9. Aufgabenblatt zu Funktionale Programmierung vom 12.12.2006. Fällig: 09.01.2007 / 16.01.2007 (jeweils 15:00 Uhr)

Themen: *Funktionale auf Listen, Programmieren mit Monaden*

Für dieses Aufgabenblatt sollen Sie Haskell-Rechenvorschriften für die Lösung der unten angegebenen Aufgabenstellungen entwickeln und für die Abgabe in einer Datei namens `Aufgabe9.hs` ablegen. Sie sollen für die Lösung dieses Aufgabenblats also wieder ein "gewöhnliches" Haskell-Skript schreiben. Versehen Sie wieder wie auf den bisherigen Aufgabenblättern alle Funktionen, die Sie zur Lösung brauchen, mit ihren Typdeklarationen und kommentieren Sie Ihre Programme aussagekräftig. Benutzen Sie, wo sinnvoll, Hilfsfunktionen und Konstanten. Denken Sie bei der Kommentierung daran, dass Ihre Lösungen auch Grundlage für das Abgabegespräch am Semesterende sind.

Im einzelnen sollen Sie die im folgenden beschriebenen Problemstellungen bearbeiten.

1. Funktionale Sprachen mit *lazy*-Auswertungsstrategie erlauben einen einfachen und eleganten Umgang mit potentiell unendlichen Listen, d.h. Listen von nicht (von vornherein) beschränkter Länge. Solche potentiell unendlichen Listen werden i.a. als *Ströme* (engl. *streams*) bezeichnet. In den folgenden beiden Aufgaben sollen Sie Rechenvorschriften für zwei Ströme angeben.

   (a) Schreiben Sie eine Haskell-Rechenvorschrift `streamPrimes :: [Integer]`, die den Strom der Primzahlen in aufsteigender Folge als Ausgabe liefert. Dabei ist 2 die kleinste Primzahl. Der Aufruf von `streamPrimes` soll also die Ausgabe `[2,3,5,7,11,13,...` liefern.

   (b) Natürliche Zahlen, in deren Primfaktorzerlegung nur die Primfaktoren 2, 3 und 5 (null- oder auch mehrmals) vorkommen, heißen *Hammingzahlen*, d.h. die Menge der natürlichen Zahlen der Form $2^i * 3^j * 5^k$, $i, j, k \geq 0$. Schreiben Sie eine Haskell-Rechenvorschrift `streamHammings :: [Integer]`, die den Strom der Hamming-Zahlen in aufsteigender Folge als Ausgabe liefert. Der Aufruf von `streamHammings` soll also die Ausgabe `[1,2,3,4,5,6,8,9,10,12,15,...` liefern.

2. Ströme lassen sich elegant zur Strukturierung und Zerlegung von Problemen in Teilprobleme einsetzen. Die Stromerzeugungskomponente ist dann der *Generator* des Stroms potentieller Lösungen, eine zweite Komponente, der sog. *Filter* wählt aus dem Strom die gewünschte Lösung aus. Dieses generelle Programmierprinzip wird auch als *Generator/Filter-Prinzip* bezeichnet.

   Benutzen Sie das Generator/Filter-Prinzip, um folgende Rechenvorschriften zu realisieren:

   (a) Schreiben Sie eine Haskell-Rechenvorschrift `sumPrimes :: Integer -> Integer`, die angewendet auf ein Argument n größer oder gleich 0 die Summe der ersten n Primzahlen liefert. Die kleinste Primzahl ist 2. Der Aufruf `sumPrimes 5` soll also die Ausgabe `28` liefern. Benutzen Sie `streamPrimes` zur Implementierung von `sumPrimes`.

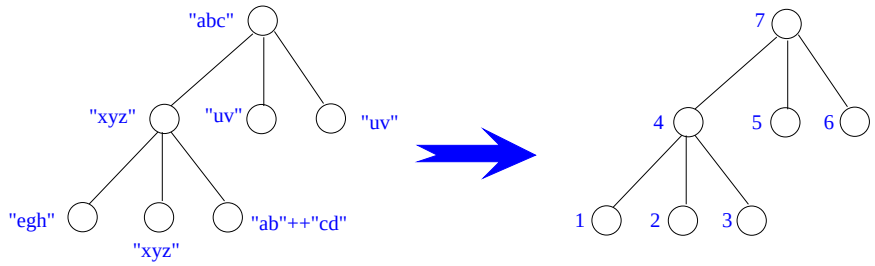
   (b) Schreiben Sie eine Haskell-Rechenvorschrift `oddHammings :: Integer -> [Integer]`, die angewendet auf ein Argument n größer oder gleich 0 die ersten n ungeraden Hamming-Zahlen liefert. Der Aufruf `oddHammings 4` soll also die Ausgabe `[1,3,5,9]` liefern. Benutzen Sie `streamHammings` zur Implementierung von `oddHammings`.

3. Gegeben sei:

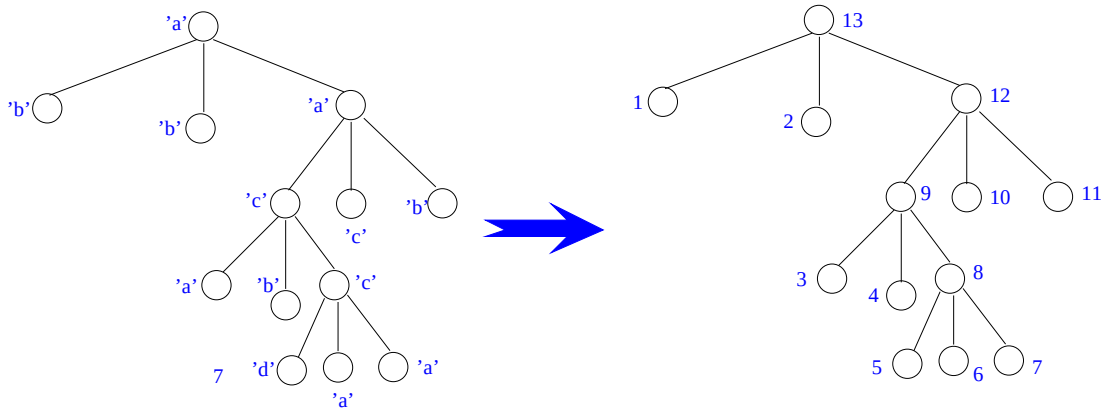
```
data TripleTree a
          = Leaf a
            | Node a (TripleTree a) (TripleTree a) (TripleTree a) deriving (Eq,Show)
```

   Gesucht ist eine Haskell-Rechenvorschrift `numberTree` mit der Signatur `numberTree :: (Eq a, Show a) => TripleTree a -> TripleTree Integer`, die angewendet auf einen Baum einen Baum zurückliefert, in dem alle Knoten und Blätter fortlaufend mit natürlichen Zahlen beginnend mit 1 benannt sind. Der Baum soll dabei in folgender Ordnung durchlaufen werden: Zunächst der linke Teilbaum, dann der mittlere Teilbaum, danach der rechte Teilbaum und schließlich die Wurzel. Die folgende Abbildung illustriert das gewünschte Abbildungsverhalten:

Bsp. 1)



Bsp. 2)



Sie könnnen diese Aufgabe wahlweise mit oder ohne Verwendung von (Zustands-) Monaden lösen.

Hinweis:

- Verwenden Sie *keine* Module. Wenn Sie Funktionen wiederverwenden möchten, kopieren Sie diese in die Abgabedatei `Aufgabe9.hs`. Andere als diese Datei werden vom Abgabeskript ignoriert.
- Bitte testen Sie Ihre Lösungen mit der aktuellen Version von Hugs auf der b1: `/usr/local/bin/hugs`

Frohe Weihnachten

und

einen guten Rutsch ins neue Jahr!