Name: _____

Matriculation Number: _____

Points: _____

This exam consists of 4 exercises, for a total of 44 points (so there is roughly 1 point per 2 minutes).
The available points per exercise are written in the margin.

Exercise 1: Types and Type-Classes 9

Consider the following Haskell code:

```
data Tree a = Nil | Node (Tree a) a (Tree a) deriving (Eq,Ord)

c1 = Node
c2 = Node Nil $ True (Node Nil False Nil)
c3 = Node Nil 5 $ Node Nil 7 Nil

f x = if x > Nil then x else Nil
```

(a) For each of the constants `c1`, `c2` and `c3`, either specify its most general type, or state that its definition yields a typing error. (6)

(b) Of the following statements exactly one is correct. Mark it or don't mark any answer and get 1 point. (3)

- ☐ `f` is not type-correct.
- ☐ The most general type of `f` is `Tree a -> Tree a`.
- ☐ The most general type of `f` is `Ord a => Tree a -> Tree a`.
- ☐ The most general type of `f` is `(Ord a, Ord b) => Tree a -> Tree b`.
- ☐ The most general type of `f` is `Ord a => Tree a -> Tree b`.

Exercise 2: Lazy Evaluation 12

Consider the following Haskell code:

```
d2 [] = []
d2 [_] = []
d2 (x : _ : xs) = x : d2 xs

list_1 = d2 . filter odd $ [0..]

list_2 = filter odd . d2 $ [0..]

list_3 = 1 : zipWith (+) list_3 (d2 [5..])
```

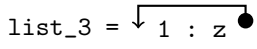
(a) Of the following statements exactly one is correct. Mark it or don't mark any answer and get 1 point. (3)

- ☐ `list_1` is the list of all odd numbers `[1,3,5,7,...]`
- ☐ `list_1` is the list `[0,4,8,12,...]`
- ☐ `list_1` is the list `[1,5,9,13,...]`
- ☐ `list_1` evaluates to `[]`
- ☐ `list_1` is non-terminating and does not produce a single element

(b) Of the following statements exactly one is correct. Mark it or don't mark any answer and get 1 point. (3)

- ☐ `list_2` is the list of all even numbers `[0,2,4,6,8,...]`
- ☐ `list_2` is the list `[1,5,9,13,...]`
- ☐ `list_2` is the list `[0,4,8,12,...]`
- ☐ `list_2` evaluates to `[]`
- ☐ `list_2` is non-terminating and does not produce a single element

(c) Continue the evaluation of `list_3` step by step until the first 3 elements are constructed. Here, you can assume that arithmetic is always immediately evaluated. You may abbreviate `zipWith (+)` as `z`. (6)

`list_3 =`  `(d2 [5..]) =`

Exercise 3: Programming with Lists 16

Consider the following Haskell code which contains a list of newborns of different countries.

```
type First_Name = String
type Last_Name = String
type Country = String

data PersonT = Person First_Name Last_Name Country

newborns_2019 :: [PersonT]
newborns_2019 = [
  Person Rainer Unsinn Germany,
  Person Gladys Friday USA,
  ...
  ]
```

(a) The definition of `newborns_2019` results in a compile error (assuming that `...` represents a proper list of persons). Provide a corrected definition. (3)

(b) Define a function `all_names :: Country -> [First_Name]` which computes all first names of the newborns of the given country. (3)

- Duplicates should not be removed.
- You must define your function via list comprehensions.

(c) Write a function `nub :: Eq a => [a] -> [a]` that removes duplicates within a finite list. (3)

- You must not use recursion, but should use `foldl` or `foldr` and `filter` or `elem`.

(d) Write a function `preferred_names :: Country -> [First_Name]` that creates a sorted list of first names of newborns of the given country, where the most popular name comes first in the list. (7)

- There is no restriction on how to program this part.
- You may use `nub`, `all_names` and sorting algorithms like `sort :: Ord a => [a] -> [a]` or `sortBy :: (a -> a -> Ordering) -> [a] -> [a]`.

Exercise 4: I/O and Modules 7

Consider the following Haskell module.

```
module Factorial(fact) where

fact :: Integer -> Integer
fact n
  | n == 0 = 1
  | n > 0  = n * fact (n - 1)
```

Write a Haskell program (outside of the module `Factorial`) which repeatedly asks the user for a natural number n and then prints the factorial of n.

- The program should only stop, once number 0 has been entered (or any kind of error occurs).
- The program should be compilable via `ghc --make`.