2. Aufgabenblatt zu Funktionale Programmierung vom Mi, 24.10.2012. Fällig: Mi, 31.10.2012 / Mi, 07.11.2012 (jeweils 15:00 Uhr)

Themen: *Funktionen über ganzen Zahlen, Zeichenreihen, 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. Schreiben Sie eine Haskell-Rechenvorschrift `kgv :: Integer -> Integer -> Integer`, die angewendet auf zwei von 0 verschiedene Argumente m und n das *kleinste gemeinsame Vielfache* der Beträge von m und n berechnet; ist eines der Argumente 0, soll die Funktion als Resultat 0 liefern.

   Folgende Beispiele illustrieren das Aufrufverhalten:

   ```
   kgv 12 20 ->> 60
   kgv 15 40 ->> 120
   kgv (-15) 40 ->> 120
   kgv 17 0 ->> 0
   ```

2. Schreiben Sie eine Haskell-Rechenvorschrift `agv :: Integer -> Integer -> (Integer,Integer) -> [Integer]`, die angewendet auf zwei von 0 verschiedene Argumente m und n und ein Paar ganzer Zahlen (p, q) alle echt positiven gemeinsamen Vielfachen v der Beträge von m und n mit $p \leq v \leq q$ bestimmt; die gemeinsamen Vielfachen sollen dabei in der Resultatliste aufsteigend geordnet sein. Hat eines der Argumente m und n den Wert 0, soll die Funktion als Resultat die leere Liste liefern.

   Folgende Beispiele illustrieren das Aufrufverhalten:

   ```
   agv 12 20 (50,130) ->> [60,120]
   agv 15 40 ((-50),150) ->> [120]
   agv (-15) 20 (60,180) ->> [60,120,180]
   agv 15 40 (200,10) ->> []
   agv 17 0 ((-1000),1000) ->> []
   ```

Eine einfache Datenbank zur Speicherung von Flugdaten bestehend aus Passagiernamen (nur Vornamen!), Flugnummer, Abflug- und Ankunftsort und Ticketpreis kann in Haskell wie folgt realisiert werden:

```
type PassName         = String
type FlightNumber     = Integer
type PlaceOfDeparture = String
type Destination      = String
type Airfare          = Integer
type Database         = [(PassName,FlightNumber,PlaceOfDeparture,Destination,Airfare)]
```

Schreiben Sie Haskell-Rechenvorschriften für folgende Datenbankaufgaben:

3. Eine Funktion `flights :: Database -> PassName -> [(FlightNumber,Airfare)]`, die für einen Passagier in aufsteigender Folge alle Nummern gebuchter Flüge zusammen mit dem jeweiligen Ticketpreis ausgibt.

4. Eine Funktion `pass2Dest :: Database -> Destination -> [PassName]`, die für einen Ankunftsort lexikalisch aufsteigend sortiert alle Passagiernamen ausgibt, die einen Flug zu diesem Ort gebucht haben.

5. Eine Funktion `mostValuedPass :: Database -> PlaceOfDeparture -> Destination -> ([PassName],Airfare)`, die für eine bestimmte Verbindung lexikalisch absteigend sortiert die Namen derjenigen Passagiere bestimmt, die den höchsten Ticketpreis für diese Verbindung bezahlt haben, sowie diesen Ticketpreis.

Folgende Beispiele illustrieren das Aufrufverhalten:

```
db = [ ("Anton",857,"Vienna","London",237),
       ("Berta",456,"Paris","Berlin",278),
       ("Anton",123,"Rome","London",417),
       ("Anton",109,"London","Berlin",237),
       ("Karla",888,"Vienna","Rome",350),
       ("Karla",832,"Rome","London",417),
       ("Berta",857,"Vienna","London",199),
       ("Karla",753,"Vienna","London",237) ]

flights db "Anton" ->> [(109,237),(123,417),(857,237)]
pass2Dest db "London" ->> ["Anton","Anton","Berta","Karla","Karla"]
mostValuedPass db "Vienna" "London" ->> (["Karla","Anton"],237)
```

Wichtiger Hinweis: Wenn Sie einzelne Rechenvorschriften aus früheren Lösungen 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 09.11.2012, 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"]
```