> Int
add x y z = x + y + z
```

For the tree `t` from (a), "`foldTree add 0 t`" should compute:

```
foldTree add 0 (Node (Node Leaf 2 Leaf) 1 Leaf)
              = add  (add  0    2 0   ) 1 0
              = 3
```

Here, `Node` is replaced by `add` and `Leaf` is replaced by `0`.

Now use the `foldTree` function to implement the `swapTree` function again.

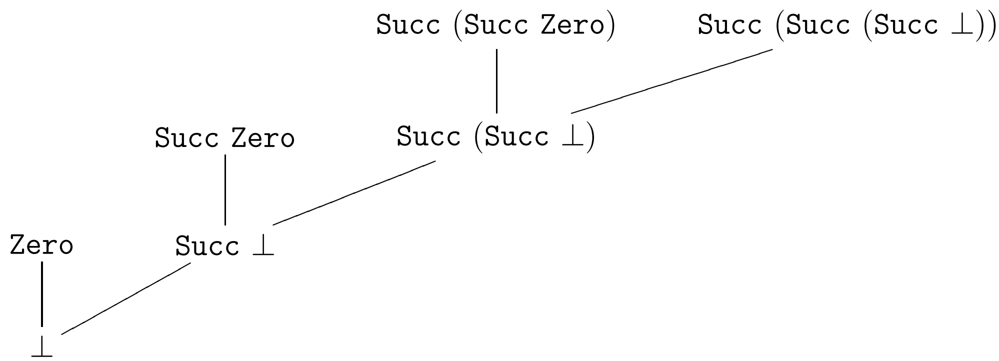
```
swapTree = foldTree (\x y z -> Node z y x) Leaf
```

First name	Last name	Matriculation number

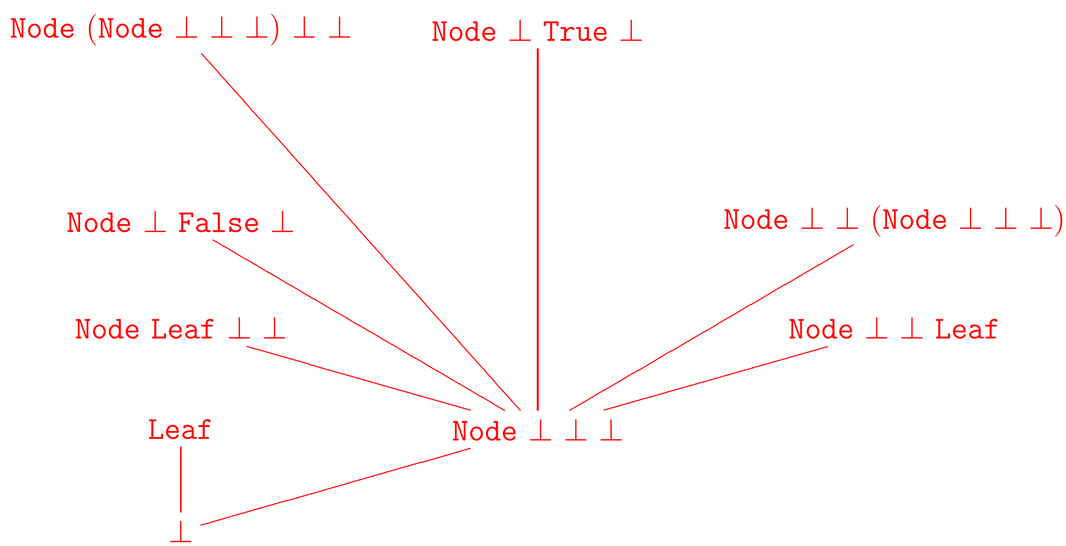
(c) Consider the following data type declaration for natural numbers:

```
data Nats = Zero | Succ Nats
```

A graphical representation of the first four levels of the domain for `Nats` could look like this:



Sketch a graphical representation of the first three levels of the domain for the data type `Tree Bool`.



First name	Last name	Matriculation number

Exercise 2 (2+3 points)

Consider the following Haskell declarations for the `half` function:

```
half :: Int -> Int
half (x+2) = 1 + (half x)
half _ = 0
```

(a) Give the Haskell declarations for the higher-order function `f_half` corresponding to `half`, i.e., the higher-order function `f_half` such that the least fixpoint of `f_half` is `half`. In addition to the function declaration(s), also give the type declaration of `f_half`. Since you may use full Haskell for `f_half`, you do not need to translate `half` into simple Haskell.

```
f_half :: (Int -> Int) -> (Int -> Int)
f_half half (x+2) = 1 + (half x)
f_half half _ = 0
```

(b) We add the Haskell declaration `bot = bot`. For each $n \in \mathbb{N}$ determine which function from $\mathbb{Z}_\perp$ to $\mathbb{Z}_\perp$ is computed by $\texttt{f\_half}^n$ `bot`. Here "$\texttt{f\_half}^n$ `bot`" represents the n-fold application of `f_half` to `bot`, i.e., it denotes $\underbrace{\texttt{f\_half (f\_half} \ldots \texttt{(f\_half}}_{n \text{ times}} \texttt{ bot)} \ldots)$.

Give the function computed by "$\texttt{f\_half}^n$ `bot`" in closed form, i.e., using a non-recursive definition.

$$(\texttt{f\_half}^n(\perp))(x) = \begin{cases} \lfloor \frac{x}{2} \rfloor, & \text{if } 1 < x < 2n \\ 0, & \text{if } x \leq 1 \wedge n > 0 \\ \perp, & \text{if } n = 0 \vee x = \perp \vee x \geq 2n \end{cases}$$

First name	Last name	Matriculation number

Exercise 3 (3+3 points)

Let $\sqsubseteq$ be a complete order and let f be a function which is continuous (and therefore also monotonic).

Prove or disprove the following statements:

(a) $\{ \ f^n(\bot) \ \mid \ n \in \{0,1,2,\ldots\} \ \}$ is a chain.

We prove $f^n(\bot) \sqsubseteq f^{n+1}(\bot)$ for all $n \in \{0,1,2,\ldots\}$ by induction on n.

- $n = 0$: By definition we have $\bot \sqsubseteq f(\bot)$.
- $n \to n+1$: The function f is continuous and therefore also monotonic. Thus, $f^n(\bot) \sqsubseteq f^{n+1}(\bot)$ implies $f^{n+1}(\bot) \sqsubseteq f^{n+2}(\bot)$.

(b) $\sqcup\{ \ f^n(\bot) \ \mid \ n \in \{0,1,2,\ldots\} \ \}$ is a fixpoint of f.

$$
\begin{aligned}
f(\sqcup\{f^n(\bot) \mid n \in \{0,1,2,\ldots\}\}) &\overset{f \text{ continuous}}{=} \sqcup f(\{f^n(\bot) \mid n \in \{0,1,2,\ldots\}\}) \\
&= \sqcup\{f^{n+1}(\bot) \mid n \in \{0,1,2,\ldots\}\} \\
&= \sqcup\{f^n(\bot) \mid n \in \{1,2,\ldots\}\} \\
&= \sqcup(\{f^n(\bot) \mid n \in \{1,2,\ldots\}\} \cup \{\bot\}) \\
&= \sqcup(\{f^n(\bot) \mid n \in \{1,2,\ldots\}\} \cup \{f^0(\bot)\}) \\
&= \sqcup\{f^n(\bot) \mid n \in \{0,1,2,\ldots\}\}
\end{aligned}
$$

First name	Last name	Matriculation number

Exercise 4 (3 points)

We define the following algebraic data type for lists:

```
data List a = Nil | Cons a (List a)
```

Write declarations in **simple** Haskell for the function `maxList :: List Int -> Int`. Here, for empty lists the function should return `bot`. For non-empty lists, `maxList` should return the maximum of that list. For example, `maxList (Cons 1 (Cons (-2) Nil))` should return `1`.

Your solution should use the functions defined in the transformation from the lecture such as $\texttt{sel}_{n,i}$, $\texttt{isa}_{\underline{\texttt{constr}}}$, $\texttt{argof}_{\underline{\texttt{constr}}}$, and `bot`. You do not have to use the transformation rules from the lecture, though. Additionally, you may use the built-in function `max :: Int -> Int -> Int` for computing the maximum of two integers.

```
maxList= \xs -> if (isa_Cons xs)
                   then if (isa_Nil (sel_2,2 (argof_Cons xs)))
                        then sel_2,1 (argof_Cons xs)
                        else max (sel_2,1 (argof_Cons xs)) (maxList (sel_2,2 (argof_Cons xs)))
                   else bot
```

First name	Last name	Matriculation number

Exercise 5 (2+4 points)

Consider the following data structures for natural numbers and polymorphic lists:

```
data Nats = Zero | Succ Nats
data List a = Nil | Cons a (List a)
```

Let δ be the set of rules from Definition 3.3.5, i.e., δ contains among others the following rules:

$$\begin{array}{rcl} \texttt{fix} & \rightarrow & \lambda f.\ f\ (\texttt{fix}\ f) \\ \texttt{if True} & \rightarrow & \lambda x\ y.\ x \\ \texttt{isa}_{\texttt{Nil}}\ \texttt{Nil} & \rightarrow & \texttt{True} \end{array}$$

(a) Please translate the following Haskell-expression into an equivalent lambda term (e.g., using $\mathcal{L}am$). It suffices to give the result of the transformation.

$\texttt{let length=}\ \backslash\texttt{xs -> if (isa}_{\texttt{Nil}}\ \texttt{xs) then Zero}$
$\qquad\qquad\qquad\qquad\texttt{else Succ (length (sel}_{2,2}\ \texttt{(argof}_{\texttt{Cons}}\ \texttt{xs)))}$
$\texttt{in length}$

$\texttt{fix}\ (\lambda length\ xs.\ \texttt{if}\ (\texttt{isa}_{\texttt{Nil}}\ xs)\ \texttt{Zero}\ (\texttt{Succ}\ (length\ (\texttt{sel}_{2,2}\ (\texttt{argof}_{\texttt{Cons}}\ xs)))))$

First name	Last name	Matriculation number

(b) Let "$\texttt{fix}\, t$" be the lambda term from (a). Please reduce "$(\texttt{fix}\ t)\ \texttt{Nil}$" by WHNO-reduction with the $\to_{\beta\delta}$-relation. You have to give **all** intermediate steps until one reaches **weak head normal form**.

We have $t = (\lambda length\ xs.\ \texttt{if}\ (\texttt{isa}_{\texttt{Nil}}\ xs)\ \texttt{Zero}\ (\texttt{Succ}\ (length\ (\texttt{sel}_{2,2}\ (\texttt{argof}_{\texttt{Cons}}\ xs)))))$

$$\begin{array}{ll}
 & \texttt{fix}\ t\ \texttt{Nil} \\
\to_\delta & (\lambda f.\ f\ (\texttt{fix}\ f))\ t\ \texttt{Nil} \\
\to_\beta & t\ (\texttt{fix}\ t)\ \texttt{Nil} \\
\to_\beta & (\lambda xs.\ \texttt{if}\ (\texttt{isa}_{\texttt{Nil}}\ xs)\ \texttt{Zero}\ (\texttt{Succ}\ (\texttt{fix}\ t\ (\texttt{sel}_{2,2}\ (\texttt{argof}_{\texttt{Cons}}\ xs)))))\ \texttt{Nil} \\
\to_\beta & \texttt{if}\ (\texttt{isa}_{\texttt{Nil}}\ \texttt{Nil})\ \texttt{Zero}\ (\texttt{Succ}\ (\texttt{fix}\ t\ (\texttt{sel}_{2,2}\ (\texttt{argof}_{\texttt{Cons}}\ \texttt{Nil})))) \\
\to_\delta & \texttt{if}\ \texttt{True}\ \texttt{Zero}\ (\texttt{Succ}\ (\texttt{fix}\ t\ (\texttt{sel}_{2,2}\ (\texttt{argof}_{\texttt{Cons}}\ \texttt{Nil})))) \\
\to_\delta & (\lambda x\ y.\ x)\ \texttt{Zero}\ (\texttt{Succ}\ (\texttt{fix}\ t\ (\texttt{sel}_{2,2}\ (\texttt{argof}_{\texttt{Cons}}\ \texttt{Nil})))) \\
\to_\beta & (\lambda y.\ \texttt{Zero})\ (\texttt{Succ}\ (\texttt{fix}\ t\ (\texttt{sel}_{2,2}\ (\texttt{argof}_{\texttt{Cons}}\ \texttt{Nil})))) \\
\to_\beta & \texttt{Zero}
\end{array}$$

First name	Last name	Matriculation number

Exercise 6 (4 points)

Use the type inference algorithm $\mathcal{W}$ to determine the most general type of the following λ-term under the initial type assumption A_0. Show the results of all sub-computations and unifications, too. If the term is not well typed, show how and why the $\mathcal{W}$-algorithm detects this.

$$\lambda f.\ f\ (\texttt{Succ Zero})$$

The initial type assumption A_0 contains at least the following:

$$\begin{array}{lcl} A_0(f) & = & \forall a.\ a \\ A_0(\texttt{Succ}) & = & \texttt{Nats} \to \texttt{Nats} \\ A_0(\texttt{Zero}) & = & \texttt{Nats} \end{array}$$

$W(A_0,\ \lambda f.\ f\ (\texttt{Succ Zero}))$

$\quad W(A_0 + \{f :: b_0\},\ f\ (\texttt{Succ Zero}))$

$\quad\quad W(A_0 + \{f :: b_0\},\ f)$

$\quad\quad = (\texttt{id},\ b_0)$

$\quad\quad W(A_0 + \{f :: b_0\},\ \texttt{Succ Zero})$

$\quad\quad\quad W(A_0 + \{f :: b_0\},\ \texttt{Succ})$

$\quad\quad\quad = (\texttt{id},\ (\texttt{Nats} \to \texttt{Nats}))$

$\quad\quad\quad W(A_0 + \{f :: b_0\},\ \texttt{Zero})$

$\quad\quad\quad = (\texttt{id},\ \texttt{Nats})$

$\quad\quad\quad$ building mgu of $(\texttt{Nats} \to \texttt{Nats})$ and $(\texttt{Nats} \to b_1) = [b_1/\texttt{Nats}]$

$\quad\quad = ([b_1/\texttt{Nats}],\ \texttt{Nats})$

$\quad\quad$ building mgu of b_0 and $(\texttt{Nats} \to b_2) = [b_0/(\texttt{Nats} \to b_2)]$

$\quad = ([b_1/\texttt{Nats}, b_0/(\texttt{Nats} \to b_2)],\ b_2)$

$= ([b_1/\texttt{Nats}, b_0/(\texttt{Nats} \to b_2)],\ ((\texttt{Nats} \to b_2) \to b_2))$

Resulting type: $((\texttt{Nats} \to b_2) \to b_2)$