

04IN1023: Grundlagen der funktionalen Programmierung

Nachklausur WiSe 2019/20

Universität Koblenz-Landau, FB4

Prof. Dr. Ralf Lämmel,

M.Sc., M.Ed. Hakan Aksu

04 November 2020

Name, Vorname	
Matrikel-Nr.	
Email@uni-koblenz.de
Studiengang	<input type="checkbox"/> BSc Inf <input type="checkbox"/> BSc CV <input type="checkbox"/>
Prüfungsversuch	<input type="checkbox"/> 1 <input type="checkbox"/> 2 <input type="checkbox"/> 3

Hiermit bestätige ich, dass ich zur Klausur angemeldet und zugelassen bin.
Eine falsche Angabe wird als Täuschungsversuch gewertet.

Unterschrift: _____

Korrekturabschnitt

Aufgabe	Punkte (0-2)
1	
2	
3	
4	
5	
6	
7	
8	
9	

1 “Simple Algorithmen”

Implementieren Sie die Funktion *countLiteral*, die einen String und einen Character einliest. Die Funktion soll die Anzahl der Vorkommen des mitgegebenen Characters im String zählen und zurückgeben.

```
tests :: Test
tests = TestLabel "Klausur" (TestList [
    countLiteral "Hallo" 'l' ~?= 2,
    countLiteral "Grundlagen der funktionalen Programmierung" 'n' ~?= 6,
    countLiteral "Ich bin ein String" 'x' ~?= 0,
])
```

```
countLiteral :: String -> Char -> Int
```

2 “Suchen und Sortieren”

Implementieren Sie die Funktion *getMin*, die einen Integer-Array einliest und das kleinste Element zurückgibt.

```
tests :: Test
tests = TestLabel "Klausur" (TestList [
    getMin [] ~?= Nothing,
    getMin [5,2] ~?= Just 2,
    getMin [4,8,1,5] ~?= Just 1
])
```

```
getMin :: [Int] -> Maybe Int
```

3 “Simple Datenmodelle”

Deklarieren Sie einen vereinfachten Datentypen zur Verwaltung einer Filmbibliothek. Ein *Film* hat einen Namen, eine Beschreibung, ein Erscheinungsjahr und eine Liste von Darstellern. Der Name und die Beschreibung sind String-Werte. Das Erscheinungsjahr ist ein Integer-Wert. Jeder Darsteller kann ein Hauptdarsteller oder Nebendarsteller mit einem Namen und einem Geburtsjahr sein.

4 “Funktionale Datenstrukturen”

Gegeben ist folgender Queue-ADT:

```
data DQueue a = Empty | Enqueue a (DQueue a)
  deriving (Eq, Show, Read)
```

```
empty :: DQueue a
empty = Empty
```

```
enqueue :: a -> DQueue a -> DQueue a
enqueue = Enqueue
```

```
dequeue :: DQueue a -> Maybe (DQueue a, a)
dequeue Empty = Nothing
dequeue (Enqueue x Empty) = Just (Empty, x)
dequeue (Enqueue x q) = Just (Enqueue x q', x')
  where
    Just (q', x') = dequeue q
```

Implementieren Sie die Funktionen *size* und *contains*.

Hinweise:

```
size :: DQueue a -> Int
```

```
contains :: DQueue a -> a -> Bool
```

5 “Funktionen höherer Ordnung”

Implementieren Sie die Funktion *sumWordsLength*, die als Parameter einen String-Array nimmt. Die Längen der Strings sollen zusammenaddiert und zurückgegeben werden. Es ist nicht erlaubt einen lokalen Scope zu benutzen (keine Hilfsfunktionen und -variablen). Anonyme Funktionen sind erlaubt!

Hinweise:

```
foldr :: Foldable t => (a -> b -> b) -> b -> t a -> b
```

```
tests :: Test
```

```
tests = TestLabel "Klausur" (TestList [  
    sumWordsLength [] ~?= 0,  
    sumWordsLength ["Hallo", "Welt"] ~?= 9,  
    sumWordsLength ["Grundlagen", "der", "funktionalen", "Programmierung"] ~?= 39  
])
```

```
sumWordsLength :: [String] -> Int
```

6 “Typ-Klassen Polymorphismus”

Implementieren Sie zwei Instanzen der Klasse Street für die Datentypen Autobahn (Freeway) und Landstraße (Highway).

```
data Freeway = Freeway String Float -- | String ist die Bezeichnung, Float ist die Streckenlänge  
data Highway = Highway String Float
```

```
class Street a  
  where  
    maxSpeed :: a -> Int
```

Die maximal erlaubte Geschwindigkeit (maxSpeed) ist eine Integer-Konstante mit den Werten:

- Autobahn: 130
- Landstraße: 100

Hier noch einige Hinweise:

```
sampleF1 = Freeway "A7" 962,7  
sampleF2 = Freeway "A3" 769  
sampleH1 = Highway "Mainzer Landstrae" 8,3
```

```
tests :: Test  
tests = TestLabel "Klausur" (TestList [  
  maxSpeed sampleF1 ~?= 130,  
  maxSpeed sampleF2 ~?= 130,  
  maxSpeed sampleH1 ~?= 100  
])
```

7 “Functors & Friends”

Betrachten Sie die folgende Datenstruktur für Binäre Bäume.

```
data BTree a = Leaf a | Node a (BTree a) (BTree a)
```

Implementieren Sie die entsprechende Instanz für die Typklasse *Functor* (nur *fmap*).

```
sample1 :: BTree Int
```

```
sample1 = Node 5 (Leaf 1) (Node 8 (Node 4 (Leaf 6) (Leaf 9)) (Leaf 2))
```

```
sample2 :: BTree String
```

```
sample2 = Node "Hallo" (Leaf "Welt") (Leaf "!")
```

```
tests :: Test
```

```
tests = TestLabel "Klausur" (TestList [
```

```
  fmap (+1) sample1 ~?= Node 6 (Leaf 2) (Node 9 (Node 5 (Leaf 7) (Leaf 10)) (Leaf 3)),
```

```
  fmap (length) sample2 ~?= Node 5 (Leaf 4) (Leaf 1)
```

```
])
```


8 “Unparsing & Parsing”

Implementieren Sie die Funktion `unparseKeyValueList`, die eine `KeyValueList` als Parameter nimmt und mithilfe der Bibliothek `HughesPJ` in folgende Form überführt:

- Zu Beginn steht das Schlüsselwort 'Liste:'
- Die Schlüssel-Wert-Paare werden untereinander aufgelistet und mit einem Zeilenumbruch getrennt.
- Das Schlüsselpaar, bestehend aus einem String- und einem Integer-Wert, wird mit einem Leerzeichen getrennt. Nach einem Doppelpunkt und einem Leerzeichen folgt der Float-Wert.

```
data DoubleKeyValueList = KeyValueList [(Key, Value)]
type Key = (String, Int)
type Value = Float

sample1 :: KeyValueList
sample1 = KeyValueList
    [ ("Apple", 3), 4.5 ], [ ("Egg", 2), 1.5 ], [ ("Sugar", 5), 3.5 ]

show (unparseKeyValueList sample1) liefert folgende Ausgabe:
Liste:
Apple 3: 4.5
Egg 2: 1.5
Sugar 5: 3.5

unparseKeyValueList :: DoubleKeyValueList -> Doc

Hier sind einige Funktionen aus der HughesPJ-Bibliothek:
text, float, int, <>, <+>, $$
```

9 “Monaden”

Die Queue Datenstruktur (Vorne wird hinzugefügt, hinten wird entfernt) beschreibt eine Route und zeigt, wann nach rechts oder links abgelenkt werden muss.

```
data DQueue = Enqueue String DQueue | Empty
```

Implementieren Sie die Funktion *evalM*, die eine DQueue als Parameter nimmt. Die Funktion soll mithilfe des Writer-Monaden zum Einen die Anzahl der "links"-Abbiegungen bestimmen und zum Anderen die einzelnen Abbiegungen (wie unten im Beispiel) loggen.

```
sample1 = Enqueue "rechts" $ Enqueue "links"  
          $ Enqueue "links" $ Enqueue "rechts" $ Enqueue "links" $ Empty
```

```
> evalM sample1  
WriterT (Identity (3,["links", "rechts", "links", "links", "rechts"]))
```

```
evalM :: DQueue -> Writer [String] Int
```