Exercise 1 (2+2+2 points)

The following data structure represents binary trees only containing values in the inner nodes:

```haskell
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 \( \text{Tree Int} \) in Haskell would be:

```haskell
Node (Node Leaf 2 Leaf) 1 Leaf
```

Implement the following functions in Haskell.

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

\[
\text{swapTree} \colon \text{Tree a} \rightarrow \text{Tree a}
\]

```
\[
\text{swapTree} \ Leaf = \ Leaf
\]

\[
\text{swapTree} \ (\text{Node} \ l \ x \ r) = \text{Node} \ \text{swapTree} \ r \ x \ \text{swapTree} \ l
\]

```
(b) The function \texttt{foldTree} of type \( (a \to b \to a \to a) \to a \to \text{Tree}\ b \to a \) works as follows: \texttt{foldTree} \( n\ 1\ t \) replaces all occurrences of the constructor \texttt{Node} in the tree \( t \) by \( n \) and it replaces all occurrences of the constructor \texttt{Leaf} in \( t \) by \( 1 \). Suppose there is the function \texttt{add}:

\[
\begin{align*}
\text{add} & \colon \text{Int} \to \text{Int} \to \text{Int} \to \text{Int} \\
\text{add}\ x\ y\ z &= x + y + z
\end{align*}
\]

For the tree \( t \) from (a), \texttt{"foldTree \ add\ 0\ t"} should compute:

\[
\begin{align*}
\text{foldTree}\ \text{add}\ 0\ (\text{Node}\ (\text{Node}\ \text{Leaf}\ 2\ \text{Leaf})\ 1\ \text{Leaf}) \\
&= \text{add}\ (\text{add}\ 0\ 2\ 0)\ 1\ 0 \\
&= 3
\end{align*}
\]

Here, \texttt{Node} is replaced by \texttt{add} and \texttt{Leaf} is replaced by \( 0 \).

Now use the \texttt{foldTree} function to implement the \texttt{swapTree} function again.
(c) Consider the following data type declaration for natural numbers:

\[
\text{data Nats = Zero | Succ Nats}
\]

A graphical representation of the first four levels of the domain for \text{Nats} could look like this:

\[
\begin{array}{c}
\text{Succ (Succ (Succ (Succ ⊥))}) \\
\text{Succ (Succ ⊥)} \\
\text{Succ Zero} \\
\text{Zero} \\
\text{⊥}
\end{array}
\]

Sketch a graphical representation of the first three levels of the domain for the data type \text{Tree Bool}. 
Exercise 2 (2+3 points)

Consider the following Haskell declarations for the \texttt{half} function:

\begin{verbatim}
half :: Int -> Int
half (x+2) = 1 + (half x)
half _ = 0
\end{verbatim}

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

\begin{verbatim}
f\_half :: (Int -> Int) -> (Int -> Int)
f\_half half (x+2) = 1 + (half x)
f\_half half _ = 0
\end{verbatim}

(b) We add the Haskell declaration \texttt{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 \texttt{f\_half} to \texttt{bot}, i.e., it denotes \(\underbrace{\texttt{f\_half} (\texttt{f\_half} \ldots (\texttt{f\_half} \texttt{bot}) \ldots)}_{n \text{ times}}\).

Give the function computed by \(\texttt{f\_half}^n \texttt{bot}\) in closed form, i.e., using a non-recursive definition.
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 \( f \).
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 `sel_n`, `isa_constr`, `argof_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 \( \delta \) be the set of rules from Definition 3.3.5, i.e., \( \delta \) contains among others the following rules:

\[
\begin{align*}
\text{fix} & \rightarrow \lambda f. f (\text{fix} f) \\
\text{if True} & \rightarrow \lambda x y. x \\
\text{isa}_{N1} \text{Nil} & \rightarrow \text{True}
\end{align*}
\]

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

\[
\begin{align*}
\text{let } \text{length} & = \lambda \text{xs} \rightarrow \text{if } (\text{isa}_{N1} \text{xs}) \text{ then Zero } \\
& \quad \text{else Succ } (\text{length} (\text{sel}_{2,2} (\text{argof} \text{Cons} \text{xs}))) \\
\text{in } \text{length}
\end{align*}
\]
(b) Let “fix $t$” be the lambda term from (a). Please reduce “(fix $t$) Nil” by WHNO-reduction with the $\rightarrow_{\beta\delta}$-relation. You have to give all intermediate steps until one reaches weak head normal form.
Exercise 6 (4 points)

Use the type inference algorithm $\mathcal{W}$ to determine the most general type of the following $\lambda$-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.

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

The initial type assumption $A_0$ contains at least the following:

$$A_0(f) = \forall a. a$$
$$A_0(\text{Succ}) = \text{Nats} \rightarrow \text{Nats}$$
$$A_0(\text{Zero}) = \text{Nats}$$