technische universität dortmund

FLDIT

Lehrstuhl 1
Logik in der Informatik

Prof. Dr. Peter Padawitz

Jos Kusiek (*jos.kusiek@tu-dortmund.de*) Wintersemester 2017/2018

Übungen zu Funktionaler Programmierung
Übungsblatt 2

Ausgabe: 20.10.2017, **Abgabe:** 27.10.2017 – 16:00 Uhr, **Block:** 1

Aufgabe 2.1 (4 Punkte) *Datentypen*

a) Modellieren folgende Eigenschaften mit Datentypen. Geben Sie den Attributen Namen.

- Ein Konto hat einen Kontostand und einen Kunden als Besitzer.
- Für einen Kunden werden die Daten Vorname, Name und Adresse (`String`) gespeichert.

b) Legen Sie ein Beispielkonto als Tupel mit Konstruktor an (ohne Attribute).

c) Legen Sie ein weiteres Beispielkonto als attributiertes Tupel an.

Lösungsvorschlag

```
data Account = Account { balance :: Int, owner :: Client }
  deriving Show
data Client = Client
  { name :: String
  , surname :: String
  , address :: String
  } deriving Show

client1 :: Client
client1 = Client "Max" "Mustermann" "Musterhausen"
acc1 :: Account
acc1 = Account 100 client1

client2 :: Client
client2 = Client
  { name = "John"
  , surname = "Doe"
  , address = "Somewhere"
  }
acc2 :: Account
acc2 = Account{balance = 0, owner = client2}
```

Aufgabe 2.2 (2 Punkte) *Maybe*
Mit dem Datentyp `Maybe` können partielle Funktionen definiert werden. Definieren Sie eine Divisionsfunktion `div' :: Int -> Int -> Maybe Int`, welche bei der Division durch Null `Nothing` ausgibt.

Hinweis: Sie dürfen die Funktion `div` wiederverwenden.

Lösungsvorschlag

```
div' :: Int -> Int -> Maybe Int
_ `div'` 0 = Nothing
x `div'` y = Just (x `div` y)
```

Aufgabe 2.3 (3 Punkte) *Fallunterscheidungen nach Bedingungen*
Implementieren Sie folgende Funktionen in Haskell und geben Sie die Typen der Funktionen an.

a) $collatz(n) = \begin{cases} n/2, & \text{falls } n \text{ gerade} \\ 3n+1, & \text{falls } n \text{ ungerade} \end{cases}$

Sie können die Haskell-Funktion `div` und `even` benutzen.

b) $f(x,y) = \begin{cases} y*2, & \text{falls } x = 0,\ y > 50 \\ y+2, & \text{falls } x = 0,\ y \leq 50 \\ x*2, & \text{falls } y = 0,\ x < 100 \\ x+y, & \text{sonst} \end{cases}$

Lösungsvorschlag

```
collatz :: Int -> Int
collatz n
  | even n    = div n 2
  | otherwise = 3 * n + 1

f :: (Int,Int) -> Int
f (0,y)
  | y > 50         = y*2
  | otherwise      = y+2
f (x,0) | x < 100 = x * 2
f (x,y)            = x + y
```

Aufgabe 2.4 (3 Punkte) *Präfixdarstellung*
Fügen Sie die impliziten Klammern in folgende Haskell-Ausdrücke ein. Wandeln Sie danach den Ausdruck in seine Präfixdarstellung.

a) `x + 3 * y * z`

b) `add3 1 2 3`

c) `f $ g . h x`

Lösungsvorschlag

a) Wie in der Mathematik gilt Punkt vor Strich. Bei Haskell sind Additionsoperator und Multiplikationsoperator linksassoziativ, d.h. die Operatoren werden von links nach rechts ausgewertet.

Klammern: `x + ((3 * y) * z)`
Präfix: `(+) x ((*) ((*) 3 y) z)`

b) Funktionsanwendungen sind auch linksassoziativ, werden also auch von links nach rechts ausgewertet. Funktionsanwendungen besitzen außerdem automatisch die höchste Priorität und werden vor allen anderen Operatoren ausgeführt.

Klammern: `((add3 1) 2) 3`
Der Ausdruck ist bereits in Präfixdarstellung.

c) Der Applikationsoperator (`$`) und die Komposition (`.`) sind beide rechtsassoziativ. Der Ausdruck wird also von rechts nach links ausgewertet. Die Funktionsanwendungen besitzt dabei die höchste Priorität, dann folgt die Komposition und dann der Applikationsoperator.

Klammern: `f $ (g . (h x))`
Präfix: `($) f ((.) g (h x))`