Fachbereich Mathematik und Informatik
Institut für Informatik

AG Datenbanken / Informationssysteme
Prof. Dr. H. Schweppe

Algorithmen und Programmierung I

WS 2004 / 2005

6.4.2005

Nachklausur

Name,	Vorname,	Matrikelnr

3.1	3.2	3.3	3.4	3.5	3.6	3.7	3.8	3.9	3.10	3.11	
4	4	5	5	6	5	6	10	9	10	10	Σ

- Alle Teilnehmer(innen) von ALP1, die das Scheinkriterium noch nicht erfüllt haben, aber mindestens 50% der Übungspunkte erreicht haben, sind teilnahmeberechtigt.
- Zum Bestehen der Klausur müssen mindesten 50 % der Punkte (37) erreicht werden.
- Dauer: **16:00 – 19:00 .**
- Legen Sie Studien- und Lichtbildausweis sichtbar auf den Platz.
- Keine Unterlagen, keine elektronischen Geräte (wie Notebook, Mobiltelefon) zugelassen!
- Die Klausur umfasst **10** Seiten, davon eine weiße. Keine eigenen Blätter benutzen!
- Prüfen Sie die Vollständigkeit Ihres Exemplars.
- Hilfsmittel und Unterlagen sind nicht erlaubt.
- Abschreiben führt zu sofortigem Ausschluss aller Beteiligten (0 Punkte).
- Sofern nicht schon angegeben, muss zu jeder definierten Funktion die Signatur angegeben werden.

Ich bin einverstanden, dass mein Ergebnis unter meiner Matrikelnummer auf der Webseite von Alp1 und auf einem Listenaushang im Institut veröffentlicht wird. Die Ergebniswebseite kann nur innerhalb der FU eingesehen werden.

Alternative: keine Unterschrift und ab 15.4.2005 Ergebnis im Sekretariat R 167 erfragen.

……
Unterschrift

Aufgabe K3.1 (4 P)
Ein Palindrom ist eine Zeichenkette, die vorwärts wie rückwärts gelesen identisch ist (Beispiel: "ANNA").
Schreiben Sie eine Funktion **`isPalindrom`**, die feststellt, ob eine Zeichenkette ein Palindrom ist.

```
isPalindrom :: String -> Bool
isPalindrom x = x == reverse x
```

Name,	Vorname,	Matrikelnr

Aufgabe K3.2 (4 P)
a) Welche allgemeinste Signatur hat die folgende Funktion und was leistet sie?

```
bizzar xs = foldr (++) [] (map biz xs)
where biz x = [x]
```

```
bizzar :: [a] -> [a]
```
bizzar ist die Identitätsfunktion auf Listen.

Aufgabe K3.3 (5 P)
Gesucht ist eine endrekursive Haskell-Funktion, die die Fibonacci-Zahlen bestimmt. Die Funktion ist bekanntlich rekursiv auf den natürlichen Zahlen definiert:
Fib 0 = 0, Fib 1 = 1, Fib n = Fib (n-1) + Fib (n-2)

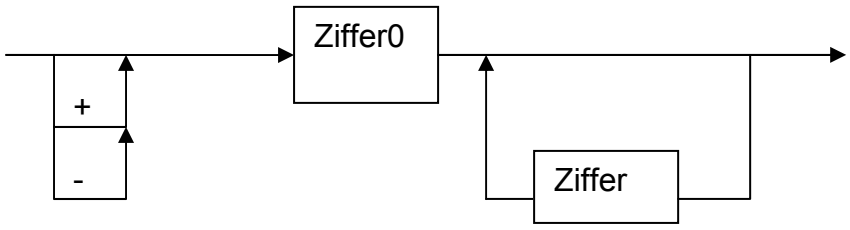
```
fib  i = fib' 0 1 i
 where fib' v n 0 = v
           fib' v n 1 =  n
          fib' v n i = fib' n (v+n) i
```

Name,	Vorname,	Matrikelnr

Aufgabe K3.4 (5 P)

Zeichenketten, die nur aus Ziffern bestehen, sollen in Zahlen vom Typ **`Integer`** umgewandelt werden. Die Zeichenkette kann ein führendes '+' oder '-' -Zeichen besitzen. Führende Nullen treten nicht auf.

a) (1 P) Geben Sie ein Syntaxdiagramm für die Zahlen an.



Ziffer0 : Ziffer 1 .. 9
Ziffer : 0 | Ziffer 0

(müssen nicht unbedingt in Syntaxdiagrammform angegeben werden.)

b) (4 P) Implementieren Sie die Funktion **`toInt :: String -> Integer`** mit Hilfe des Hornerschemas zur Auswertung von Polynomen.
Hinweis: Sie können davon ausgehen, dass
- Argumente syntaktisch korrekt im obigen Sinne sind – also keine Fehlerbehandlung nötig,
- alle Hilfsfunktionen, die man benötigt (z.B. Potenzierung) **`Integer`**-Argumente verwenden,

so dass keine Typkonflikte zwischen **`Int`** und **`Integer`** auftreten.

-- ord sei definiert

```
toInt  xs = horner 10 (map toNumber xs)
  where toNumber x = ord x - disp
        disp       = ord '0'
        horner :: Num a => a -> [a] -> a
        horner x = foldl (horn x) 0
        horn :: Num a => a -> a -> a -> a
        horn x e b = e*x + b
```

Name,	Vorname,	Matrikelnr

Aufgabe K3.5 (6 P)

Betrachtet werden Ausdrücke, die nur aus Klammern oder Leerzeichen bestehen. Klammerausdrücke sind korrekt, wenn es zu jeder öffnenden Klammer eine passende schließende gibt (z.B. `"(() ) ()"` oder `"()()()()"`, nicht jedoch `"())(()"`) .

Schreiben Sie eine Haskell-Funktion **`klammern`**, die den Wert **`True`** liefert, wenn der Klammerausdruck in diesem Sinne korrekt ist, **`False`** sonst.

Hinweis: Zählen Sie die Anzahl offener Klammern….

```
klammern :: String -> Bool
klammern xs = klammern' 0 xs
  where klammern' n []
            | n== 0      = True
            | otherwise  = False
        klammern' n (x:xs)
            | n < 0      = False
            | x == '('   = klammern' (n+1) xs
            | x == ')'   = klammern' (n-1) xs
            | x == ' '   = klammern' n xs
            | otherwise = False
```

Name,	Vorname,	Matrikelnr

Aufgabe K3.6 (5 P)
Geben Sie Bedingungen an, unter denen

a) `f . g` `g:: a -> b , f:: b -> c` (Typ Wertebereich von `g` = Typ Def.bereich von `f`)

b) `f . g` und gleichzeitig `g . f`
typkorrekt sind. Die Funktionen müssen den Typ `a -> b, b -> a` haben

c) Die Funktionen `f` und `g` seien folgendermaßen definiert:
`f (x,y) = (x,['b'..y])` und `g (x,xs) = x + length xs`
Welchen Typ hat `h = g . f` ?

d) Der Typ von `foldr1 (.) [f,g,h,k]` ist[1]
(1) `String -> String`
(2) der gleiche wie der von `f` **x**
(3) der gleiche wie der von `(.)`
(4) `[a] -> [b]`
(5) nichts von (1) – (4)

e) `f` habe den Typ `String -> String`. Dann hat `[f x | x <-xs]` den Typ
(1) `String -> String`
(2) `String`
(3) `[String]` **x**
(4) `[Char]`
(5) nichts von (1)-(4)

Aufgabe 3.7 (6 P)

Im Folgenden sei (P,V) die Spezifikation einer Funktion f

a) Die Implementierung von f genügt der Spezifikation, wenn gilt:
(1) für alle Argumente x,y,z,… P(x,y,z,..) $\wedge$ V (x,y,z…)
(2) es gibt Argumente x,y,z,.. P(x,y,z,..) $\wedge$ V (x,y,z…)
(3) es gibt Argumente x,y,z,.. P(x,y,z,..) $\Leftrightarrow$ V (x,y,z…)
(4) für alle Argumente x,y,z,… P(x,y,z,..) $\Leftrightarrow$ V (x,y,z…)
(5) für alle Argumente x,y,z,… P(x,y,z,..) $\Rightarrow$ V (x,y,z…) **x**
(6) für alle Argumente x,y,z,… P(x,y,z,..) $\Leftarrow$ V (x,y,z…)
(7) keine der Alternativen 1-6

b) Eine Spezifikation (P,V) von F heißt stärker als eine Spezifikation (P',V'), wenn gilt:
(1) P $\wedge$ V $\Rightarrow$ P' $\wedge$ V'
(2) P' $\Rightarrow$ P $\wedge$ V $\Rightarrow$ V' **x**
(3) P' $\Rightarrow$ P $\wedge$ V' $\Rightarrow$ V
(4) P $\Rightarrow$ P' $\wedge$ V $\Rightarrow$ V'
(5) Vergleich von Spezifikationen ist nicht sinnvoll

[1] [1] (Bei Multiple Choice-Aufgaben je 1 Pkt Abzug für eine falsche Antwort)

Name,	Vorname,	Matrikelnr

Aufgabe **K3.8 (10 P)**

Gesucht ist eine Funktion **`diffDays`** , die die Anzahl von Tagen zwischen zwei Daten der Form Tag/Monat/Jahr in Tagen bestimmt. Die Daten sind beide "positiv", d.h. das kleinste Datum ist der 1. Januar 01 n. Chr. .
Definieren Sie sich zunächst geeignete Datentypen. Auf dem Datums-Datentyp soll eine Ordnung definiert sein. Beachten Sie, dass der Februar in Schaltjahren 29 Tage hat. Ein Jahr ist Schaltjahr, wenn die Jahreszahl durch 4 teilbar ist, bei vollen hundert Jahren jedoch nur, wenn die Jahreszahl durch 400 teilbar ist. 1900 war also kein Schaltjahr, da zwar durch 4, aber nicht durch 400 teilbar. Dagegen waren 2000 ebenso wie 2004 Schaltjahre.

Die Funktion hat also zwei Datums-Argumente und liefert die Anzahl von Tagen zwischen den beiden Daten. (Beispiel: zwischen dem 6.4. und dem 8.4. liegt ein Tag)
Hinweis:
Da die Anzahl der Tage im Februar (und damit die Gesamtanzahl von Tagen eines Jahres) vom Jahr j abhängt, empfiehlt es sich, eine Funktion zu benutzen, die die Liste der Tage jedes Monats für ein Argument j liefert. Die kann so aussehen:

```
type Tag    = Int
type Monat  = Int
type Jahr   = Int
type Datum  = (Tag, Monat, Jahr)

-- nützliche Funktionen
mListe :: Int -> [Int]
mListe j = [31,feb,31,30,31,30,31,31,30,31,30,31]
                where feb = if schaltJahr j then 29 else 28
tageJahr j = ...

schaltJahr j = ...

eqDat, gtDat :: Datum -> Datum -> Bool
eqDat (d,m,y) (d',m',y') = d==d' && m==m' && y==y'
gtDat (d,m,y) (d',m',y')
   =  y > y' || (y==y' && m > m') || (y==y' && m==m' && d > d')

tageJahr :: Int -> Int
tageJahr j = sum (mListe j)

schaltJahr j = (j `mod`400 == 0 ) || (j `mod`4 == 0 && j `mod` 100 /= 0)

eqDat, gtDat :: Datum -> Datum -> Bool


diffTg :: Datum -> Datum -> Int
diffTg d1 d2
  | d1 `eqDat`d2   = 0
  | d1 `gtDat` d2  = diffTg d2 d1  -- erstes Argument: kleineres Datum
  | otherwise          = jahresDiff d1 d2 + tageIn d2 - tageIn d1 -1
  -- Berechnet werden die Tage der zwischen den Daten liegenden Jahre
  -- ohne das des größeren Jahres, mit dem des kleineren
  -- Dann müssen die Tage in d2 (größeres Jahr) addiert, die in d1
  -- subtrahiert werden
  -- subtrahiere 1, da die Anzahl Tage zwischen den Daten gefordert

jahresDiff (_,_,j1) (_,_,j2) = foldr (+) 0 (map tageJahr [j1..(j2-1)])
tageIn (t,m,j) = sum volleMonate + t
  where volleMonate = take (m-1) (mListe j)
```

Name,	Vorname,	Matrikelnr

Aufgabe K3.9 (9 P)

Gegeben sei der Binärbaum `huffTree` eines Huffman-Codes des Typs
`data CT a = N (CT a) (CT a) | L a`
Kodiert werden Zeichen vom Typ `Char`, ein kodierter Text ist vom Typ `[Bit]` mit
`data Bit = O | 1`

Gesucht sind Funktionen `decode:: (CT Char) -> [Bit] ->[Char]` ,
und `encode`, die eine Zeichenkette in eine Bitfolge konvertieren bzw. umgekehrt.
Hinweise:
Für die Kodierung einer Zeichenkette ist es nützlich, den Baum in eine Assoziationsliste zu verwandeln, die für jedes Zeichen seinen Huffman-Code enthält.
In der zu kodierenden Zeichenkette kommen nur Zeichen vor, die ein Codewort besitzen.
Eine diesbezügliche Fehlerbehandlung ist nicht nötig.

```
data CT  a =  N (CT  a) (CT  a) |  L a
              deriving Show
data HT  a b
  = H b (HT  a b) (HT  a b) | B b a
      deriving Show

type CTree = CT Char
type HTree = HT Char Int
type CTab  = [(Char, [Bit])]
type Histo = [(Char, Int)]

-- die folgenden Funktionen waren gesucht, ggf. ohne Typ-Alias
decode :: CTree -> [Bit] -> [Char]
decode t []       = []
decode t (x:xs)   = decodeW t (x:xs)
   where decodeW (L c) xs       = c: (decode t xs)
         decodeW (N l r) (O:xs) = decodeW l xs
         decodeW (N l r) (I:xs) = decodeW r xs

-- Baum in Assoziationsliste transformieren
transform :: CTree -> CTab
transform = convert []

convert ::[Bit] -> CTree -> CTab
convert cw (L c) = [(c,cw)]
convert cw (N l r)
  = convert (cw ++ [O]) l ++ (convert (cw ++ [I]) r )

-- codieren durch Aufsuchen jedes Zeichens und Ersetzen durch
-- Codewort
encode :: CTab -> [Char] -> [Bit]
encode ct  = concat . map (lookup' ct)

lookup' [] c = error "wrong character"
lookup' ((x,cw):ct) c
    | x==c    =   cw
    | otherwise = lookup' ct c
```

In der Klausur war nur die encode-Funktion gefragt, also die lZeilen ab "--Baum in…"

Name,	Vorname,	Matrikelnr

Aufgabe K3.10 (10 P)

Zeigen Sie durch strukturelle Induktion, dass für endliche Listen xs gilt:

Wenn ⊗ eine assoziative Operation ist und e ein Element mit der Eigenschaft e ⊗ x = x ⊗ e, dann gilt:

foldr (⊗) e xs = foldl (⊗) e xs

.

Hinweis:
Im Induktionsschritt werden Sie auf eine Gleichung der Form:
x ⊗ (foldr (⊗) e xs) = x ⊗ (foldl (⊗) e xs) stoßen. Leider ist die rechte Seite noch nicht die der Behauptung….
Die Behauptung ergibt sich aber aus folgendem Hilfsatz:
Wenn ⊗ assoziativ ist, dann gilt für alle endlichen Listen zs und alle x,y:

x ⊗ foldl (⊗) y zs = foldl (⊗) (x ⊗ y) zs

Den müssen Sie allerdings beweisen, wenn sie ihn benutzen.

Beweis:

Induktionsanfang: foldr (⊗) e [] = e = foldl (⊗) e []
Induktionsvoraussetzung: (*) für Listen der Länge n.
Induktionsschluss:

foldr (⊗) e (x:xs) = x ⊗ foldr (⊗)e xs) =	(nach Induktionsvorauss.)
x ⊗ foldl (⊗) e xs =	(nach Hilfssatz **)
foldl (⊗) (x ⊗ e) xs =	wegen e ⊗ x = x ⊗ e
foldl (⊗) (e ⊗ x) xs =	Def. von foldl
foldl (⊗) e (x: xs)	

Beweis des Hilfssatzes durch Induktion über die Länge von zs (**)

Induktionsanfang: x ⊗ foldl (⊗) y [] = x ⊗y = foldl (⊗) (x ⊗ y) zs nach Def. von foldl
Induktionsvoraussetzung: Beh. gilt für Listen zs der Länge n
Induktionsschluss:

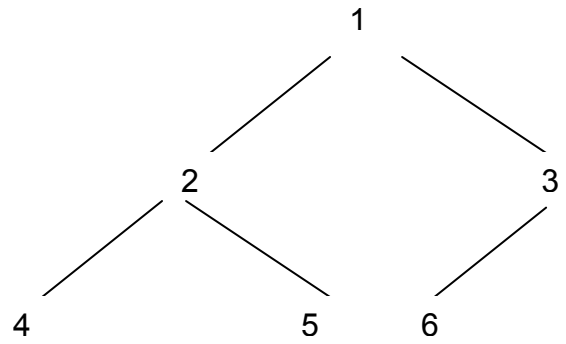
x ⊗ foldl (⊗) y (z:zs) =	Def. Von foldl
x ⊗ foldl (⊗) (y ⊗ z) zs =	Induktionsvorauss.
foldl (⊗) (x ⊗ (y ⊗ z)) zs =	Assoziativität von ⊗
foldl (⊗) ((x ⊗ y) ⊗ z) zs =	Def. von foldl
foldl (⊗) (x ⊗ y) (z:zs)	

Name,	Vorname,	Matrikelnr

Aufgabe K3.11 (10 P)

Die post- (in-, pre-)-order –Traversierung eines binären Baumes liefert jeweils eine Liste von Knotenwerten. Gegeben seien die Listen, die bei einer inorder- und bei einer preorder-Traversierung entstehen.
Gesucht ist der ursprüngliche Baum. Schreiben Sie eine Haskell-Funktion, die den Baum des Typs
`data Btree a = E | N (Btree a) (Btree a) a  --  E: leerer Baum`
aus den beiden Listen rekonstruiert.

Zu den Listen `[4,2,5,1,6,3]` und `[1,2,4,5,3,6]` gehört z.B. der Baum



Hinweis: machen Sie sich zumindest den Algorithmus klar und beschreiben Sie den ggf. verbal.

```
data Btree a =  E | N (Btree a) (Btree a) a   --  E: leerer Baum

reconstruct:: Eq a => [a] -> [a] -> Btree a
reconstruct _ []  = Empty
reconstruct p i   = N (reconstruct pl i') (reconstruct pr i'') headP
  -- p: preorder-Liste der Knoten, i: inorder-Liste
  where  i'  = takeWhile (/= headP) i
         i'' = tail (dropWhile (/= headP) i)
         headP = head p
         tailP = tail p
         pl = takeWhile (\x ->(elem x i')) tailP
         pr = dropWhile (\x ->(elem x i')) tailP
```

Zusatzblatt

Name,	Vorname,	Matrikelnr