2. Aufgabenblatt zu Funktionale Programmierung vom 28.10.2005, fällig: 04.11.2005

Themen: *Funktionen auf ganzen Zahlen und Listen*

Für dieses Aufgabenblatt sollen Sie die zur Lösung der unten angegebenen Aufgabenstellungen zu entwickelnden Haskell-Rechenvorschriften in einer Datei namens `Aufgabe2.lhs` in Ihrem home-Verzeichnis ablegen. **Anders** als bei der Lösung zum ersten Aufgabenblatt sollen Sie also dieses Mal ein "**literate Script**" schreiben.

Versehen Sie wie schon in Ihrer Lösung zum ersten Aufgabenblatt auch hier alle Funktionen, die Sie zur Lösung benötigen, mit ihren Signaturen und kommentieren Sie Ihre Programme wieder aussagekräftig. Benutzen Sie, wo sinnvoll, Hilfsfunktionen und Konstanten. Im einzelnen sollen Sie folgende Problemstellungen bearbeiten:

1. In den USA sind fast ausschließlich Münzen im Wert von 1, 5, 10 und 25 Cent in Umlauf. Prinzipiell lässt sich, wenn auch im täglichen Leben wohl nur für kleinere Beträge sinnvoll, jeder auf US-Dollar lautende Betrag unter ausschließlicher Verwendung von Münzen der obigen Nennwerte begleichen. Ein Betrag von 11 Cent ließe sich etwa mit einer 10 Cent- und einer 1 Centmünze begleichen. Alternativ auch mit zwei 5 Cent- und einer 1 Centmünze, oder einer 5 Cent- und sechs 1 Centmünzen oder mit elf 1 Centmünzen.

   Wieviele Möglichkeiten gibt es prinzipiell einen vorgegebenen Dollarbetrag auf diese Art zu bezahlen?

   Schreiben Sie eine Haskell-Rechenvorschrift `anzZahlungsvarianten` mit der Signatur `anzZahlungsvarianten :: Integer -> Integer`, die diese Anzahl für nichtnegative Argumente berechnet. Dabei ist das Argument in Cent gedacht. Das heißt, der Aufruf `anzZahlungsvarianten 123` soll als Resultat die Anzahl von Varianten liefern, einen Betrag von 1.23 USD entsprechend 123 Cent mit Münzen obiger Nennwerte zu begleichen. Für negative Argumente soll die Funktion `anzZahlungsvarianten` das Resultat 0 liefern.

2. In dieser Aufgabe betrachten wir noch einmal die Quersumme nichtnegativer Dezimalzahlen. Offenbar haben die Zahlen 123, 103002 und 321 alle dieselbe Quersumme, nämlich 6. In der Folge beschränken wir uns allerdings auf Dezimalzahlen, in denen die Ziffer 0 nicht auftritt. Wir bezeichnen diese Dezimalzahlen als *0-freie* Dezimalzahlen. Zu einer gegebenen natürlichen Zahl n, $n \geq 0$, fragen wir uns, wieviele 0-freie nichtnegative Dezimalzahlen es gibt, die n als Quersumme besitzen.

   Schreiben Sie eine Haskell-Rechenvorschrift `anzZahlen` mit der Signatur `anzZahlen :: Integer -> Integer`, die diese Anzahl für nichtnegative Argumente berechnet. Für negative Argumente soll die Funktion `anzZahlen` das Resultat 0 liefern.

3. Im folgenden ist ein Anfangsstück des Pascalschen Dreiecks dargestellt:

```
                1
              1   1
            1   2   1
          1   3   3   1
        1   4   6   4   1
      1   5   10  10  5   1
    ...
```

   Dieses Dreieck ist wie folgt konstruiert. In der 1. Zeile steht eine 1. Jede weitere Zeile enthält genau eine Zahl mehr als ihre Vorgängerzeile. Ist das Dreieck bis zu einer bestimmten Zeile k konstruiert, werden die Zahlen der nachfolgenden Zeile $k+1$ dadurch gebildet, dass sukzessive die Summe zweier benachbarter Zahlen der Zeile k berechnet wird und zusätzlich am Anfang und am Ende der so entstehenden neuen Zeile eine 1 angefügt wird.

   Schreiben Sie eine Haskell-Rechenvorschrift `pascalTriangle` mit der Signatur `pascalTriangle :: Int -> Int -> [[Int]]`, die angewendet auf zwei natürliche Zahlen m und n mit $1 \leq m \leq n$ die m-te bis n-te Zeile (jeweils einschließlich) des Pascalschen Dreiecks als Liste von Listen ganzer Zahlen ausgibt. Verletzen die Argumente m und n die Beziehung $1 \leq m \leq n$ soll als Ergebnis die leere Liste abgeliefert werden. Beispiele: Der Aufruf `pascalTriangle 2 4` soll das Ergebnis `[[1,1],[1,2,1],[1,3,3,1]]` liefern, der Aufruf `pascalTriangle 4 2` das Ergebnis `[]`.

4. Schreiben Sie eine Haskell-Rechenvorschrift `teileListe` mit der Signatur `teileListe :: [Int] -> [[Int]]`, die angewendet auf eine Argumentliste ganzer Zahlen eine Liste von zwei Listen ganzer Zahlen liefert, so dass die erste Teilliste die Projektion der Argumentliste auf die nichtnegativen Elemente der Argumentliste ist und die zweite Teilliste die Projektion der Argumentliste auf die negativen Elemente der Argumentliste ist. Drei Beispiele: Angewendet auf die Liste `[1,2,-5,1,0,-6,0,1]` soll die Funktion `teileListe` das Resultat `[[1,2,1,0,0,1],[-5,-6]]` liefern, angewendet auf die Liste `[1,2,2,0]` die Liste `[[1,2,2,0],[]]`, angewendet auf die Liste `[]` die Liste `[[],[]]`.

Denken Sie bitte daran, dass Sie für die Lösung dieses Aufgabenblatts ein "literate" Haskell-Skript schreiben sollen!