7. Aufgabenblatt zu Funktionale Programmierung vom 24.11.2004, fällig: 01.12.2004

Themen: *Funktionen auf ganzen Zahlen, Zeichenreihen und Listen*

Für dieses Aufgabenblatt sollen Sie Haskell-Rechenvorschriften zur Lösung der unten angegebenen Aufgabenstellungen entwickeln und für die Abgabe in einer Datei namens `Aufgabe7.hs` ablegen. Sie sollen dieses mal also wieder ein "gewöhnliches" Haskell-Script schreiben. Versehen Sie 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 folgende Problemstellungen bearbeiten:

1. In der ersten Aufgabe greifen wir noch einmal die b-adischen Zahlen im Sinne von Aufgabenblatt 6 auf, die wir jetzt allerdings auch auf die Darstellung negativer Zahlen ausdehnen. Negative Zahlen werden durch ein vorangestelltes "-" Zeichen ausgezeichnet, im Zehnersystem etwa stellen wir die Zahl -42 durch die Zeichenreihe `-42` dar. Auch für negative Zahlen sind keine führenden Nullen zulässig. Eine Ausnahme machen wir lediglich für die Zahl 0, die wir durch die Zeichenreihe `0` darstellen. `-0` ist nicht zulässig, ebensowenig wie `-0042`. In gleicher Weise treffen wir diese Vereinbarungen auch für die anderen Zahlsysteme, die wir auf Aufgabenblatt 6 betrachtet haben. Auf diese Weise haben wir für jedes b-adische Zahlsystem mit $1 < b \leq 36$ für jede ganze Zahl eine eindeutige Darstellung ohne führende Nullen (letzteres bis auf die 0 selbst).

   Darüberhinaus beschäftigen wir uns in dieser Aufgabe mit der Darstellung und Konvertierung von Zahlen im (-2)-adischen Zahlsystem. Den Schlüssel dazu liefert uns der folgende Satz: Jede ganze Zahl $z \in \mathbb{Z} \setminus \{0\}$ läßt sich auf genau eine Weise in der Form

   $$z = \sum_{i=0}^{n} d_i * (-2)^i$$

   darstellen, wobei $n \in I\!N \cup \{0\}$, $d_i \in \{0,1\}$, $d_n \neq 0$. Mit der Zusatzvereinbarung die Zahl 0 auch im (-2)-adischen System durch 0 darzustellen, besitzt jede ganze Zahl eine eindeutige Darstellung ohne führende Nullen im (-2)-adischen System (auch hier mit Ausnahme der 0 selbst).

   (a) Schreiben Sie eine Haskell-Rechenvorschrift `convertMin2`, die eine (-2)-adische in eine b-adische Zahl mit $1 < b \leq 36$ konvertiert. Die Funktion `convertMin2` hat also die Signatur `convertMin2`

`:: String -> Int -> String`. Dabei bezeichnet das erste Argument die zu konvertierende Zahl und das zweite Argument die Basis des b-adischen Systems, in das das erste Argument konvertiert werden soll. Das Ein-/Ausgabeverhalten ist dann wie folgt festgelegt: Hat das zweite Argument einen Wert zwischen 2 und 36 und ist das erste Argument gültige Repräsentation einer (-2)-adischen Zahl, so wird das Argument in die Darstellung des durch das zweite Argument bestimmten b-adischen Systems konvertiert; ansonsten ist das Ergebnis die Zeichenreihe `Ungueltige Eingabe`.

(b) Schreiben Sie eine Haskell-Rechenvorschrift `convertDezToMin2`, die eine Dezimalzahl, d.h. eine 10-adische Zahl in ihre (-2)-adische Darstellung überführt. Die Funktion `convertDezToMin2` hat also die Signatur `convertDezToMin2 :: String -> String`. Dabei bezeichnet das Argument die zu konvertierende Dezimalzahl. Das Ein-/Ausgabeverhalten ist wie folgt festgelegt: Ist das Argument gültige Repräsentation einer Dezimalzahl, so wird das Argument in seine (-2)-adische Darstellung konvertiert; ansonsten ist das Ergebnis die Zeichenreihe `Ungueltige Eingabe`.

(c) In dieser Aufgabe geht es um die Addition (-2)-adischer Zahlen. Schreiben Sie dazu eine Haskell-Rechenvorschrift `addMin2` mit der Signatur `addMin2 :: String -> String -> String`, die die Summe zweier (-2)-adischer Zahlen bestimmt. Rechnen Sie dabei im (-2)-adischen System! Es ist also nicht statthaft, die Argumente in eine andere (Zahlsystem-) Darstellung zu überführen, dort zu rechnen und anschließend wieder das Ergebnis zu konvertieren. Das Ein-/Ausgabeverhalten ist dann wie folgt festgelegt: Sind beide Argumente gültige Repräsentation einer (-2)-adischen Zahl, so hat die Funktion als Ergebnis die (-2)-adische Summe dieser beiden Zahlen; ansonsten ist das Ergebnis die Zeichenreihe `Ungueltige Eingabe`.

2. Spätestens seit `HTML` erfreuen sich Auszeichnungssprachen, sog. Markup Languages, größter Popularität. In dieser Aufgabe wollen wir eine sehr einfache Auszeichnungssprache betrachten. Schlüsselsymbole sind dabei die beiden Klammernpaare { } und [ ]. Dabei zeichne das Klammerpaar { } fett gedruckten Text und das Klammerpaar [ ] normalen Text aus. Alle anderen Zeichen haben keine spezielle Bedeutung. Eine Zeichenreihe heißt *wohlgeformt* genau dann, wenn nach Streichen aller von {, }, [, und] verschiedenen Zeichen ein Klammergebirge zurückbleibt, das Wort der von der folgenden Grammatik erzeugten Sprache ist:

$$S \ ::= \ [\,] \mid \{\,\} \mid [S] \mid \{S\} \mid S\,S$$

Gesucht ist nun eine Haskell-Rechenvorschrift `isWellFormed` mit der Signatur `isWellFormed :: String -> Bool`, die eine vorgelegte Zeichenreihe darauf überprüft, ob sie wohlgeformter Ausdruck unserer Auszeichnungssprache ist. In diesem Fall liefert sie den Wahrheitswert `True` zurück, sonst `False`.