EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Functional Programming
Wintersemester 2021/2022
Prof. Dr. T. Grust, D. Hirn

Assignment 1 (29.10.2021)

Handin until: Friday, 05.11.2021, 00:00

Exercise 1: Types **[8 Points]**

Please answer the following questions about Haskell's type system.

1. Consider the following types:

   (a) `a -> b -> c -> d`

   (b) `a -> (b -> c) -> d`

   (c) `a -> b -> (c -> d)`

   Which pairs of types are equivalent and which are not? Explain.

2. Can you give multiple definitions of a function of type `(a, b) -> a` that **behave differently**, that is, return different values for the same argument? Explain briefly. Assume that your function actually has to return a value (i.e. no crashes, no infinite loops and recursions).

3. Consider a function of type `[a] -> a`. Based on that *polymorphic* type definition, could it be a function which

   (a) ... returns the largest element of the list?

   (b) ... computes the sum of all list elements?

   (c) ... returns a *constant* value?

   (d) ... performs I/O operations (e.g. prints a value to the terminal)?

   Explain your answers.

4. The following function is supposed to extract the first character from a given string.

```haskell
getFirstLetter :: [Char] -> [Char]
getFirstLetter s = head s
```

   Fix any type errors.

5. Given the the functions `fst :: (a, b) -> a` and `snd :: (a, b) -> b`, derive the type of the following expression:

```haskell
snd . snd . fst
```

Exercise 2: Finger Exercises **[9 Points]**

1. Define an infix operator that implements logical implication. You can use the Boolean operators `(&&)`, `(||)`, `not`, or a conditional expression (`if` e_1 `then` e_2 `else` e_3). The new operator's precedence should be less than the three Boolean operators' above. Please give some example expressions to show this behavior.

```
(==>) :: Bool -> Bool -> Bool
```

2. Define a function `distance` to calculate the Euclidean distance between to Points $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$.

```
distance :: (Double, Double) -> (Double, Double) -> Double
```

3. Write a function `gcdEuclid` i j, such that computes the greatest common divisor of two integers $i, j > 0$ using Euclid's algorithm[1].

```
gcdEuclid :: Int -> Int -> Int
```

   Include a brief comment on how your implementation would behave if parameters i or j are ≤ 0.

Exercise 3: Safe Head **[3 Points]**

Consider the Haskell function `head :: [a] -> a`, which returns the first element of a list. `head` is not able to return a value if the list is empty. `Haskell` would report an error at runtime.
Now imagine the function `headMaybe` with the following type:

```
headMaybe :: [a] -> Maybe a

> headMaybe [1, 2, 3]
Just 1
> headMaybe "abc"
Just 'a'
> headMaybe []
Nothing
```

`headMaybe` returns `Nothing` on failure and `Just` x on success, where x is the head of the argument list.
Define the function `headMaybe` so that it behaves as in the description above.
Hint: You can use the built-in predicate `null :: [a] -> Bool` to test whether a given list is empty.

[1] https://en.wikipedia.org/wiki/Euclidean_algorithm#Implementations