7. Aufgabenblatt zu Funktionale Programmierung vom 28.11.2012.
Fällig: 05.12.2012 / 12.12.2012 (jeweils 15:00 Uhr)

Themen: *Datenbankoperationen und kombinatorische Knobeleien*

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. Wir betrachten folgende Typen:

```
type Movie       = (Title,Regisseur,MainActors,ReleaseDate,Genre,SalesPrice)
type Title       = String
type Regisseur   = String
type Actor       = String
type MainActors  = [Actor]
type ReleaseDate = Int
data Genre       = Thriller | Fantasy | ScienceFiction | Comedy deriving (Eq,Ord,Show)
type SalesPric   = Int
type Database    = [Movie]
```

   Schreiben Sie Haskell-Rechenvorschriften für folgende Datenbankoperationen:

   - Lösche alle Duplikate in der Datenbank. Als Duplikat gilt dabei ein Eintrag, der in allen Komponenten mit einem anderen Eintrag übereinstimmt. Als Duplikat gilt insbesondere auch ein Eintrag, der sich von einem anderen nur durch die Reihenfolge der Hauptdarsteller unterscheidet. Welches der Duplikate jeweils gelöscht wird, spielt keine Rolle:
     `rm_dup :: Database -> Database`
   - Liefere alle Regisseure zusammen mit den Filmtiteln und dem Erscheinungsjahr:
     `get_rtd :: Database -> [(Regisseur,Title,ReleaseDate)]`
   - Liefere alle Regisseure zusammen mit den Filmtiteln und dem Genre, die zugleich (einer der) Hauptdarsteller in diesem Film waren:
     `get_rtg :: Database -> [(Regisseur,Title,Genre)]`
   - Liefere die Titel aller Filme zusammen mit den Listen ihrer Hauptdarsteller, die im Jahr n erschienen sind:
     `get_tad :: Database -> ReleaseDate -> [(Title,MainActors,ReleaseDate)]`
   - Liefere die Liste aller Filme zusammen mit dem Erscheinungsjahr, in denen Schauspieler s einer der Hauptdarsteller war:
     `get_atr :: Database -> Actor -> [(Actor,Title,ReleaseDate)]`
   - Verändere die Preise aller Filme des Genres g, in denen Regisseur r Regie geführt hat, um x. Für positives x führt dies zu einer Erhöhung, für negatives x zu einer Erniedrigung. Ist der sich so ergebende neue Preis kleiner oder gleich 0, wird er auf 1 gesetzt:
     `upd_dbgri :: Database -> Genre -> Regisseur -> Int -> Database`
   - Lösche alle Filme, in denen Schauspieler s einer der Hauptdarsteller war und die im Jahr j oder später erschienen sind:
     `upd_dbad :: Database -> Actor -> ReleaseDate -> Database`
   - Liefere alle Filme, die im Jahr j oder früher erschienen sind und in denen Schauspieler s nicht unter den Hauptdarstellern war:
     `get_dbda :: Database -> ReleaseDate -> Actor -> Database`
   - Sortiere die Datenbank absteigend nach Erscheinungsjahr, d.h. neuere Filme vor älteren Filmen. Sind in einem Jahr mehrere Filme erschienen, ist die Reihenfolge unerheblich, in der diese Filme nach der Sortierung aufgeführt sind:
     `sort_dbj :: Database -> Database`

- Sortiere die Datenbank aufsteigend nach Genres und innerhalb der Genres aufsteigend nach Regisseuren; hat ein Regisseur innerhalb eines Genres mehrfach Regie geführt, ist die Reihenfolge unerheblich, in der diese Filme nach der Sortierung aufgeführt sind:
  `sort_dbgr :: Database -> Database`

2. Wir betrachten folgende Typen:

```
type ListOfValues = [Integer]
type TargetValue  = Integer
type Game         = (ListOfValues,TargetValue)
data Operators    = Plus | Times | Minus | Div deriving (Eq,Ord,Show)
type Solution     = [Operators]
```

Ein Spiel ist durch eine Liste $[l_1, l_2, \ldots, l_n]$ von Werten und einen Zielwert z gegeben. Ziel des Spieles ist es, eine Folge von Operatoren $[op_1, \ldots, op_{n-1}]$ so zu finden, dass der Ausdruck

$$(\ldots((l_1\ op_1\ l_2)\ op_2\ l_3)\ldots\ l_{n-1})\ op_{n-1}\ l_n)$$

den Wert z hat. Dabei stehen die Operatoren `Plus`, `Times`, `Minus` und `Div` für die ganzzahlige Addition, Multiplikation, Subtraktion und Division. Die ganzzahlige Division ist dabei wie auf Aufgabenblatt 4 definiert, d.h., der ganzzahlige Quotient zweier Zahlen m und n mit $n \neq 0$ ist die eindeutig bestimmte Zahl q mit $m = q * n\ +\ r$, wobei $0 \leq r < n$.

Schreiben Sie eine Haskell-Rechenvorschrift `solve :: Game -> Solution`, die angesetzt auf ein Spiel g eine Lösung s bestimmt. Gibt es keine Lösung, so liefert die Funktion `solve` die leere Liste als Resultat. Gibt es mehrere Lösungen, so ist es egal, welche davon Ihre Funktion als Ergebnis bestimmt.

Folgende Beispiele zeigen das gewünschte Ein-/Ausgabeverhalten:

```
solve ([3,6,(-2),3,21],42) ->> [Plus,Plus,Div,Times]
solve ([10,6,12,15,3,13],(-52)) ->> [Times,Div,Minus,Div,Times]
solve ([42],42) ->> []
solve ([42],21) ->> []
solve ([3,2],1) ->> [Minus]
solve ([3,2],1) ->> [Div]
solve ([3,2],42) ->> []
```

Haskell Live

An einem der kommenden *Haskell Live*-Termine, der nächste ist am Freitag, den 30.11.2012, werden wir uns u.a. mit der Aufgabe *World of Perfect Towers* beschäftigen.

World of Perfect Towers

In diesem Spiel konstruieren wir Welten perfekter Türme. Dazu haben wir n Stäbe, die senkrecht auf einer Bodenplatte befestigt sind und auf die mit einer entsprechenden Bohrung versehene Kugeln gesteckt werden können. Diese Kugeln sind ebenso wie die Stäbe beginnend mit 1 fortlaufend nummeriert.

Die auf einen Stab gesteckten Kugeln bilden einen Turm. Dabei liegt die zuerst aufgesteckte Kugel ganz unten im Turm, die zu zweit aufgesteckte Kugel auf der zuerst aufgesteckten, usw., und die zuletzt aufgesteckte Kugel ganz oben im Turm. Ein solcher Turm heißt *perfekt*, wenn die Summe der Nummern zweier unmittelbar übereinanderliegender Kugeln eine Zweierpotenz ist. Eine Menge von n perfekten Türmen heißt *n-perfekte Welt*.

In diesem Spiel geht es nun darum, n-perfekte Welten mit maximaler Kugelzahl zu konstruieren. Dazu werden die Kugeln in aufsteigender Nummerierung, wobei mit der mit 1 nummerierten Kugel begonnen wird, so auf die n Stäbe gesteckt, dass die Kugeln auf jedem Stab einen perfekten Turm bilden und die Summe der Kugeln aller Türme maximal ist.

Schreiben Sie in Haskell oder einer anderen Programmiersprache ihrer Wahl eine Funktion, die zu einer vorgegebenen Zahl n von Stäben die Maximalzahl von Kugeln einer n-perfekten Welt bestimmt und die Türme dieser n-perfekten Welt in Form einer Liste von Listen ausgibt, wobei jede Liste von links nach rechts die Kugeln des zugehörigen Turms in aufsteigender Reihenfolge angibt.