2. Aufgabenblatt zu Funktionale Programmierung vom Mi, 29.10.2014. Fällig: Mi, 05.11.2014 / Mi, 12.11.2014 (jeweils 15:00 Uhr)

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

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.

Ein Text kann in Haskell als Liste von Zeilen dargestellt durch Zeichenreihen realisiert werden:

```
type Line = String
type Text = [Line]
```

Unter einem *Wort* in einem so dargestellten Text verstehen wir eine maximale aufeinanderfolgende Folge von Zeichen, die das Leerzeichen nicht enthält. Zwei Wörter in einem Text sind also durch ein oder mehrere Leerzeichen getrennt, nicht aber durch einen Zeilenübergang. Endet eine Zeile mit einem vom Leerzeichen verschiedenen Zeichen und beginnt die folgende Zeile mit einem vom Leerzeichen verschiedenen Zeichen, so beginnt mit dieser Zeile kein neues Wort, sondern das Wort aus der Vorzeile wird fortgesetzt. Die Länge eines Wortes ist die Länge seiner Zeichenfolge; ein Wort, das aus n aufeinanderfolgenden Zeichen besteht, hat also Länge n.

In der Folge wollen wir einige Rechenvorschriften schreiben, die Texte untersuchen oder transformieren.

1. Schreiben Sie eine Haskell-Rechenvorschrift `maxWordLength` mit der Signatur

   ```
   maxWordLength :: Text -> Integer
   ```

   Angewendet auf einen Text t liefert `maxWordLength` die Länge des oder der längsten in t vorkommenden Wörter.

2. Schreiben Sie eine Haskell-Rechenvorschrift `numWordOcc` mit der Signatur

   ```
   numWordOcc :: Text -> Integer -> Integer
   ```

   Angewendet auf einen Text t und eine Zahl n, $n \geq 1$, liefert `numWordOcc` die Anzahl der in t vorkommenden paarweise verschiedenen Wörter der Länge n. Für $n < 1$ liefert `numWordOcc` das Resultat -1.

3. Schreiben Sie eine Wahrheitswertfunktion `reverseCheck` mit der Signatur

   ```
   reverseCheck :: Text -> Bool
   ```

   Angewendet auf einen Text t liefert `reverseCheck` den Wahrheitswert `True`, falls es eine oder mehrere Zeilen in t gibt, die von links nach rechts und von rechts nach links gelesen zeichenweise übereinstimmen. Die Wahrheitswertfunktion `reverseCheck` liefert insbesondere den Wahrheitswert `True`, wenn eine Zeile von links nach rechts und von rechts nach links gelesen zeichenweise übereinstimmt. Gibt es keine solche Zeile oder Zeilen, liefert `reverseCheck` den Wahrheitswert `False`.

4. Schreiben Sie eine Haskell-Rechenvorschrift `formatText` mit der Signatur

   ```
   formatText :: Text -> Integer -> Text
   ```

   Angewendet auf einen Text t und eine Zeilenlänge n, $n \geq 1$, liefert `formatText` einen neu formatierten Text t'. In t' haben mit Ausnahme höchstens der letzten Zeile alle Zeilen die Länge n, d.h. enthalten n Zeichen (einschließlich Leerzeichen); die letzte Zeile kann weniger als n Zeichen enthalten. Ist eine Zeile zu kurz, so wird die folgende Zeile an sie angehängt. Ist eine Zeile zu lang, gleich ob von vornherein oder durch Aneinanderhängen zweier Zeilen, so wird der n Zeichen übersteigende Zeilenrest in die folgende Zeile verschoben usw. Ist $n < 1$, so liefert `formatText` den Argumenttext t unverändert zurück.

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 31.10.2014, 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"]
```