6. Oktober 2022

Listen in Haskell

Ihr habt jetzt eure ersten Funktionen in Haskell entwickelt. Das Prinzip des Programmierens ist, dass ich eine Funktion (also einen Namen) festlege, der eine bestimmte Anzahl Parameter bekommt (die Variablen) und ich dann rechts vom '=' angebe, was zu tun ist. Ganz leicht!! Oder?

Da die Syntax ungewohnt und neu ist, hier nochmal einige Beispiele:

Listing 1: Einige erste kleine Programme in Haskell

```
text = "Hallo Welt!"      -- Definition einer konstanten Funktion text
summe x y = x + y
quad x = x*x
hochvier x = quad (quad x)      -- Funktionen als Parameter moeglich

potenz x y = x * potenz x (y-1)    -- Rekursion

maxi x y | x > y = x     -- | ein Guard
         | otherwise = y

max3 x y z  | (x > y) && (x > z) = x
            | otherwise = z

-- wievielgleiche liefert zu drei gegebenen Zahlen die Anzahl gleicher Zahlen
-- Loesung siehe Unterricht

-- Gesucht ist eine Funktion, die den groessten gemeinsamen Teiler (ggT) zweier
    Zahlen bestimmt.
ggt:: Integer -> Integer -> Integer
ggt m n | m == n     = m
        | m > n      = ggt (m-n) n
        | otherwise = ggt (n-m) m

-- anders mit Kontrollstrukturen
ggt'  m n =
       if m == n then m else
              (if m > n then ggt' (m-n) n else ggt' (n-m) m )

--fuenfmalplusquadrat tut das, was der Name aussagt ;-)

tupel = (4,"Hallo") -- neuer Typ mit gemischten Daten

-- Fakultaetsfunktion, so wie bekannt
fac:: Integer ->Integer
fac 0 = 1
fac n = n * fac (n-1)

-- fib n liefert die n-te Fibonacchi-Zahl
fib:: Integer -> Integer
fib 0 = 0
fib 1 = 1
fib n = fib (n-1) + fib (n-2)
```

Der Listentyp

Listen sind Sammlungen von Daten gleichen Typs. Z. B. ist [3,5,2] eine Liste von `Integer`. Als Typ `[Integer]`. Auch Wörter sind Listen. So ist `wort = „Hallo“` eine Liste von `Char` der Form `wort = ['H','a','l','l','o']`. Die leere Liste wird mit [] dargestellt.
Für Listen gibt es nur eine Operation, auf die alle anderen aufbauen. Es ist der „Cons-“ Operator :. Er fügt vorne an die Liste ein Element an. So ist 1:[3,5,2] = [1,3,5,2]. Mit `(x:xs)` wird ein sogenanntes Muster angegeben, welches eine Liste in Kopf x und Rest xs zerlegt. Eine Möglichkeit, eine Liste vom Kopf her zu zerlegen.

Listing 2: Ein erstes Programm mit Listen

```
laenge:: [Integer] -> Integer
laenge [] = 0
laenge (x:xs) = 1+laenge xs
```

Aufgabe: Entwickle eine Funktion `istleer` die überprüft, ob eine gegebene Liste leer ist oder nicht.

Da Listen lediglich aus Kopf und Rest bestehen, kann man die Elemente einer Liste sehr leicht rekursiv durchlaufen. Möchte man z.B. jedes Element einer Liste von Zahlen verdoppeln, so kann man das so realisieren:

Listing 3: Ein erstes Programm mit Listen

```
verdoppeln:: [Integer] -> [Integer]
verdoppeln [] = []
verdoppeln (x:xs) = (2*x):verdoppeln xs
```

Aufgaben: Gesucht sind die folgenden Funktionen (Signatur nicht vergessen)

a) `zwischensetzen` setzt zwischen jedes Element der Liste ein gegebenes
Bsp: `zwischensetzen 'a' ['c','d','f','t']` ⇝ `['c','a','d','a','f','a','t']`

b) `letztes` liefert das letzte Element einer Liste

c) `verbinden` fügt zwei Listen aneinander

d) `erste` bekommt einen Parameter m und gibt die ersten m Elemente, soweit vorhanden, aus
Bsp: `erste 3 ['c','d','f','t','a']` ⇝ `['c','d','f']`

e) `letzten` gibt die letzten m Elemente aus

f) `zaehlen` zählt das Vorhandensein bestimmter Elemente in der Liste
Bsp: `zaehlen 'd' ['c','d','f','t','a','d']` ⇝ `2`

g) `ohnevokal` gibt einen übergebenen String ohne Vokale zurück

h) `ersetzen` arbeitet wie `ohnevokal`, nur werden bestimmte Elemente durch andere ersetzt
Bsp: `ersetzen 3 8 [2,5,6,4,5,3,4,2,1,3,4,5,6,7,8]` ⇝ `[2,5,6,4,5,8,4,2,1,8,4,5,6,7,8]`