7. Aufgabenblatt zu Funktionale Programmierung vom 01.12.2009. Fällig: 09.12.2009 (wg. Feiertag) / 16.12.2008 (jeweils 15:00 Uhr)

Themen: *Funktionen höherer Ordnung*

Für dieses Aufgabenblatt sollen Sie Haskell-Rechenvorschriften für die Lösung der unten angegebenen Aufgabenstellungen entwickeln und für die Abgabe in einer Datei namens `Aufgabe7.hs` ablegen. Sie sollen für die Lösung dieses Aufgabenblatts also 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.

Im einzelnen sollen Sie die im folgenden beschriebenen Problemstellungen bearbeiten.

1. Funktionale Programmiersprachen bieten Rekursion als zentrales Sprachmittel an, Wiederholungen auszudrücken. Funktionen höherer Ordnung erlauben uns, auch aus imperativen Sprachen bekannte Iterationsmuster wie for-, while- und repeat-Schleifen nachzubilden.

   In dieser Aufgabe sollen Sie Haskell-Rechenvorschriften `for`, `while`, und `repeat` höherer Ordnung mit folgenden syntaktischen Signaturen formulieren:

   - `for :: (a -> a) -> Int -> a -> a`
   - `while :: (a -> Bool) -> (a -> a) -> a -> a`
   - `repeat :: (a -> a) -> (a -> Bool) -> a -> a`

   die folgendermaßen definiert sind:

   (a) Angewendet auf eine Funktion `f`, eine nicht-negative Zahl `n` und ein Argument `z` liefert die Funktion `for` das Resultat der n-fachen Anwendung von `f` auf `z` als Ergebnis; für negative Argumente liefert sie das Argument `z` unverändert zurück.

   (b) Angewendet auf eine Wahrheitswertfunktion `b`, eine Funktion `f` und ein Argument `z` liefert die Funktion `while` das Argument `z` unverändert ab, falls `b` für `z` falsch ist, ansonsten das Resultat von `while` angewendet auf `b`, `f` und das Bild von `z` unter `f`.

   (c) Angewendet auf eine Funktion `f`, eine Wahrheitswertfunktion `b` und ein Argument `z` liefert die Funktion `repeat` das Bild von `z` unter `f` als Resultat ab, falls `b` für dieses Bild wahr ist, ansonsten das Resultat von `repeat` angewendet auf `b`, `f` und das Bild von `z` unter `f`. (Lösen Sie den Namenskonflikt zur vordefinierten Funktion `repeat` auf, indem Sie diesen Namen verstecken (siehe dazu Folie 26).)

2. "Teile und Herrsche" beschreibt ein wichtiges Organisationsprinzip von Algorithmen. Wenn wir zwei Typen `p` und `s` annehmen, deren Werte Probleme bzw. Lösungen dieser Probleme beschreiben, läßt sich das diesem Prinzip zugrundeliegende Organisationsschema in einem Funktional `divAndConquer` kapseln:

```
divAndConquer :: (p -> Bool) -> (p -> s) -> (p -> [p]) -> (p -> [s] -> s) -> p -> s
divAndConquer ind solve divide combine initProblem
    = dac initProblem
    where dac problem
              | ind problem = solve problem
              | otherwise   = combine problem (map dac (divide problem))
```

   Dabei stützt sich das Funktional `divAndConquer` auf folgende Argumentfunktionen:

   - `ind :: p -> Bool`: ...liefert `True`, wenn die als Argument übergebene Probleminstanz nicht mehr teilbar ist, `False` sonst.
   - `solve :: p -> s`: ...liefert die Lösung zu einer nicht mehr teilbaren Probleminstanz.
   - `divide :: p -> [p]`: ...liefert eine Liste von Instanzen von Teilproblemen, wenn die als Argument übergebene Probleminstanz teilbar ist.
   - `combine :: p -> [s] -> s`: ...konstruiert aus der Instanz des Ausgangsproblems und den Lösungen seiner Teilprobleme die Lösung des Ausgangsproblems.

(a) Die naive Berechnung des Wertes a^n erfordert $n-1$ Multiplikationen. Mit $O(log\ n)$ Multiplikationen kommt der Algorithmus zur schnellen Exponentiation aus. Dieser Algorithmus beruht auf folgendem Prinzip, wobei *floor* und *ceiling* zwei auf den reellen Zahlen definierte Funktionen bezeichnen, die ein reellwertiges Argument n auf die größte ganze Zahl kleiner oder gleich n bzw. die kleinste ganze Zahl größer oder gleich n abbilden. Offenbar gilt: $n = floor(n/2) + ceiling(n/2)$ sowie weiter:

$$a^n = (a^{n/2})^2 \qquad \text{falls } n \text{ gerade}$$
$$a^n = a(a^{floor(n/2)})^2 \qquad \text{falls } n \text{ ungerade}$$

Zeigen Sie, dass der Algorithmus zur schnellen Exponentiation dem "Teile und Herrsche"-Prinzip genügt und sich als Spezialisierung des Funktionals `divAndConquer` formulieren lässt. Geben Sie dazu geeignete Implementierungen der Funktionen `indExp`, `solveExp`, `divideExp` und `combineExp` an, so dass `quickExp` zur schnellen Exponentiation in Ihrer Abgabedatei insgesamt wie folgt implementiert ist:

```
quickExp :: (Integer,Integer) -> Integer
quickExp = divAndConquer indExp solveExp divideExp combineExp
```

(b) In dieser Aufgabe betrachten wir noch einmal das Problem der *Türme von Hanoi*. Zeigen Sie, dass sich auch dieses Problem als Spezialisierung des Funktionals `divAndConquer` formulieren lässt. Geben Sie dazu geeignete Implementierungen der Funktionen `indHan`, `solveHan`, `divideHan` und `combineHan` an, so dass `towOfHanoi` zur Lösung dieses Problems insgesamt wie folgt implementiert ist, wobei die Argumente dieselbe Bedeutung haben wie in Aufgabe 1 von Aufgabenblatt 3 haben.

```
towOfHanoi :: (Int,Char,Char,Char) -> [(Char,Char)]
towOfHanoi = divAndConquer indHan solveHan divideHan combineHan
```