RHEINISCH-
WESTFÄLISCHE
TECHNISCHE
HOCHSCHULE
AACHEN

LEHR- UND FORSCHUNGSGEBIET INFORMATIK II
RWTH Aachen · D-52056 Aachen · GERMANY
http://www-i2.informatik.rwth-aachen.de/lufgi2

Prof. Dr. Jürgen Giesl
Darius Dlugosz, Thomas v. d. Maßen, Antje Nowack

Übung *Informatik I - Programmierung* - Blatt 12

(Lösungsvorschlag)

Aufgabe 1

Listen können mit Hilfe der Listenkonstruktoren `[]` und `:` aufgebaut werden. Dabei steht `[]` für die leere Liste und `x:xs` für eine Liste mit dem Kopfelement `x` (das erste Element der Liste `x:xs`) und der Restliste `xs`. Alle Listenelemente müssen vom gleichen Typ sein. $[x_1, x_2, x_3]$ ist Abkürzung für $x_1 : x_2 : x_3 : []$, z.B. `[0, 1, 2, 3]` (= `0:[1, 2, 3]` = `0:(1:[2, 3])` = usw.) ist eine Abkürzung für `0:1:2:3:[]`.

(a) falsch, da
$[] : xs = [[], x_1, \ldots, x_n] \neq [x_1, \ldots, x_n] = xs$
Insbesondere ist $[] : xs = [[], x_1, \ldots, x_n]$ nicht Typkorrekt, da alle Elemente einer Liste den gleich Typ haben müssen. Hier sind $x_1, \ldots, x_n$ keine Listen, während das erste Element der Liste $[[], x_1, \ldots, x_n]$ eine Liste ist.

(b) falsch, da
$[] : xs = [[], x_1, \ldots, x_n] \neq [[], [x_1, \ldots, x_n]] = [[], xs]$
Wie in Teilaufgabe (a) ist $[] : xs = [[], x_1, \ldots, x_n]$ nicht Typkorrekt.

(c) falsch, da
$xs : [] = [xs] \neq xs$

(d) richtig

(e) falsch, da `x : y` nicht definiert ist (`y` ist keine Liste).

(f) richtig, da
$(x : xs) \texttt{ ++ } xs = [x, x_1, \ldots, x_n] \texttt{ ++ } [y_1, \ldots, y_n] = [x, x_1, \ldots x_n, y_1, \ldots, y_n]$
$= x : [x_1, \ldots x_n, y_1, \ldots, y_n] = x : (xs \texttt{ ++ } ys)$

Aufgabe 2

(a)
```
minus :: Int -> Int -> Int
minus 0 y = 0
minus x 0 = x
minus (x+1) (y+1) = minus x y
```

(b)
```
mul :: Int -> Int -> Int
mul 0 y = 0
mul (x+1) y = y + (mul x y)
```

(c)
```
div' :: Int -> Int -> Int
div' x y | x < y = 0
         | otherwise = 1 + (div' (minus x y) y)
```

(d)
```
gcd' :: Int -> Int -> Int
gcd' x y | x == 0 = y
         | y == 0 = x
         | x <= y = gcd' x (minus y x)
         | otherwise = gcd' (minus x y) y
```

(e)
```
lcm' :: Int -> Int -> Int
lcm' x y = div' (mul x y) (gcd' x y)
```

Aufgabe 3

(a)
```
sum' :: [Int] -> Int
sum' xs = subsum 0 xs
     where subsum :: Int -> [Int] -> Int
           subsum n [] = n
           subsum n (x : xs) = subsum (n + x) xs
```

(b)
```
fac :: Int -> Int
fac n = subfac 1 n
    where subfac :: Int -> Int -> Int
          subfac r 0 = r
          subfac r (n + 1) = subfac (r * (n + 1)) n
```

(c)
```
reverse' :: [Int] -> [Int]
reverse' xs = subrev [] xs
         where subrev :: [Int] -> [Int] -> [Int]
               subrev rs [] = rs
               subrev rs (x:xs) = subrev (x:rs) xs
```

Aufgabe 4

Das Programm:

```
{-
 Author: Darius Dlugosz
 Umgebung: Hugs98, Windows 2000
 Erstellt: 22.01.02
```

```
-}

{-
Die Funktion "wandle" wandelt eine in Wortdarstellung
(als String) gegebene ganze Zahl zwischen 1 und 9999 in den
zugehoerigen ganzzahligen int-Wert.
-}
wandle :: String -> Int
wandle [] = 0

wandle ('e':'i':'n':xs) = wandle' xs
       where wandle' "s" = 1
             wandle' xs  = wandle_gross xs 1

wandle ('z':'w':'e':'i':xs) = wandle_gross xs 2

wandle ('d':'r':'e':'i':xs) = wandle' xs
       where wandle' "ssig" = 30
             wandle' xs     = wandle_gross xs 3

wandle ('v':'i':'e':'r':xs) = wandle_gross xs 4

wandle ('f':'u':'e':'n':'f':xs) = wandle_gross xs 5

wandle ('s':'e':'c':'h':xs) = wandle' xs
       where wandle' ('s':xs) = wandle_gross xs 6
             wandle' xs       = wandle_gross xs 6

wandle ('s':'i':'e':'b':xs) = wandle' xs
       where wandle' ('e':'n':xs) = wandle_gross xs 7
             wandle' xs           = wandle_gross xs 7

wandle ('a':'c':'h':'t':xs) = wandle_gross xs 8

wandle ('n':'e':'u':'n':xs) = wandle_gross xs 9


wandle "elf" = 11
wandle "zwoelf" = 12

wandle "zehn" = 10
wandle "zwanzig" = 20
```

```
{-
Eine Hilfsfunktion, die den Rest-String (Suffix) der Funktion "wandle"
bearbeitet unter Beruecksichtigung der bereits (im Prefix der Wortdarstellung)
erkannten Zahl (als zweites Argument).
-}
wandle_gross :: String -> Int -> Int
wandle_gross [] x = x
wandle_gross ('u':'n':'d':ys) x = x + wandle ys
wandle_gross ("zehn") x = x + 10
wandle_gross ("zig") x = x * 10
wandle_gross ('h':'u':'n':'d':'e':'r':'t':ys) x = x * 100 + wandle ys
wandle_gross ('t':'a':'u':'s':'e':'n':'d':ys) x = x * 1000 + wandle ys
```