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

Hans-Meerwein-Straße, Lahnberge
35032 Marburg
Telefon (06421) 28-21520, Ebene D5

Informatik IIIb (Wintersemester 1999/2000)

KLAUSUR (4. Februar 2000)

Bitte in Druckschrift ausfüllen:

Nachname	
Vorname	
Matr.-Nr.	

Hinweise:

- **Bearbeitungszeit:** 90 Minuten
  Gesamtpunktzahl: 100 Punkte

  Zum Bestehen der Klausur sind **40 Punkte** erforderlich.
- Bearbeiten Sie jede Aufgabe auf einem eigenen Blatt. Alle ausgehändigten Blätter sind zurückzugeben und, soweit beschrieben, mit Namen zu versehen.
- Als Hilfsmittel sind ausschließlich die Vorlesungsmitschrift inklusive den Übungen sowie das Skript zugelassen.

Viel Erfolg!

Aufgabe	Punkte	Erreichte Punkte
1	9	
2	12	
3	10	
4	15	
5	12	
6	8	
7	10	
8	8	
9	8	
10	8	
Σ	100	
Note:		

1. Funktionsdefinitionen — 9 Punkte

   Die Funktion `numChar :: Char -> String -> Int` liefert die Anzahl der Vorkommen eines Buchstabens in einer Zeichenkette. Definieren Sie diese Funktion

   (a) mit Pattern Matching und Rekursion

   (b) mit `foldr`

   (c) mithilfe einer Listenabstraktion.

2. Binärbäume — 12 Punkte

   Gegeben seien die folgenden Definitionen:

```
data BBaum a = Empty | Node a (BBaum a) (BBaum a)

foldTree f e Empty        = e
foldTree f e (Node x l r) = f x (foldTree f e l) (foldTree f e r)

size = foldTree (\n l r -> 1 + l + r) 0
```

   (a) Geben Sie die Typen der Funktionen `foldTree` und `size` an.

   (b) Schreiben Sie eine Haskell-Funktion `allTrees`, die zu einer Zahl n eine Liste aller Binärbäume `t :: BBaum ()` mit n Knoten, d.h. `size t == n`, erzeugt. Beispielsweise sollen für $n = 2$ die folgenden Bäume erzeugt werden:

```
(Node () (Node () Empty Empty) Empty)
(Node () Empty (Node () Empty Empty))
```

3. Listenabstraktionen — 10 Punkte

   Definieren Sie folgende Funktionen unter Verwendung von Listenabstraktionen:

   (a) `nulleins :: [String]` erzeugt die Liste aller Zeichenketten der Form 0^n1^n für $n = 0, 1, 2, \ldots$ in aufsteigender Länge.

   (b) `perfectNums :: [Int]` erzeugt die Liste aller perfekten Zahlen. Eine Zahl ist perfekt, wenn sie die Summe ihrer (echten) Teiler ist. Die kleinste perfekte Zahl ist etwa 6, da $6 = 1 + 2 + 3$.

4. Funktionen höherer Ordnung — 15 Punkte

   Vektoren und Matrizen lassen sich durch Listen darstellen:

```
type Vektor a = [a]
type Matrix a = [Vektor a]
```

   Implementieren Sie die folgenden Vektor- und Matrixoperationen unter Verwendung von Funktionen höherer Ordnung:

(a) `vekAdd :: Num a => Vektor a -> Vektor a -> Vektor a`
addiert zwei Vektoren.

(b) `sProd :: Num a => Vektor a -> Vektor a -> a`
bestimmt das Skalarprodukt von zwei Vektoren.

(c) `matVek :: Num a => Matrix a -> Vektor a -> Vektor a`
multipliziert eine Matrix mit einem Vektor.

(d) `trans :: Matrix a -> Matrix a` transponiert eine Matrix.
Hinweis: Vervollständigen Sie die folgende Definition:

```
trans []     = []
trans [z]    = ...
trans (z:zs) = ...
```

(e) `matMat :: Num a => Matrix a -> Matrix a -> Matrix a`
multipliziert zwei Matrizen.

5. Typen 12 Punkte

Geben Sie die Typen folgender Ausdrücke an:

(a) `map map` (b) `map foldr` (c) `foldr map`

Geben Sie Funktionen mit folgenden Typen an:

(d) `a -> Bool` (e) `a -> a -> b -> (b,a)` (f) `a -> b`

6. Listeninduktion 8 Punkte

Beweisen Sie, daß für beliebige Typen `a, b, c`, Werte `e::c`, Funktionen `f::a->b` und `g::b->c->c` die folgende Gleichung gilt:

```
(foldr g e) . (map f) = foldr h e
```

wobei `h x y = g (f x) y` sei.

7. Normalformen 10 Punkte

Geben Sie zu jedem der folgenden Ausdrücke die Normalform und die Kopfnormalform an, sofern diese existieren.

(a) `filter (\ x -> x ‘mod‘ (2*3) == 0)`

(b) `filter (\ x -> x ‘mod‘ 3 /= 0) [1,3,5,7,9]`

(c) `map ((+1).(*2)) (from 0),` wobei `from n = n : from (n+1)`

(d) `foldr (map.(*)) [1] [1,2,3]`

8. Auswertungsstrategien — 8 Punkte

Gegeben seien folgende Definitionen:

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

dfs              :: Tree a -> [a]
dfs (Leaf x)     =  [x]
dfs (Node x l r) =  x : append (dfs l) (dfs r)

append           :: [a] -> [a] -> [a]
append []     ys =  ys
append (x:xs) ys =  x : (append xs ys)

main = dfs (Node 1 (Leaf 2) (Node 3 (Leaf 4) (Leaf 5)))
```

Geben Sie die „leftmost innermost"- und die „leftmost outermost"-Auswertung von `main` an.

9. Holländisches Flaggenproblem — 8 Punkte

Gegeben sei die folgende Typdeklaration zur Definition gefärbter Objekte:

```
data ColObject a = Red a | White a | Blue a
```

Schreiben Sie eine Funktion

```
rwb :: [ColObject a] -> [ColObject a]
```

die eine Liste gefärbter Objekte so umordnet, daß zuerst die roten Objekte, anschliessend alle weißen Objekte und zuletzt alle blauen Objekte auftreten. Die relative Ordnung gleichfarbiger Elemente soll dabei erhalten bleiben.

10. Interaktive Ein-/Ausgabe — 8 Punkte

Schreiben Sie (a) mit und (b) ohne Verwendung von `interact` eine Funktion:

```
myInteract :: (Char -> Char) -> IO ()
```

die fortlaufend Zeilen von der Standardeingabe liest, mit der übergebenen Funktion zeichenweise transformiert und die transformierten Zeilen auf die Standardausgabe schreibt.