8. Aufgabenblatt zu Funktionale Programmierung vom 10.12.2014.
Fällig: 17.12.2014 / 14.01.2015 (jeweils 15:00 Uhr)

Themen: *Es bleibt sportlich: Funktionen auf Zahlen und anderen Datenstrukturen*

Für dieses Aufgabenblatt sollen Sie Haskell-Rechenvorschriften für die Lösung der unten angegebenen Aufgabenstellungen entwickeln und für die Abgabe in einer Datei namens `Aufgabe8.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 benötigen, 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.

Die Winterpause der Fußballbundesliga steht kurz bevor. Wer überwintert als Herbstmeister? Wer hat 3 Spieltage vor der Pause rechnerisch noch Chancen Platz 1 oder Platz 2 zu erreichen? Sportkommentatoren empfehlen in dieser Situation üblicherweise den Rechenschieber auszupacken. Tatsächlich hilft der Rechenschieber für die Lösungsfindung überhaupt nicht. Anders als geeignete Haskell-Rechenvorschriften, die wir deshalb schreiben wollen, um diese Chancen auszuloten. Dabei benutzen wir folgende Modellierung:

```
data Verein       = Sturm | WAC | Austria | WrNeustadt | RBSbg
                  | Groedig | Rapid | Admira | Ried | Altach deriving (Eq,Ord,Show)

type Spiel        = (Verein,Verein)
newtype Spieltag  = St (Spiel,Spiel,Spiel,Spiel,Spiel) deriving (Eq,Show)

data Restprogramm = Rp Spieltag Spieltag Spieltag

type Punkte       = (Verein -> Nat)
type Budget       = (Verein -> Nat)
```

wobei `Nat` die bereits bekannte Darstellung für natürliche Zahlen ist:

```
data Nat         = Z | S Nat deriving (Eq,Ord,Show)
```

Weiters verwenden wir den Typ

```
data Herbstmeister = NochOffen | AlsHMstehtfest Verein deriving (Eq,Show)
```

Wie in der richtigen Bundesliga gehen wir davon aus, dass es für die siegreiche Mannschaft eines Spiels drei Punkte gibt, für die unterlegene Mannschaft keinen Punkt. Bei einem Unentschieden erhalten beide Mannschaften je einen Punkt. Ein Wert des Typs `Punkte` repräsentiert den erreichten Punktestand nach Abschluss des viertletzten Spieltags vor Beginn der Winterpause. Aufgrund von Entscheidungen am grünen Tisch zu weiter zurückliegenden Spieltagen mag vor Beginn des drittletzten Spieltags jeder Punktestand möglich sein. Dass eine Mannschaft mit 500 Punkten die Tabelle alleine anführt ebenso wie dass alle Mannschaften mit je 27 Punkten die Tabelle gemeinsam anführen. Bei Punktegleichheit zwischen zwei Mannschaften entscheidet (statt wie in der richtigen Bundesliga das Torverhältnis) das niedrigere Budget über den besseren Tabellenplatz. Dabei gehen wir davon aus, dass die Budgets aller Mannschaften paarweise verschieden sind, eine Bedingung, von der wir annehmen, dass sie die Liga vor Beginn überprüft und ggf. sicherstellt.

Insgesamt gelten folgende Gültigkeitsannahmen:

- Ein Budget `b :: Verein -> Nat` ist gültig genau dann, wenn die Budgets aller Vereine definiert und paarweise verschieden sind.
- Ein Spiel `(v1,v2)` ist gültig genau dann, wenn `v1` und `v2` verschieden sind.
- Ein Spieltag ist gültig genau dann, wenn alle Spiele des Spieltags gültig sind und jeder der zehn Ligavereine in genau einem der Spieltagsspiele als Heim- oder Auswärtsmannschaft gesetzt ist.

- Ein Restprogramm (aus drei Spieltagen) ist gültig genau dann, wenn jeder der drei Spieltage gültig ist und die Spiele aller drei Spieltage paarweise verschieden sind.

Für die letzten drei Spieltage nehmen wir weiters an, dass keine weiteren Entscheidungen am grünen Tisch getroffen werden. Allein die möglichen Spielausgänge der letzten drei Spieltage vor der Winterpause zählen.

1. Schreiben Sie Wahrheitswertfunktionen, die überprüfen, ob ein Budget, ein Spiel, ein Spieltag, ein Restprogramm gültig sind:

   (a) `isValidBudget :: Budget -> Bool`

   (b) `isValidSpiel :: Spiel -> Bool`

   (c) `isValidSpieltag :: Spieltag -> Bool`

   (d) `isValidRestprogramm :: Restprogramm -> Bool`

2. Erstellen Sie eine Tabelle gemäß Punktestand und Budget. Ist das Budgetargument nicht gültig, so sehen Sie eine Fehlerbehandlung gemäß Panikmodus (siehe Kapitel 15.1 der Vorlesung) vor und brechen Sie die Auswertung mit Ausgabe der Zeichenreihe *Ungueltige Eingabe* ab.

   (e) `mkTabelle :: Punkte -> Budget -> [Verein]`
   Je besser ein Verein ist, desto weiter links steht er in der Ergebnisliste. Der Tabellenführer bildet also das Kopfelement der Liste, das Tabellenschlusslicht das letzte Element.

3. Steht schon ein Team als Herbstmeister fest? Welches Team kann noch aus eigener Kraft Herbstmeister werden und als Tabellenerster überwintern? Welche Teams haben bei günstigem Ausgang auch von Spielen ohne eigene Beteiligung noch Chancen Herbstmeister zu werden? Welche Teams haben noch Chancen Vize-Herbstmeister zu werden? Schreiben Sie entsprechende Haskell-Rechenvorschriften zur Beantwortung dieser Fragen. Sollte eine dieser Funktionen mit einem nicht gültigen Budget oder einem nicht gültigen Restprogramm aufgerufen werden, so sehen Sie eine Fehlerbehandlung gemäß Panikmodus vor und brechen Sie die Auswertung mit Ausgabe der Zeichenreihe *Ungueltige Eingabe* ab.

   (f) `hm_fix :: Punkte -> Budget -> Restprogramm -> Herbstmeister`

   (g) `hm_ausEigenerKraft :: Punkte -> Budget -> Restprogramm -> [Verein]`

   (h) `hm_alleMitRechnerischerChance :: Punkte -> Budget -> Restprogramm -> [Verein]`

   (i) `vhm_alleMitRechnerischerChance :: Punkte -> Budget -> Restprogramm -> [Verein]`

   Enthält die Resultatliste einer Funktion mehr als einen Verein, so sollen die Vereine nach aufsteigendem Budget geordnet sein, d.h. ein Verein steht um so weiter links in der Ergebnisliste je kleiner sein Budget ist.

Hinweis:

- Verwenden Sie *keine* Module. Wenn Sie Funktionen wiederverwenden möchten, kopieren Sie diese in die Abgabedatei `Aufgabe8.hs`. Andere Dateien als diese werden vom Abgabeskript ignoriert.

Haskell Live

An einem der kommenden *Haskell Live*-Termine, der nächste ist am Freitag, den 12.12.2014, werden wir uns, wenn es die Zeit erlaubt, u.a. mit der Aufgabe *Tortenwurf* beschäftigen.

Tortenwurf

Wir betrachten eine Reihe von $n + 2$ nebeneinanderstehenden Leuten, die von paarweise verschiedener Größe sind. Eine größere Person kann stets über eine kleinere Person hinwegblicken. Demnach kann eine Person in der Reihe so weit nach links bzw. nach rechts in der Reihe sehen bis dort jemand größeres steht und den weitergehenden Blick verdeckt.

In dieser Reihe ist etwas Ungeheuerliches geschehen. Die ganz links stehende 1-te Person hat die ganz rechts stehende $n + 2$-te Person mit einer Torte beworfen. Genau p der n Leute in der Mitte der Reihe hatten während des Wurfs freien Blick auf den Tortenwerfer ganz links; genau r der n Leute in der Mitte der Reihe hatten freien Blick auf das Opfer des Tortenwerfers ganz rechts.

Wieviele Permutationen der n in der Mitte der Reihe stehenden Leute gibt es, so dass gerade p von ihnen freie Sicht auf den Werfer und r von ihnen auf das Tortenwurfopfer hatten?

Schreiben Sie in Haskell oder einer anderen Programmiersprache ihrer Wahl eine Funktion, die zu einer vorgegebenen Zahl $n \leq 10$ von Leuten in der Mitte der Reihe, davon p mit $1 \leq p \leq n$ mit freier Sicht auf den Werfer und r mit $1 \leq r \leq n$ mit freier Sicht auf das Opfer, diese Anzahl von Permutationen berechnet.