Hertentamen Functioneel Programmeren— 8 april 2008

De nagekeken tentamens zijn in te zien bij de docent, J.H. Jongejan, Bernoulliborg kamer 366.

Opmerkingen:

- Schrijf **netjes** en duidelijk, met zwarte of blauwe pen.
- Zet op het eerste blad alle gegevens als naam, etc., en het totaal aantal ingeleverde bladen, en nummer de ingeleverde bladen.
- Lees de opgaven eerst goed door.
- Houd je programma's kort en helder, mede door verstandig gebruik te maken van standaardfuncties uit het boek (in het bijzonder uit het gedeelte over lijsten) en/of door listcomprehension.
- Motiveer je antwoorden.

1. (15 punten)

   Gegeven is de functie `collapse :: Tree a -> [a]` met de volgende eigenschappen

```
collapse Nil      = []                                  (* co.1 *)
collapse Node a l r = collapse l ++ [a] ++ collapse r   (* co.2 *)
```

   Gegeven is tevens de functie `mt :: (a -> b) -> Tree a -> Tree b` met de eigenschappen

```
mt f Nil          = Nil                                 (* mt.1 *)
mt f (Node x l r) = Node (f x) (mt f l) (mt f r) (* mt.2 *)
```

   Bewijs met volledige inductie over alle eindige bomen

```
map f (collapse t) = collapse (mt f t)
```

2. (10 punten)

   a) Geef het type en de implementatie van `zipWith`.
   b) Geef het type van de expressie `zipWith filter`.

3. (20 punten)

Een **searchtree** is een geordende boom. We gebruiken hiervoor het Haskell type:

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

a) Welke eigenschappen moet een boom hebben om geordend genoemd te mogen worden?
b) We kunnen een abstract data type maken middels:

```
module Tree
  (Tree,            -- constructor
   nil,             -- Tree a
   isNil,           -- Tree a -> Bool
   isNode,          -- Tree a -> Bool
   leftSub,         -- Tree a -> Tree a
   rightSub,        -- Tree a -> Tree a
   treeVal,         -- Tree a -> a
   insTree,         -- Ord a => a -> Tree a -> Tree a
   delete,          -- Ord a => a -> Tree a -> Tree a
   minTree          -- Ord a => Tree a -> Maybe a
  ) where ...
```

Hierbij is gegeven het standaardtype

```
data Maybe a = Nothing | Just a
```

De functies `leftSub, rightSub, treeVal` zijn hier de 'component extractors'.
a) Geef de implementatie van `rightSub`
b) Geef de implementatie van `minTree`
c) Geef de implementatie van `delete`

Het kan nodig zijn om hulpfuncties in te voeren (deze worden niet geëxporteerd).

4. (10 punten)

Implementeer een Haskell functie `lineup :: [Int] -> [[Int]]`. Deze functie gedraagt zich zoals in het volgende voorbeeld

```
lineup [5,3,4,5,3,1,2,3,3,1] = [[1,1],[2],[3,3,3],[4],[5,5]]
```

5. (25 punten)

Gegeven is een grammatica voor prefix expressies:

```
Pexp -> digit | '+' Pexp Pexp | '*' Pexp Pexp
```

a) Geef het volledig type: `type Parse a b = [a] -> ???`
b) Schrijf een parser die een invoerstring kan parsen die aan deze grammaticaregels voldoet. Gebruik daarbij de primitieve parsers

```
succeed, token, spot, alt, build, >*>, list
```

c) Breid de parser uit b) uit, zodat een boomstructuur van de input (de zogenaamde syntaxtree) wordt opgebouwd tijdens het parsen. Definieer daartoe een geschikte algebraïsche datastructuur met de naam `STree`.
d) Geef de implementatie van een functie `eval :: STree -> Int`, die zo'n boom uitrekent.

3