STARKER START ins Studium

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Fachbereich Informatik und Mathematik
Institut für Informatik

Vorsemesterkurs Informatik

Sommersemester 2015

Aufgabenblatt Nr. 3B

Aufgabe 1 (Listen)

a) Implementieren Sie in Haskell eine Funktion `viertes :: [a] -> a`, die das vierte Element einer Liste liefert. Hat die Eingabeliste weniger als vier Elemente, so sollte eine Fehlermeldung generiert werden.

b) Implementieren Sie in Haskell eine Funktion `ohneLetztes :: [a] -> [a]`, die für eine Liste der Länge n die ersten $n-1$ Elemente der Liste (als Liste) liefert. Für den Fall $n = 0$ soll `ohneLetztes` eine Fehlermeldung ausgeben.

c) Implementieren Sie in Haskell eine Funktion `swap3 :: [a] -> [a]`, die als Eingabe eine Liste erhält und je drei nebeneinander liegende Elemente umdreht. Z.B. soll `vertausche [1,2,3,4,5,6,7,8]` als Ergebnis die Liste `[3,2,1,6,5,4,7,8]` liefern.

Aufgabe 2 (Listen und Tupel)

Die Zuckerraffinerie Rohr & Rübe hat verschiedene Lieferungen Zuckerrüben und Zuckerrohr bekommen. Die Daten einer Lieferung bestehen aus dem Paar

(Menge in Tonnen, Preis pro Tonne in EUR)

Definieren Sie in Haskell eine Funktion, die als Eingabe eine Liste von Lieferungen erhält und den Durchschnittspreis pro Tonne Zuckerrohstoff berechnet.

Hinweis: Definieren Sie zunächst eine Funktion, die das Paar (Gesamtmenge, Gesamtwert) berechnet.

Aufgabe 3 (Strings, Rekursion)

Ein *Palindrom* ist eine Zeichenkette die vorwärts und rückwärtsgelesen, dass gleiche Wort ergibt, z.B. ANNA oder RENTNER. Implementieren Sie in Haskell eine *rekursive* Funktion `istPalindrom :: String -> Bool`, die eine Zeichenkette als String erwartet und `True` oder `False` liefert, jenachdem, ob die Eingabe ein Palindrom ist oder nicht.

Aufgabe 4 (Strings)

Implementieren Sie in Haskell eine Funktion, die einen Text als String erhält und alle Zeilen mit Zeilennummern versieht und den nummerierten Text als Ausgabe liefert.

Hinweis: Mit der vordefinierten Funktion `show` können Sie eine Zahl in einen String konvertieren.

Aufgabe 5 (Listen und Typen)

Definieren Sie in Haskell je eine Funktion, die den folgenden Typ besitzt:

- `[Bool]`
- `Bool -> [[Integer]]`
- `[Integer] -> [Integer]`
- `[[[a]]] -> a`
- `Bool -> [Bool -> Bool]`

Aufgabe 6 (Türme von Hanoi)

In der Vorlesung wurde ein Haskell-Programm erläutert, dass die Zugfolge für das „Türme von Hanoi"-Spiel berechnet (zu finden in der Datei `hanoi.hs`). Die Zugfolge ist eine Liste von Zahlpaaren der Form (x, y), wobei x und y Zahlen aus der Menge $\{1, 2, 3\}$ sind. Ein solches Paar meint: schiebe die oberste Scheibe vom Stapel x auf den Stapel y.

Ziel der Aufgabe ist es, in Haskell eine Funktion `simuliere` zu implementieren, die einen Zustand des Hanoi-Spiels und eine Zugfolge als Eingaben erhält und die einzelnen Züge simuliert.

Der Zustand des Spiel wird als 3-Tupel (start,ziel,hilf) dargestellt, wobei die einzelnen Komponenten die entsprechenden Stapel sind. Ein Stapel selbst wird als Liste von geordneten Zahlen dargestellt. Die Startsituation des „Türme von Hanoi"-Spiels mit einem Turm der Höhe `n` ist daher das Tupel `([1,2,...,n],[],[])`, da der Zielstapel und der Hilfsstapel am Anfang leer sind.

Die Ausgabe der Funktion `simuliere` ist die Liste der Zustände (beginnend mit dem Anfangszustand), die erreicht werden.

Der Typ von `simuliere` ist daher

```
([Int], [Int], [Int]) -> [(Int, Int)] -> [([Int], [Int], [Int])]
```

Ein Beispielaufruf für einen Turm der Höhe 3 ist:

```
*> simuliere  ([1,2,3],[],[]) (start_hanoi 3)
[([1,2,3],[],[]),
 ([2,3],[1],[]),
 ([3],[1],[2]),
 ([3],[],[1,2]),
 ([],[3],[1,2]),
 ([1],[3],[2]),
 ([1],[2,3],[]),
 ([],[1,2,3],[])
]
```

Implementieren Sie `simuliere` in Haskell.