**PHILIPPS-UNIVERSITÄT MARBURG**
Fachbereich Mathematik und Informatik
Prof. Dr. R. Loogen

D-35032 Marburg
Hans Meerwein Straße
Lahnberge

# Übungen zur „Praktischen Informatik III“, WS 2005/06

**Nr. 7, Abgabe: 13. Dezember 2005 vor der Vorlesung**

---

## A. Hausaufgaben

Die Lösungen sollten grundsätzlich schriftlich, Programme zusätzlich auf Diskette oder per E-Mail an Ihre Tutorin abgegeben werden. Die Abgabe ist in Gruppen bis zu zwei Personen erlaubt.

30. Buchstabenhäufigkeit **4 Punkte**

(a) Schreiben Sie eine Funktion zum Mischsortieren, bei der die Mischfunktion als Argument übergeben wird: / 1

```
mergeSort :: ([t] -> [t] -> [t]) -> [t] -> [t]
```

Beim Mischsortieren wird die zu sortierende Folge in zwei Teilfolgen zerlegt, die rekursiv mit demselben Verfahren sortiert werden. Die zwei sortierten Teilfolgen werden dann zu einer sortierten Gesamtfolge gemischt.

(b) Definieren Sie unter Verwendung von `mergeSort` eine Funktion / 1

```
sortfold :: [(Char, Int)] -> [(Char, Int)]
```

die eine Liste von Paaren aus je einem Buchstaben und einer Zahl nach den Buchstaben sortiert und dabei Paare mit gleichen Buchstaben durch die Addition der Zahlen zusammenfasst.

*Beispiel:* `sortfold [('c',2), ('d',1), ('a',4), ('d',2), ('c',5)]`
`==> [('a',4), ('c',7), ('d',3)]`

(c) Implementieren Sie unter Verwendung der in (a) und (b) definierten Funktionen die Funktion / 2

```
frequency :: [Char] -> [(Char, Int)]
```

die zu einer gegebenen Zeichenkette eine Liste aller darin vorkommenden Zeichen zusammen mit der Häufigkeit ihres Auftretens erstellt. Die Ergebnisliste soll dabei nach der Häufigkeit sortiert sein.

*Beispiel:* `frequency "battat"  ==>  [('b',1), ('a',2), ('t',3)]`

31. Rosenbäume **4 Punkte**

Ein Rosenbaum ist ein Baum, dessen Knoten nicht notwendigerweise dieselbe Anzahl an Teilbäumen besitzen. Blätter werden durch Knoten repräsentiert, die keine Teilbäume haben:

```
data Rose a = Node a [Rose a]
```

Implementieren Sie die folgenden Funktionen:

(a) `dfo :: Rose a -> [a]` traversiert einen Rosenbaum in Tiefensuche. / 1

(b) `bfo :: Rose a -> [a]` traversiert einen Rosenbaum in Breitensuche. / 2

(c) `toB :: Rose a -> BTree a` / 1

transformiert einen Rosenbaum in einen binären Baum vom Typ `BTree a`:

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

Dabei soll gemäß dem folgenden Beispiel vorgegangen werden:

```
   1                      o
  /|\                    / \
 2 3 4      --->        o   4
  / \                 /   \
 5   6               o     o
                    / \   / \
                   1   2 o   6
                        / \
                       3   5
```

32. Monadische Kontrollstrukturen **4 Punkte**

(a) Drücken Sie die in Haskell vordefinierte Funktion / 1

```
sequence        :: [IO a] -> IO [a]
sequence []     =  return []
sequence (a:as) =  do { r <- a; rs <- sequence as; return (r:rs) }
```

als Instanz von `foldr` aus, ohne die do-Notation zu verwenden.

(b) Definieren Sie eine Kontrollstruktur / 2

```
repeatUntil :: (a -> Bool) -> (a -> IO a) -> a -> IO a
```

Ein Aufruf `(repeatUntil p f x)` soll die durch f bestimmte Aktion solange wiederholen, bis das Prädikat `p` für den zurückgegebenen Wert erfüllt ist. Dabei soll der Schleifenrumpf zunächst für den Startwert `x` durchlaufen werden, d.h. als erstes wird die Aktion `(f x)` ausgeführt. Die nachfolgenden Durchläufe sollen jeweils durch Anwendung von `f` auf den zuvor zurückgegebenen Wert bestimmt werden.

(c) Drücken Sie die im Skript auf Seite 54 angegebene Beispielfunktion `f` zum Einlesen und Aufsummieren von ganzen Zahlen (siehe Beispieldatei `bspMonadicIO.hs` auf der Vorlesungsseite) als Instanz von `repeatUntil` aus. / 1

---

## B. Mündliche Aufgaben

33. (a) Geben Sie Funktionen mit folgenden Typen an:
    i. `a -> Char`  ii. `Char -> a`  iii. `a -> b -> a -> (b,a)`  iv. `a -> b`

    (b) Bestimmen Sie (falls möglich) den allgemeinsten Typ der folgenden Ausdrücke:
    v. `map map`  vi. `map foldr`  vii. `foldr map`  viii. `foldr foldr`

34. Unifikation

    Lösen Sie die folgenden Typgleichungen:

    (a) `([f], Int -> (e, Int), b) = (d, c -> g, d)`

    (b) `(c -> Int, [a], b, c) = (f, d, e, f)`

    (c) `[[([Bool -> h], Bool, h)]] = [[([j], e, d -> e)]]`