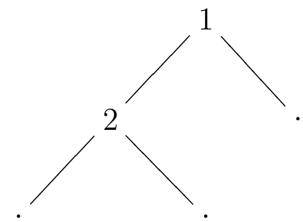
First name	Last name	Matriculation number

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)
```

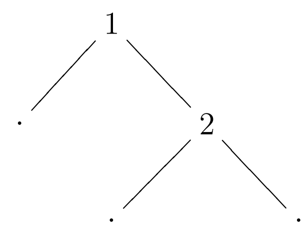
Consider the tree `t` of integers on the right-hand side. The representation of `t` as an object of type `Tree Int` in Haskell would be:




```
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.



(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`.

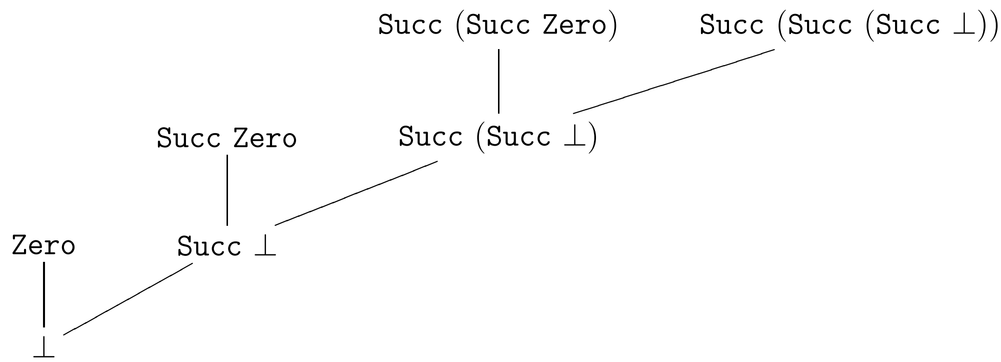
Now use the `foldTree` function to implement the `swapTree` function again.

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.

(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 $\texttt{f\_half}^n\ \texttt{bot}$. Here "$\texttt{f\_half}^n\ \texttt{bot}$" 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 "$\texttt{f\_half}^n\ \texttt{bot}$" in closed form, i.e., using a non-recursive definition.

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.

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

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 $\mathtt{sel}_{n,i}$, $\mathtt{isa}_{\underline{\mathtt{constr}}}$, $\mathtt{argof}_{\underline{\mathtt{constr}}}$, 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.

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.

```
let length= \xs -> if (isa_Nil xs) then Zero
                     else Succ (length (sel_2,2 (argof_Cons xs)))
in length
```

First name	Last name	Matriculation number

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

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.

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

The initial type assumption A_0 contains at least the following:

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