Tentamen Functioneel Programmeren—12 november 2010

De nagekeken tentamens zijn in te zien op kamer BB 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. Definieer een functie `unique :: Int -> [Int] -> [Int]` die alle voorkomens van de `Int` parameter, behalve het eerste voorkomen, uit de argument lijst weglaat en de lijst verder ongewijzigd laat.
   Dus `unique 2 [3,2,1,2,2,5,2] = [3,2,1,5]`.

2. Gegeven zijn de functies `occurs` die het aantal voorkomens van een `Int` in een `Ntree` telt, en `collapse` die een `Ntree` omvormt tot een `[Int]`:

```
occurs : NTree -> Int -> Int
occurs NilT p = 0
occurs (Node c l r)
   | p==c = 1 + occurs l p + occurs r p
   | otherwise = occurs l p + occurs r p

collapse :: NTree -> ]Int]
collapse NilT = []
collapse (Node n l r) = collapse l ++ [n] ++ collapse r
```

   Bewijs met volledige inductie over alle eindige `NTree tr` dat geldt:

```
occurs tr x = length (filter (x==) (collapse tr))
```

   Je wordt geacht `filter` en `length` te kennen!
   Je mag gebruik maken van de volgende `filter` eigenschap:
   `filter (xs++ys) = filter xs ++ filter ys`
   Zorg voor een heldere presentatie van je bewijs.

3. Even opfrissen:

```
curry :: ((a,b) -> c) -> (a -> b -> c)
curry g x y = g (x,y)

uncurry :: (a -> b -> c) -> ((a,b) -> c)
uncurry f (x,y) = f x y
```

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

4. Een **binary searchtree** is een boom met geordende elementen: de getallen in de linker subboom zijn kleiner dan die in de root, die in de rechter subboom zijn groter (er komen geen duplicaten voor in de boom!) en bovendien zijn linker en rechter subtree zelf recursief een binary searchtree:

```
data BsTree Int = Nil | Node Int (BsTree Int) (BsTree Int)
```

We kunnen een abstract data type maken middels een `module BsTree`:

```
module BsTree
  (BsTree,          -- constructor
   nil,             -- BsTree Int
   isNil,           -- BsTree Int -> Bool
   isNode,          -- BsTree Int -> Bool
   leftSub,         -- BsTree Int -> BsTree Int
   rightSub,        -- BsTree Int -> BsTree Int
   insTree,         -- Int -> BsTree Int -> BsTree Int
   delTree          -- Int -> BsTree Int -> BsTree Int
  } where ...
```

a) Geef de implementatie van `isNode`
b) Geef de implementatie van `rightSub`
c) Geef de implementatie van `insTree`.
Je mag hierbij gebruik maken van alle functies uit de bovengenoemde signatuur.

5. (a) Wat is precies het type van `Parse a b`?
   (b) Beschouw nu invoerstrings die bestaan uit een rij digits danwel een min-teken, gevolgd door een rij digits. van de vorm "-37" Gevraagd wordt om een parser te ontwerpen die dit soort strings herkent en de betreffende `Int` waarde als resultaat oplevert. Je mag daarbij gebruik maken van de elementaire parse functies

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

Hier staat `nelist` voor het parsen van een niet-lege lijst, en `optional` voor 0 of 1 maal parsen.