Tentamen Functioneel Programmeren—31 oktober 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. (20 punten)

   a) Geef het type én de implementatie van `foldr1`.
   b) Geef het type én de implementatie van `uncurry`.
   c) Geef het type én de implementatie van `zip`.
   d) Geef het type van `uncurry zip`.

2. (20 punten) Gegeven is de implementatie van `reverse`:

```
reverse [] = []
reverse (x:xs) = reverse xs ++ [x]
```

   Bewijs met volledige inductie over alle eindige lijsten `xs` dat

```
foldr (+) 0 (reverse xs) = foldr (+) 0 xs
```

   Hiervoor heb je wel een hulpbewijs nodig, over alle eindige lijsten `ys`. Bewijs die dus eerst!

```
foldr (+) 0 (ys++[x]) = (foldr (+) 0 ys) + x
```

3. (15 punten)

   Even opfrissen:

```
curry :: ((a,b) -> c) -> (a -> b -> c)
```

   a) Laat zien waarom `curry uncurry` niet door de typering van Haskell heen komt.
   b) Wat is het type van `curry id`?

4. (20 punten)

We kunnen een abstract data type maken voor een **binaire zoekboom** middels een `module Tree`:

```
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
   insert,         -- Ord a => a -> Tree a -> Tree a
   delete,         -- Ord a => a -> Tree a -> Tree a
   minTree         -- Ord a => Tree a -> a
  ) where
    data Tree a = Nil | Node a (Tree a) (Tree a)
    ...
```

a) Geef de implementatie van `leftSub`.
b) Geef de implementatie van `insert`.
c) Geef de implementatie van `minTree`.
d) Geef de implementatie van `delete`.
Het is mogelijk dat je nog één of meer extra hulpfuncties moet definiëren.

5. (20 punten)

Gegeven is dat de invoer voldoet aan de grammaticaregels:

```
E = Int | '(' E '-' E ')'
Int = D | Int D
D = '0'|'1'|..|'9'
```

a) Geef een data definitie Exp om E's te representeren.
b) Bouw een parser `pExp :: Parse Char Exp`, die de input (`[Char]`) omzet naar een Exp.
Je mag hierbij gebruik maken van:

```
type Parse a b = [a] -> (b,[a])

succeed :: b -> Parse a b
spot    :: (a -> Bool) -> Parse a a
token t = spot (==t)
alt     :: Parse a b -> Parse a b -> Parse a b
(>*>)   :: Parse a b -> Parse a c -> Parse a (b,c)
build   :: Parse a b -> (b -> c) -> Parse a c
neList  :: Parse a b -> Parse a [b]
```

Hierbij is `neList` een parser voor een niet-lege lijst van objecten.