**PHILIPPS-UNIVERSITÄT MARBURG** D-35032 Marburg
Fachbereich Mathematik und Informatik Hans Meerwein Straße
Prof. Dr. R. Loogen Lahnberge

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

**Nr. 8, Abgabe: 20. Dezember 2005 vor der Vorlesung**

---

## A. Hausaufgaben

35. Dekodierung — 4 Punkte

Die Funktion `caesar` (siehe Datei `bspHOF.hs` auf der Vorlesungsseite) verschlüsselt Texte nach der Caesar-Methode, d.h. durch eine zyklische Alphabetverschiebung um eine vorgegebene ganze Zahl.

(a) Schreiben Sie eine Funktion `decaesar :: Int -> String -> String`, die eine Caesarkodierung mit vorgegebener Verschiebungszahl dekodiert. / 1

(b) Definieren Sie eine Funktion `codeBreakerFreq :: String -> String`, die eine Zeichenkette in Caesarverschlüsselung dekodiert, wobei die Verschiebung dadurch ermittelt wird, dass der häufigste Buchstabe im kodierten Text mit 'e' bzw. 'E' identifiziert wird. Verwenden Sie die in Aufgabe 30 (Übungsblatt 7) entwickelte Funktion `frequency` (siehe Modul `Frequency.hs` auf der Vorlesungsseite), um den häufigsten Buchstaben eines Textes zu bestimmen. / 2

(c) Testen Sie Ihre Funktion mit dem folgenden Text (siehe Datei `mitteilung.txt` auf der Vorlesungsseite): / 1

```
Npughat: Va qre yrgmgra Ibeyrfhat ibe Jrvuanpugra, q.u. nz
Zvggjbpu, qrz 21. Qrmrzore 2005, svaqrg mh Ortvaa qre Ibeyrfhat
rvar süaszvaügvtr Yrvfghatfxbagebyyr fgngg, orv qre Obahfchaxgr
süe qvr Xynhfhe rejbeora jreqra xöaara. Qvr Grvyanuzr vfg avpug
irecsyvpugraq. Qrawravtra, qvr avpug grvyaruzra zöpugra bqre
xöaara, ragfgrura xrvareyrv Anpugrvyr.
```

36. Typklassen — 4 Punkte

Gegeben seien die folgenden Deklarationen:

```
-- verschiedene Baumdefinitionen
data BinTree a   = Leaf a  | Node (BinTree a) (BinTree a)
data LabTree l a = LLeaf a | LNode l (LabTree l a) (LabTree l a)
data STree a     = Empty   | Split a (STree a) (STree a)
data RoseTree a  = RNode a [RoseTree a]
data AExp a      = Val a | Plus (AExp a) (AExp a) | Mult (AExp a) (AExp a)
-- Typklasse Tree
class Tree t where
  subtrees :: t -> [t] -- liefert direkte Teilbaeume
```

(a) Geben Sie für jede Baumart eine Instanzendeklaration für die Klasse `Tree` an. / 1

(b) Definieren Sie die folgenden Funktionen: / 3

- `depth :: Tree t => t -> Int` bestimmt die Tiefe eines Baumes.
- `size  :: Tree t => t -> Int` bestimmt die Knotenanzahl eines Baumes.
- `dfs   :: Tree t => t -> [t]` liefert die Liste *aller* Teilbäume in Präfixordnung.

37. Unterklassen

4 Punkte

Gegeben sei folgende Deklaration einer Unterklasse der Klasse `Tree` zur graphischen Ausgabe von Bäumen:

```
class Tree t => DrawTree t where
  drawTree :: t -> String
  labTree  :: t -> String

  drawTree = unlines . drawTree' labTree
```

(a) Geben Sie für die Baumdefinitionen aus der vorigen Aufgabe Instanzendeklarationen für `DrawTree` an. Die Defaultdefinition von `drawTree` soll dabei nicht verändert werden. / 1

(b) Definieren Sie die Funktion / 3

`drawTree' :: Tree t => (t -> String) -> t -> [String]`

Das erste Argument dieser Funktion ist eine Funktion mit dem Typ `(t -> String)`, die die Zeichenkettendarstellung der Wurzelmarkierung des Argumentbaums liefert. Im Beispiel (siehe unten) wird ein Knoten ohne Markierung durch das Zeichen `@` dargestellt. Das zweite Argument von `drawTree'` ist der darzustellende Baum. Das Resultat der Funktion ist eine Liste von Zeilen, die in der Defaultdefinition von `drawTree` mittels `unlines` zu einer Zeichenkette mit `\n`-Zeichen nach jeder Zeile zusammengefügt werden.

Beispiel:

```
> drawTree (Node (Node (Leaf 1) (Leaf 2))  (Node (Leaf 3) (Leaf 4)))
--@--@--1
  |  |
  |  `--2
  |
  `--@--3
     |
     `--4
```

---

## B. Mündliche Aufgaben

38. Typinferenz

(a) Führen Sie für die folgende Funktion eine Typinferenz mit dem in der Vorlesung vorgestellten Verfahren durch:

```
span p []     = ([], [])
span p (x:xs)
  = let (ys, zs) = span p xs
    in if p x then (x:ys, zs)
       else ([], x:xs)
```

(b) Geben Sie zu jeder der nachfolgenden bedingten und unbedingten Gleichungen, die zusammen die Funktion `merge` definieren, den Typ an und leiten Sie hieraus den Typ der Funktion `merge` her.

```
merge (x:xs) (y:ys)
   | x<y        = x : merge xs (y:ys)
   | x==y       = x : merge xs ys
   | otherwise  = y : merge (x:xs) ys
merge (x:xs) [] = x : xs
merge [] (y:ys) = y : ys
merge [] []     = []
```