2. Aufgabenblatt zu Funktionale Programmierung vom Mi, 28.10.2015. Fällig: Mi, 04.11.2015 / Mi, 11.11.2015 (jeweils 15:00 Uhr)

Themen: *Funktionen über ganzen Zahlen, Wahrheitswerten, Listen und Tupeln*

Für dieses Aufgabenblatt sollen Sie Haskell-Rechenvorschriften für die Lösung der unten angegebenen Aufgabenstellungen entwickeln und für die Abgabe in einer Datei namens `Aufgabe2.hs` ablegen. Sie sollen für die Lösung dieses Aufgabenblatts also ein "gewöhnliches" Haskell-Skript schreiben. Versehen Sie wieder alle Funktionen, die Sie zur Lösung brauchen, mit ihren Typdeklarationen und kommentieren Sie Ihre Programme aussagekräftig. Benutzen Sie, wo sinnvoll, Hilfsfunktionen und Konstanten.

1. Vor der Einführung des metrischen Systems war die englische Währungseinheit *Pound Sterling* in *Shilling* und *Pence* unterteilt. 20 Pence ergaben einen Shilling, 12 Shilling ein Pound Sterling. In Haskell führen wir dafür folgende Typsynonyme ein:

   ```
   type Pence         = Int
   type Shilling      = Int
   type PoundSterling = Int
   type Amount        = (PoundSterling,Shilling,Pence)
   ```

   Ein Geldbetrag (ps, s, p) ist *gültig*, falls gilt: $0 \leq ps$, $0 \leq s$ und $0 \leq p$. Ein gültiger Geldbetrag (ps, s, p) liegt in *Normalform* vor, falls gilt: $0 \leq s < 12$ und $0 \leq p < 20$.

   - Implementieren Sie Selektorfunktionen `pence :: Amount -> Pence`, `shilling :: Amount -> Shilling` und `poundSterling :: Amount -> PoundSterling`.
   - Implementieren Sie eine Wahrheitswertfunktion `isSound :: Amount -> Bool`, die angewendet auf ein Argument a feststellt, ob a einen gültigen Geldbetrag darstellt.
   - Implementieren Sie eine Normalisierungsfunktion `normalize :: Amount -> Amount`, die angewendet auf einen gültigen Geldbetrag a den wertgleichen Geldbetrag in Normaldarstellung berechnet. Ist a nicht gültig, wird a selbst als Resultat geliefert.
   - Schreiben Sie eine Funktion `add :: Amount -> Amount -> Amount` zur Addition von Geldbeträgen. Angewendet auf zwei gültige Geldbeträge a und b, liefert die Funktion `add` die Summe dieser Geldbeträge in normalisierter Form. Ist eines der Argumente nicht gültig, so bricht die Funktion mit dem Aufruf der Funktion `error "Falsche Eingabe"` ab.
   - Schreiben Sie eine Funktion `interest :: Amount -> InterestRate -> Period -> Amount` zur Zinsrechnung, wobei folgende weitere Typsynonyme verwendet werden:

     ```
     type InterestRate = Float
     type Years        = Int
     type Period       = Years
     ```

     Angewendet auf einen gültigen Geldbetrag a, einen nichtnegativen Zinssatz p, $0 \leq p$, und einen Anlagezeitraum j von Jahren, $1 \leq j$, liefert `zins` in Normalform denjenigen Geldbetrag zurück, der sich aus a bei j-jähriger Veranlagung zum Prozentsatz p einschließlich Zins und Zinseszins ergibt.

     Um Rundungsabweichungen zu verhindern, führen Sie die Zinsberechnung so durch, dass zunächst a wertgleich durch b in *Pence* dargestellt wird und auf b (dargestellt als `Float`) die Zinseszinsrechnung auf dem Datentyp `Float` ohne Rundungen ausgeführt wird. Der so erhaltene Betrag (in `Float`) wird kaufmännisch gerundet und in *Pence* dargestellt. Die Normaldarstellung dieses Betrags ist das gesuchte Ergebnis der Funktion `interest`.

     Verstoßen eines oder mehrere der Argumente gegen die eingangs genannten Bedingungen, so liefert die Funktion `interest` den Wert $(0, 0, 0)$ zurück.

2. Schreiben Sie in Analogie zu den Funktionalen `curry` und `uncurry` aus der Vorlesung zwei entsprechende Funktionale `curry3 :: ((a,b,c) -> d) -> a -> b -> c -> d` und `uncurry3 :: (a -> b -> c -> d) -> (a,b,c) -> d`.

   Überlegen Sie sich geeignete Funktionen f und g, um die korrekte Arbeitsweise der Funktionale `curry3` und `uncurry3` zu überprüfen und testen. Implementieren Sie sie und testen Sie die Implementierungen ihrer Funktionale `curry3` und `uncurry3`.

3. Zwei Raketen werden auf frontalem Kollisionskurs entdeckt, die eine mit der Geschwindigkeit v_1 (gemessen in km pro s), die andere mit der Geschwindigkeit v_2 (gemessen in km pro s). Zum Entdeckungszeitpunkt sind die beiden Raketen noch x km voneinander entfernt. Welchen Abstand haben die beiden Raketen voneinander genau t Sekunden, bevor sie miteinander kollidieren?

   Schreiben Sie eine Haskell-Rechenvorschrift mit der Signatur

   ```
   dist :: DistanceAtDetection -> Speed -> Speed -> TimeBeforeCollision -> Distance
   ```

   zur Beantwortung dieser Frage, wobei die in der Signatur der Rechenvorschrift `dist` verwendeten Typsynonyme in folgender Weise deklariert sind:

   ```
   type DistanceAtDetection = Float       -- Einheit: km
   type Speed               = Float       --          km pro s
   type TimeBeforeCollision = Integer     --          s
   type Distance            = Integer     --          km
   ```

   Angewendet auf den Abstand d der Raketen zum Zeitpunkt ihrer Entdeckung, ihrer Geschwindigkeiten v_1 und v_2 und dem Zeitraum t bis zu ihrer Kollision, berechnet die Funktion `dist` in kaufmännischer Rundung auf volle ganze Zahlen die Entfernung der beiden Raketen voneinander, die sie t Sekunden vor ihrer Kollision haben. Ist eines der Argumente negativ, d.h. echt kleiner als 0, oder ist die ab ihrer Entdeckung verbleibende Flugzeit bis zur Kollision geringer als t, so liefert die Funktion `dist` das Resultat -1. Relativistische Effekte müssen nicht berücksichtigt werden. Sie können davon ausgehen, dass die Funktion `dist` nur mit Geschwindigkeiten in der Größenordnung aufgerufen wird, die heutige Raketen aus eigenem Antrieb heraus erreichen können.

4. Wir betrachten eine Variante der Wochentagsaufgabe von Aufgabenblatt 1. Ist bekannt, auf welchen Wochentag der 1. Jaenner 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 er fällt. Diesen Wochentag wollen wir für jeden Tag eines Jahres berechnen können. Zur Modellierung dieser Aufgabe in Haskell verwenden wir folgende Typsynonyme:

   ```
   type Wochentag     = String
   type Schaltjahr    = Bool
   type ErsterJaenner = Wochentag
   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 `wochentag2 :: ErsterJaenner -> Schaltjahr -> Datum -> Wochentag`, die für jedes Datum des Jahres berechnet, auf welchen Wochentag es fällt: Angewendet auf ej, sj und ein Datum (t, m) liefert `wochentag2` denjenigen Wochentag, auf den das durch (t, m) bestimmte Datum in diesem Jahr fällt. Ist eines der Argumente nicht gültig, so liefert `wochentag2` die Zeichenreihe `"Falsche Argumente"` zurück. Fällt das durch (t, m) bezeichnete Datum auf den Tag vor Sonntag, darf die Rechenvorschrift `wochentag2` wahlweise den Wert `"Samstag"` oder `"Sonnabend"` als Ergebnis liefern.

Wichtig: Wenn Sie einzelne Rechenvorschriften aus früheren Lösungen für dieses oder spätere Aufgabenblätter wieder verwenden möchten, so kopieren Sie diese in die neue Abgabedatei ein. Ein `import` schlägt für die Auswertung durch das Abgabeskript fehl, weil Ihre alte Lösung, aus der importiert wird, nicht mit abgesammelt wird. Deshalb: Kopieren statt importieren zur Wiederwendung!

Haskell Live

Am Freitag, den 30.10.2015, oder an einem der späteren Termine werden wir uns in *Haskell Live* u.a. mit der Aufgabe *"Krypto Kracker!"* beschäftigen.

Krypto Kracker!

Eine ebenso populäre wie einfache und unsichere Methode zur Verschlüsselung von Texten besteht darin, eine Permutation des Alphabets zu verwenden. Bei dieser Methode wird jeder Buchstabe des Alphabets einheitlich durch einen anderen Buchstaben ersetzt, wobei keine zwei Buchstaben durch denselben Buchstaben ersetzt werden. Das stellt sicher, dass verschlüsselte Texte auch wieder eindeutig entschlüsselt werden können.

Eine Standardmethode zur Entschlüsselung nach obiger Methode verschlüsselter Texte ist als "reiner Textangriff" bekannt. Diese Angriffsmethode beruht darauf, dass der Angreifer den Klartext einer Textphrase kennt, von der er weiß, dass sie in verschlüsselter Form im Geheimtext vorkommt. Durch den Vergleich von Klartext- und verschlüsselter Phrase wird auf die Verschlüsselung geschlossen, d.h. auf die verwendete Permutation des Alphabets. In unserem Fall wissen wir, dass der Geheimtext die Verschlüsselung der Klartextphrase

`the quick brown fox jumps over the lazy dog`

enthält.

Ihre Aufgabe ist nun, eine Liste von Geheimtextphrasen, von denen eine die obige Klartextphrase codiert, zu entschlüsseln und die entsprechenden Klartextphrasen auszugeben. Kommt mehr als eine Geheimtextphrase als Verschlüsselung obiger Klartextphrase in Frage, geben Sie alle möglichen Entschlüsselungen der Geheimtextphrasen an. Im Geheimtext kommen dabei neben Leerzeichen ausschließlich Kleinbuchstaben vor, also weder Ziffern noch sonstige Sonderzeichen.

Schreiben Sie ein Programm in Haskell oder in irgendeiner anderen Programmiersprache ihrer Wahl, das diese Entschlüsselung für eine Liste von Geheimtextphrasen vornimmt.

Angewendet auf den aus drei Geheimtextphrasen bestehenden Geheimtext (der in Form einer Haskell-Liste von Zeichenreihen vorliegt)

```
["vtz ud xnm xugm itr pyy jttk gmv xt otgm xt xnm puk ti xnm fprxq",
 "xnm ceuob lrtzv ita hegfd tsmr xnm ypwq ktj",
 "frtjrpgguvj otvxmdxd prm iev prmvx xnmq"]
```

sollte Ihre Entschlüsselungsfunktion folgende Klartextphrasen liefern (ebenfalls wieder in Form einer Haskell-Liste von Zeichenreihen):

```
["now is the time for all good men to come to the aid of the party",
 "the quick brown fox jumps over the lazy dog",
 "programming contests are fun arent they"]
```