7. Aufgabenblatt zu Funktionale Programmierung vom 27.11.2013.
Fällig: 04.12.2013 / 11.12.2013 (jeweils 15:00 Uhr)

Themen: *Typklassen und Datenbankoperationen*

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 noch einmal die Typklasse `SimNf` von Aufgabenblatt 6 und die Typen `NewString` und `ChSeq` von Zeichenketten.

```
class Ord a => SimNf a where
  similar    :: a -> a -> Bool
  normalform :: a -> a

newtype NewString = Nstr String deriving (Eq,Ord,Show)

data ChSeq = Empty
           | Head Char ChSeq deriving (Eq,Ord,Show)
```

   Machen Sie die Typen `NewString` und `ChSeq` zu Instanzen der Klasse `SimNf`. Zwei Zeichenketten `Nstr s` und `Nstr t` des Typs `NewString` heißen ähnlich, wenn die Anzahl der Vorkommen der Kleinvokale `'a'`, `'e'`, `'i'`, `'o'` und `'u'` paarweise in `s` und `t` gleich sind. Andere möglicherweise in `s` oder `t` vorkommende Zeichen sind für die Eigenschaft ähnlich ohne Bedeutung. Zeichenketten `Nstr "arOpFae;:6iiKlooPsAo"` und `Nstr "fe3.oiDEalKaoRoi"` sind also ähnlich. Die Normalform einer Zeichenkette `Nstr s` ist diejenige eindeutig bestimmte ähnliche Zeichenkette `Nstr t`, wobei `t` ausschließlich Kleinvokale enthält, die alphabetisch aufsteigend angeordnet sind. Die gemeinsame Normalform der Zeichenketten `Nstr "arOpFae;:6iiKlooPsAo"` und `Nstr "fe3.oiDEalKaoRoi"` ist also die Zeichenreihe `Nstr "aaeiiooo"`. Für Werte des Typs `ChSeq` sind die Relation 'ähnlich' und der Begriff 'Normalform' analog festgelegt.

2. 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 SalesPrice  = 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 Filme eines Regisseurs r aus dem Genre g, die zu einem Preis von p oder günstiger angeboten werden:
     `get_rgp :: Database -> Regisseur -> Genre -> SalesPrice -> Database`
   - Liefere alle Regisseure zusammen mit den Filmtiteln und ihrem Genre:
     `get_rtg :: Database -> [(Regisseur,Title,Genre)]`

- Liefere alle Schauspieler zusammen mit den Filmtiteln und dem Genre, in denen diese Schauspieler zugleich Regie geführt haben:
  `get_atg :: Database -> [(Actor,Title,Genre)]`
- Liefere die Titel aller Filme zusammen mit ihrem Genre und Erscheinungsjahr, in denen Schauspieler s (einer der) Hauptdarsteller war:
  `get_tgd :: Database -> Actor -> [(Title,Genre,ReleaseDate)]`
- Verändere die Preise aller Filme des Genres g, in denen Regisseur r Regie führte und zugleich eine Hauptrolle innehatte, 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_dbgrai :: Database -> Genre -> Regisseur -> Int -> Database`
- Lösche alle Filme, in denen Schauspieler s und Schauspieler t gleichzeitig unter den Hauptdarstellern waren und die Regie nicht bei Regisseur r lag:
  `rm_dbaard :: Database -> Actor -> Actor -> Regisseur -> Database`
- Liefere alle Filme, die im Jahr j oder später erschienen sind und in denen Schauspieler s und Schauspieler t nicht gleichzeitig unter den Hauptdarstellern waren:
  `get_dbda :: Database -> ReleaseDate -> Actor -> Actor -> Database`
- Sortiere die Datenbank aufsteigend nach Verkaufspreis, d.h. günstiger angebotene Filme vor teurer angebotenen. Werden mehrere Filme zum gleichen Preis angeboten, ist die Reihenfolge unerheblich, in der diese Filme nach der Sortierung aufgeführt sind:
  `sort_dbp :: Database -> Database`
- Sortiere die Datenbank aufsteigend nach Regisseuren und die Filme eines Regisseurs aufsteigend nach Genres; hat ein Regisseur innerhalb eines Genres mehrfach Regie geführt, ist die Reihenfolge unerheblich, in der diese Filme nach der Sortierung aufgeführt sind:
  `sort_dbrg :: Database -> Database`

Haskell Live

An einem der kommenden *Haskell Live*-Termine, der nächste ist am Freitag, den 29.11.2013, 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.