3. Aufgabenblatt zu Funktionale Programmierung vom 04.11.2015. Fällig: 18.11.2015 / 25.11.2015 (jeweils 15:00 Uhr)

Themen: *Rekursive Funktionen über Zahlen, Zeichenreihen und Tupeln*

Für dieses Aufgabenblatt sollen Sie die zur Lösung der unten angegebenen Aufgabenstellungen zu entwickelnden Haskell-Rechenvorschriften in einer Datei namens `Aufgabe3.lhs` im home-Verzeichnis Ihres Accounts auf der Maschine g0 ablegen. **Anders** als bei der Lösung zu den ersten beiden Aufgabenblättern sollen Sie dieses Mal also ein "**literate Script**" schreiben. Versehen Sie 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.

1. Wir betrachten eine Variante der Wochentagsaufgabe von Aufgabenblatt 2. Ist bekannt, auf welchen Wochentag irgendein Tag eines Jahres fällt und ob es sich bei diesem Jahr um ein Schaltjahr handelt oder nicht, so steht für jeden Tag dieses Jahres fest, auf welchen Wochentag es fällt. Diesen Wochentag wollen wir für jedes Datum eines Jahres berechnen können. Zur Modellierung dieser Aufgabe in Haskell verwenden wir wieder folgende Typsynonyme:

   ```
   type Wochentag     = String
   type Schaltjahr    = Bool
   type Tag           = Int
   type Monat         = Int
   type Datum         = (Tag,Monat)
   ```

   Ein Paar (t, m) bezeichnet ein gültiges Datum, falls $m \in \{n \mid 1 \leq n \leq 12\}$ gilt und folgende Implikationen gelten:

   $$\begin{aligned} m \in \{1,3,5,7,8,10,12\} &\Rightarrow t \in \{n \mid 1 \leq n \leq 31\} \\ m \in \{4,6,9,11\} &\Rightarrow t \in \{n \mid 1 \leq n \leq 30\} \\ m = 2 \wedge \mathit{schaltjahr} &\Rightarrow t \in \{n \mid 1 \leq n \leq 29\} \\ m = 2 \wedge \neg \mathit{schaltjahr} &\Rightarrow t \in \{n \mid 1 \leq n \leq 28\} \end{aligned}$$

   wobei der Wahrheitswert *schaltjahr* bzw. seine Negation angeben, ob ein Schaltjahr vorliegt oder nicht.

   Schreiben Sie eine Haskell-Rechenvorschrift `wochentag3 :: Datum -> Wochentag -> Schaltjahr -> Datum -> Wochentag`, die für jedes Datum des Jahres berechnet, auf welchen Wochentag es fällt: Angewendet auf ein Datum (t, m), einen Wochentag wt, wobei (t, m) auf wt fällt, einen Wahrheitswert sj, ob es sich um ein Schaltjahr handelt, und ein Datum (t', m') liefert `wochentag3` denjenigen Wochentag, auf den das durch (t', m') bestimmte Datum in diesem Jahr fällt. Ist eines der Argumente nicht gültig, so liefert `wochentag3` die Zeichenreihe `"Falsche Argumente"` zurück. Fällt das durch (t', m') bezeichnete Datum auf den Tag vor Sonntag, darf die Rechenvorschrift `wochentag3` wahlweise den Wert `"Samstag"` oder `"Sonnabend"` als Ergebnis liefern.

2. Gegeben ist eine Liste l von Zeichenreihen. Gesucht ist die Anzahl paarweise verschiedener Zeichenreihen in l, die mindestens 3 Kleinvokale enthalten. Schreiben Sie eine Haskell-Rechvorschrift `pwv :: [String] -> Int`, die angewendet auf eine Liste l von Zeichenreihen diese Anzahl berechnet. (Eine Liste von Elementen heißt *paarweise verschieden*, wenn je zwei Elemente verschieden voneinander sind. Die Liste $[34, 12, 34, 17, 34, 12]$ über ganzen Zahlen enthält 3 paarweise verschiedene Elemente, die Elemente 34, 12 und 17.)

3. Schreiben Sie eine Haskell-Rechenvorschrift `streamline :: [String] -> Int -> [String]` mit folgender Funktionalität: Angewendet auf eine Liste l von Zeichenreihen und eine echt positive ganze Zahl n, $n > 0$, entfernt `streamline` aus dem Argument l alle Elemente, die nicht genau n Vorkommen haben. Die relative Reihenfolge der verbleibenden Elemente bleibt dabei erhalten. Ist das Argument n kleiner oder gleich 0, so liefert `streamline` die leere Liste als Resultat.

4. Einige Bundesstaaten in den USA verwenden Autokennzeichen der Form `XYZ abc`, wobei `X`, `Y` und `Z` jeweils Großbuchstaben und `a`, `b` und `c` jeweils Ziffern sind. Die Nummerierung erfolgt fortlaufend. Auf `AAA 997` folgt `AAA 998`, auf `AAA 999` folgt `AAB 000`, auf `ABC 999` folgt `ABD 000`, auf `ABZ 999` folgt `ACA 000`. Auf `ZZZ 999` folgt (für diese Aufgabe) zyklisch `AAA 000`. Mit dieser zyklischen Festlegung hat jede Kennzeichenkombination eindeutig ein Nachfolger- und ein Vorgängerkennzeichen.

   Schreiben Sie zwei Haskell-Rechenvorschriften

   - `nf :: Kennzeichen -> Kennzeichen`
   - `vg :: Kennzeichen -> Kennzeichen`

   die angewendet auf ein gültiges Kennzeichen das nachfolgende (Funktion `nf`) bzw. das vorhergehende Kennzeichen (Funktion `vg`) liefern. Ist das Argument kein gültiges Kennzeichen, so liefern die beiden Funktionen das Argument unverändert als Resultat zurück. Der Typalias `Kennzeichen` ist dabei in folgender Weise festgelegt:

```
type Kennzeichen = ((Char,Char,Char),(Int,Int,Int))
```

Denken Sie bitte daran, dass Sie für die Lösung dieses Aufgabenblatts ein "literate" Haskell-Skript schreiben sollen!

Haskell Live

An einem der kommenden Termine werden wir uns in *Haskell Live* mit den Beispielen der ersten beiden Aufgabenblätter beschäftigen, sowie mit der Aufgabe *City-Maut*.

City-Maut

Viele Städte überlegen die Einführung einer City-Maut, um die Verkehrsströme kontrollieren und besser steuern zu können. Für die Einführung einer City-Maut sind verschiedene Modelle denkbar. In unserem Modell liegt ein besonderer Schwerpunkt auf innerstädtischen Nadelöhren. Unter einem *Nadelöhr* verstehen wir eine Verkehrsstelle, die auf dem Weg von einem Stadtteil A zu einem Stadtteil B in der Stadt passiert werden muss, für den es also keine Umfahrung gibt. Um den Verkehr an diesen Nadelöhren zu beeinflussen, sollen an genau diesen Stellen Mautstationen eingerichtet und Mobilitätsgebühren eingehoben werden.

In einer Stadt mit den Stadtteilen A, B, C, D, E und F und den sieben in beiden Richtungen befahrbaren Routen B–C, A–B, C–A, D–C, D–E, E–F und F–C führt jede Fahrt von Stadtteil A in Stadtteil E durch Stadtteil C. C ist also ein Nadelöhr und muss demnach mit einer Mautstation versehen werden.

Schreiben Sie ein Programm in Haskell oder in einer anderen Programmiersprache Ihrer Wahl, das für eine gegebene Stadt und darin vorgegebene Routen Anzahl und Namen aller Nadelöhre bestimmt.

Der Einfachheit halber gehen wir davon aus, dass anstelle von Stadtteilnamen von 1 beginnende fortlaufende Bezirksnummern verwendet werden. Der Stadt- und Routenplan wird dabei in Form eines Tupels zur Verfügung gestellt, das die Anzahl der Bezirke angibt und die möglichen jeweils in beiden Richtungen befahrbaren direkten Routen von Bezirk zu Bezirk innerhalb der Stadt. In Haskell könnte dies durch einen Wert des Datentyps `CityMap` realisiert werden:

```
type Bezirk     = Integer
type AnzBezirke = Integer
type Route      = (Bezirk,Bezirk)
newtype CityMap = CM (AnzBezirke,[Route])
```

Gültige Stadt- und Routenpläne müssen offenbar bestimmten Wohlgeformtheitsanforderungen genügen. Überlegen Sie sich, welche das sind und wie sie überprüft werden können, so dass Ihr Nadelöhrsuchprogramm nur auf wohlgeformte Stadt- und Routenpläne angewendet wird.