Duy Khoi Nguyen Mtr. 630305

FP-Aufgaben 05

Aufgabe 1

Welchen Typ haben die folgenden Funktionen?

- **(<)**
- **(++ [1..10])**
- **f x = (\x -> x + 1) x**

```
- (<)            :: Ord a => a -> a -> Bool
- (++[1..10])    :: Num a => [a] -> [a]
- f              :: Num a => a -> a
```

Aufgabe 2

Welchen Typ haben die folgenden Funktionen?

- **f x y = (== 3) x y**
- **g (x:xs) = (\x -> (++) [x] xs)**

```
f x y = (== 3) x y
Antwort: (== 3) steht für (\x -> x == 3),
         (\x -> x == 3) nimmt aber nur ein Argument x an.

g (x : xs) = (\x -> (++) [x] xs)
Antwort: Es funktioniert vom Compiler her.
         Bespiel: g [1,2,3,4] 9 ergibt [9,2,3,4]
         Die x als erstes Element in einer List gepattern matched
         und das x in der Anonymen Funktion sind nicht dieselben,
         die gleiche Benennung der variablen macht es sehr
         unleserlich. Es gibt kein Case für eine leere List [].
```

Aufgabe 3

a)

Mittels Pattern-Matching schreiben Sie eine Funktion, die die n-te Potenz einer Zahl liefert:

potenz :: Integer → Integer → Integer

Überlegen Sie wie man diese Funktion mit einem iterativen und auch rekursiven Prozess formulieren kann.

```
-- recursive
potenz :: Integer -> Integer -> Integer
potenz base 0 = 1
potenz base exp = base * potenz base (exp -1)

-- iterative
potenz_it :: Integer -> Integer -> Integer
potenz_it base 0 = 1
potenz_it base exp = potenz_it' base base exp
  where
    potenz_it' akk _ 1 = akk
    potenz_it' akk base exp = potenz_it' (base * akk) base (exp -1)
```

b)

Definieren Sie einen polymorphen Typ für die Funktion potenz

```
potenz_poly :: Integral a => a -> a -> a
potenz_poly base 0 = 1
potenz_poly base exp = base * potenz_poly base (exp -1)
```

c)

Finden Sie zwei partielle Applikationen zur Berechnung von „**quadrat**“ und „**dritten Potenz**“ einer Zahl.

```
quadrat :: Double -> Double
quadrat = (** 2)

quadrat2 :: Double -> Double
quadrat2 = (\base -> base ** 2)

dritte_Potenz :: Double -> Double
dritte_Potenz = (** 3)

dritte_Potenz2 :: Double -> Double
dritte_Potenz2 = (\base -> base ** 3)
```

b)

Welche polymorphen Typen haben Ihre partiellen Applikationen?

```
:: RealFloat a => a -> a
```