

Exam in Functional Programming SS 2012 (V3M)

First Name: _____

Last Name: _____

Matriculation Number: _____

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

- Informatik Bachelor
- Mathematik Master
- Informatik Master
- Software Systems Engineering Master
- Other: _____

	Available Points	Achieved Points
Exercise 1	20	
Exercise 2	38	
Exercise 3	36	
Exercise 4	20	
Exercise 5	6	
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.
- 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 give you **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**.

Exercise 1 (Quiz):
(4 + 4 + 4 + 4 + 4 = 20 points)

Give a short proof sketch or a counterexample for each of the following statements:

- a) Is \sqsubseteq always a complete partial order for flat domains like $\mathbb{Z}_\perp, \mathbb{B}_\perp, \dots$?
- b) Can the function $f : \mathbb{Z}_\perp \rightarrow \mathbb{Z}$ with $f(x) = \begin{cases} 1 & \text{if } x \in \mathbb{Z} \text{ and } x \leq 0 \\ 0 & \text{otherwise} \end{cases}$ be implemented in Haskell?
- c) Is $g : (\mathbb{Z} \rightarrow \mathbb{Z}_\perp) \rightarrow \mathbb{Z}_\perp$ with $g(h) = \begin{cases} 0 & \text{if } h(x) \neq \perp \text{ for all } x \in \mathbb{Z} \\ \perp & \text{otherwise} \end{cases}$ continuous?
- d) If a lambda term t can be reduced to s with $\rightarrow_{\beta\delta}$ using an outermost strategy, can t also be reduced to s with $\rightarrow_{\beta\delta}$ using an innermost strategy? Here, you may choose an arbitrary delta-rule set δ .
- e) The $\rightarrow_{\beta\delta}$ reduction in lambda calculus is confluent. Is Simple Haskell also confluent?

Exercise 2 (Programming in Haskell):
(10 + 10 + 8 + 10 = 38 points)

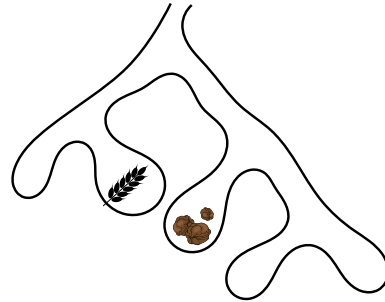
We define a polymorphic data structure `HamsterCave` to represent hamster caves which can contain different types of food.

```
data HamsterCave food
  = EmptyTunnel
  | FoodTunnel food
  | Branch (HamsterCave food) (HamsterCave food)
  deriving Show
```

The data structure `HamsterFood` is used to represent food for hamsters. For example, `exampleCave` is a valid expression of type `HamsterCave HamsterFood`.

```
data HamsterFood = Grain | Nuts deriving Show
```

```
exampleCave :: HamsterCave HamsterFood
exampleCave = Branch
  (Branch EmptyTunnel (FoodTunnel Grain))
  (Branch (FoodTunnel Nuts) (Branch EmptyTunnel EmptyTunnel))
```



- a) Implement a function `digAndFillCave :: Int -> HamsterCave HamsterFood`, such that for any integer number $n > 1$, `digAndFillCave n` creates a hamster cave without empty tunnels of depth n , such that the number of `FoodTunnels` containing `Grain` equals the number of `FoodTunnels` containing `Nuts`. Here, the depth of a cave is the maximal number of “nodes” on any path from the entry of the cave to a dead end. Thus, `exampleCave` has depth 4.

- b) Implement a fold function `foldHamsterCave`, including its type declaration, for the data structure `HamsterCave`. As usual, the fold function replaces the data constructors in a `HamsterCave` expression by functions specified by the user. The first argument of `foldHamsterCave` should be the function for the case of the empty tunnel, the second argument the function for the case of the food tunnel, and the third argument the function for the case of a branch. As an example, the following function definition uses `foldHamsterCave` to determine the number of dead ends (either with or without food) in a cave, such that the call `numberOfDeadEnds exampleCave` returns 5.

```
numberOfDeadEnds :: HamsterCave food -> Int
numberOfDeadEnds cave = foldHamsterCave 1 (\_ -> 1) (+) cave
```

- c) Implement the function `collectFood :: HamsterCave food -> (HamsterCave food, [food])`, which returns a tuple for a given hamster cave. The first argument of the tuple is the same hamster cave as the one given to the function, but without any food (i.e., every `FoodTunnel` is replaced by an `EmptyTunnel`). The second argument is a list of all the food that was removed from the cave. For the definition of `collectFood`, use only one defining equation where the right-hand side is a call to the function `foldHamsterCave`.

For example, a call `collectFood exampleCave` should return the following tuple:

```
(Branch (Branch EmptyTunnel EmptyTunnel)
        (Branch EmptyTunnel (Branch EmptyTunnel EmptyTunnel)))
,[Grain,Nuts]
```

- d) In this part of the exercise, you should create a program that navigates through a hamster cave. To do this, implement a function `exploreHamsterCave :: Show food => HamsterCave food -> IO ()` that, given a `HamsterCave` data structure, prints the kind of food it found at the current position if the `HamsterCave` is a `FoodTunnel`, prints "Didn't find any food" for an `EmptyTunnel`, and asks the user whether one should descend into the left or right part of the cave in case of a `Branch`. The user should answer with either `l` for the left cave or `r` for the right cave. If the answer of the user is valid, the function should continue in the corresponding part of the hamster cave. Otherwise, the question should be repeated after a short explanation about the possible options.

A successful run for the `exampleCave` might look as follows:

```
*Main> exploreHamsterCave exampleCave
You are standing at a fork in the tunnel. Proceed to the (l)eft or to the (r)ight? l
You are standing at a fork in the tunnel. Proceed to the (l)eft or to the (r)ight? up
Sorry, I did not understand you. Please answer with l for left or r for right.
You are standing at a fork in the tunnel. Proceed to the (l)eft or to the (r)ight? r
Yay! Found Grain!
```

In the following run, the user does not manage to find any food in the cave:

```
*Main> exploreHamsterCave exampleCave
You are standing at a fork in the tunnel. Proceed to the (l)eft or to the (r)ight? l
You are standing at a fork in the tunnel. Proceed to the (l)eft or to the (r)ight? l
Didn't find any food.
```

Hint: You should use the function `getLine :: IO String` to read the input from the user and the function `putStrLn :: String -> IO ()` to print a `String`. To save space, you may assume that the following declarations exist in your program:

```
question, mistake :: String
question = "You are standing at a fork in the tunnel. "
  ++ "Proceed to the (l)eft or to the (r)ight?"
mistake = "Sorry, I did not understand you. "
  ++ "Please answer with l for left or r for right.\n"
```

Exercise 3 (Semantics):
(21 + 10 + 5 = 36 points)

- a) i) Let \sqsubseteq be a cpo on D and $f : D \rightarrow D$ be continuous. Prove the fixpoint theorem, i.e., that $\sqcup\{f^i(\perp) \mid i \in \mathbb{N}\}$ exists and that this is the least fixpoint of f . You may use all other results from the lecture in your proof.

- ii) Let $D = 2^{\mathbb{N}}$, i.e., D is the set of all sets of natural numbers and let \subseteq denote the usual subset relation.

1) Prove that every chain $S \subseteq D$ has a least upper bound w.r.t. the relation \subseteq .

2) Prove that \subseteq is a cpo on D .

3) Give an example for an infinite chain in (D, \subseteq) .

4) Give a monotonic, non-continuous function $f : D \rightarrow D$. You do not need to prove that f has these properties.

b) i) Consider the following Haskell function `mult`:

```
mult :: (Int, Int) -> Int
mult (0, y) = 0
mult (x, y) = y + mult (x - 1, y)
```

Please give the Haskell declaration for the higher-order function `f_mult` corresponding to `mult`, i.e., the higher-order function `f_mult` such that the least fixpoint of `f_mult` is `mult`. In addition to the function declaration, please also give the type declaration of `f_mult`. You may use full Haskell for `f_mult`.

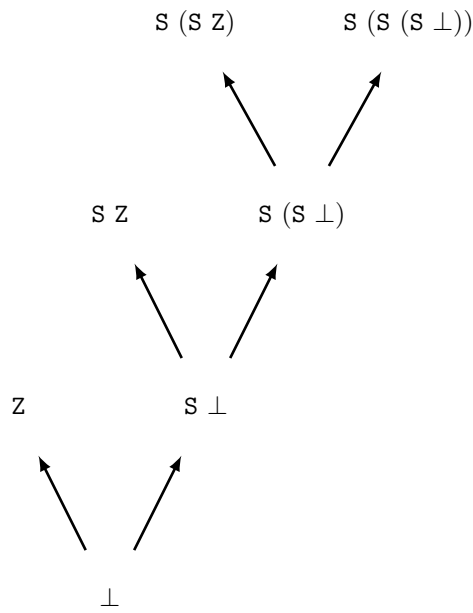
ii) Let ϕ_{f_mult} be the semantics of the function `f_mult`. Give the semantics of $\phi_{f_mult}^n(\perp)$ for $n \in \mathbb{N}$, i.e., the semantics of the n -fold application of ϕ_{f_mult} to \perp .

iii) Give all fixpoints of ϕ_{f_mult} and mark the least fixpoint.

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

```
data Nats = Z | S Nats
```

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



Now consider the following data type declarations:

`data X = A X Y | B Y`

`data Y = E Y | H`

Give a graphical representation of the first three levels of the domain for the type `X`. The third level contains the element `A (A ⊥ ⊥) ⊥`, for example.

Exercise 4 (Lambda Calculus):
(4 + 6 + 10 = 20 points)

- a) Please translate the following Haskell expression into an equivalent lambda term (e.g., using \mathcal{Lam}). Translate the pre-defined function `<` to **LessThan**, `+` to **Plus** and `-` to **Minus** (remember that the infix notation of `<`, `+`, `-` is not allowed in lambda calculus). It suffices to give the result of the transformation:

```
let quot = \x y -> if x < y then 0 else 1 + quot (x-y) y in quot v w
```

- b) Let $t = \lambda fact. (\lambda x. (\text{If } (\text{LessThanOrE } x \ 1) \ 1 \ (\text{Times } x \ (fact \ (\text{Minus } x \ 1))))$ and

$$\begin{aligned} \delta = & \{ \text{If True} \rightarrow \lambda x \ y. x, \\ & \text{If False} \rightarrow \lambda x \ y. y, \\ & \text{fix} \rightarrow \lambda f. f(\text{fix } f) \} \\ \cup & \{ \text{Minus } x \ y \rightarrow z \mid x, y \in \mathbb{Z} \wedge z = x - y \} \\ \cup & \{ \text{Times } x \ y \rightarrow z \mid x, y \in \mathbb{Z} \wedge z = x \cdot y \} \\ \cup & \{ \text{LessThanOrE } x \ y \rightarrow b \mid x, y \in \mathbb{Z} \wedge ((x \leq y \wedge b = \text{True}) \vee (x > y \wedge b = \text{False})) \} \end{aligned}$$

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 the term it represents whenever possible.

Name:

Matriculation Number:

c) We use the representation of natural numbers in the pure λ -calculus presented in the lecture. So, n is represented as $\lambda f x.f^n x$, i.e., 0 is represented as $\lambda f x.x$, 1 as $\lambda f x.f x$ and so on. Using this representation, give pure λ -terms for the following functions:

- $\overline{\text{Succ}}$, which increases a natural number n to $n + 1$. Thus, $\overline{\text{Succ}} (\lambda f x.t) \rightarrow_{\beta}^* \lambda f x.f t$, where t is the term $f^n x$.

- $\overline{\text{Plus}}$, which adds two natural numbers n, m . Here, you may use the λ -term $\overline{\text{Succ}}$. Recall that $n + m$ is the same as the m -th successor of n .

Name: _____

Matriculation Number: _____

Exercise 5 (Type Inference):**(6 points)**

Using the initial type assumption $A_0 := \{x :: \forall a.a \rightarrow \text{Int}\}$ infer the type of the expression $\lambda y.y x$ using the algorithm \mathcal{W} .