5. Aufgabenblatt zu Funktionale Programmierung vom 10.11.2004, fällig: 17.11.2004

Themen: *Funktionen auf ganzen Zahlen, Zeichenreihen und Listen*

Für dieses Aufgabenblatt sollen Sie Haskell-Rechenvorschriften zur Lösung der unten angegebenen Aufgabenstellungen entwickeln und für die Abgabe in einer Datei namens `Aufgabe5.hs` ablegen. Sie sollen also wieder ein "gewöhnliches" Haskell-Script schreiben. Versehen Sie wie auf den bisherigen Aufgabenblättern alle Funktionen, die Sie zur Lösung brauchen, mit ihren Typdeklarationen und kommentieren Sie Ihre Programme aussagekräftig. Benutzen Sie, wo sinnvoll, Hilfsfunktionen und Konstanten. Im einzelnen sollen Sie folgende Problemstellungen bearbeiten:

1. In dieser Aufgabe betrachten wir noch einmal positive ganze Zahlen in römisch(artig)er Darstellung, darunter die folgende Modifikation der auf Aufgabenblatt 4 eingeführten Variante, für die wir als Wohlgeformtheitsregel ergänzen, dass nur das Zahlzeichen `I` subtraktiv verwendet werden darf, höchstens ein Vorkommen dieses Zeichens subtraktiv verwendet werden darf und dieses dann nur als erstes Zeichen. In der Folge werden wir diese Variante als V1 bezeichnen. Daneben betrachten wir die beiden auf der von einer ihrer Kolleginnen in einem news-Gruppenbeitrag genannten Webseite

   `http://www.diaware.de/html/roemzahl.html`

   besprochenen Darstellungsvarianten für römische Zahlen gemäß "allgemeiner Regeln" und gemäß "ergänzender Regeln", die wir in der Folge als Varianten V2 und V3 bezeichnen. Die auf dieser Webseite genannten "zusätzlichen Zeichen (ZZ)" betrachten wir nicht, ebenso nicht die Zusatzregel für Ziffernblätter.

   Schreiben Sie eine Haskell-Rechenvorschrift `convert` mit der Signatur `convert: String -> Int -> Int -> String`, die römische Zahlen von einer Darstellungsvariante in eine andere überführt. Dabei bezeichnet das erste Argument die zu konvertierende Zahl, das zweite Argument die Darstellungsvariante (1, 2 oder 3), in der das erste Argument vorliegt, und das dritte Argument die Darstellungsvariante, in der das Ergebnis vorliegen soll. Das Ein-/Ausgabeverhalten ist dann wie folgt festgelegt: Ist die als Argument übergebene Zeichenreihe keine gültige Darstellung der im zweiten Argument spezifizierten Darstellungsvariante, oder besitzt diese Zahl keine gültige und eindeutige Darstellung in der im dritten Argument festgelegten Darstellungsvariante, oder sind die Werte des zweiten und dritten Arguments von 1, 2 und 3 verschieden, so liefert die Funktion `convert` das Ergebnis `Ungueltige Eingabe`, ansonsten die entsprechend konvertierte Zeichenreihe.

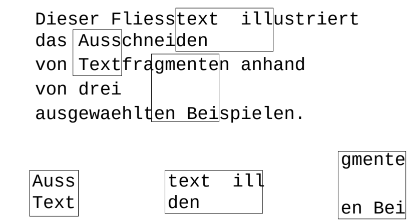
2. Eine weitere einfach zu berechnende (und auch einfach zu brechende!) Methode, die Erhaltung der Vertraulichkeit von Nachrichten zu erhöhen, ist neben der Caesar-Chiffrierung eine als Würfel-Chiffrierung bekannte Verschlüsselungstechnik. Bei der Würfel-Chiffrierung wird eine Zeilenlänge k, $k \geq 1$, als Kennzahl vereinbart. Die Chiffrierung einer Nachricht geschieht dann in folgender Weise. Die zu codierende Nachricht wird zunächst in Zeilen der Länge k aufgeschrieben, wobei die letzte Zeile ggf. unvollständig bleibt. Die codierte Nachricht erhält man dann durch spaltenweises Auslesen der Zeilen. Folgendes Beispiel illustriert das Vorgehen. Die Nachricht `functionalprogrammingisfun` liefert bei Kennzahl 7 über den Zwischenschritt

   ```
   functio
   nalprog
   ramming
   isfun
   ```

   die Codierung `fnriuaasnlmfcpmutrinionogg`.

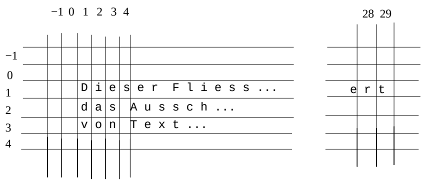
   Schreiben Sie zwei Haskell-Rechenvorschriften `codeCube` und `decodeCube`, beide mit der Funktionalität `String -> Int -> String`, zur Codierung und Decodierung von Zeichenreihen nach der Würfelmethode. Für die Funktion `codeCube` gibt das ganzzahlige Argument die für die Chiffrierung zu verwendende Kennzahl an, für die Funktion `decodeCube` die Kennzahl, die bei der Chiffrierung verwendet worden ist. Das Zeichenreihenargument stellt jeweils die zu codierende bzw. decodierende Nachricht dar.

3. Fließtext besteht aus linksbündigen Zeilen möglicherweise unterschiedlicher Länge. Oft ist man nur an einem bestimmten Ausschnitt aus einem Fließtext interessiert, wie in der folgenden Abbildung illustriert.



Ausschnitt 1 Ausschnitt 2 Ausschnitt 3

In dieser Aufgabe wollen wir das Ausschneiden rechteckig begrenzter Textfragmente realisieren. Fließtext modellieren wir dazu als Liste von Zeichenreichen. Jede Zeichenreihe repräsentiert dabei eine (möglicherweise leere) Fließtextzeile. Wir nehmen an, dass jedes Zeichen die gleiche Breite hat. Der relevante Ausschnitt wird dann durch die Position des ersten Zeichens des Ausschnitts (Zeile m, Spalte n), sowie die Breite (in Anzahl Spalten i) und Höhe (in Anzahl Zeilen j) bestimmt. Daraus ergibt sich für die zu schreibende Haskell-Rechenvorschrift `fragment` die Signatur `fragment :: [String] -> (Int,Int) -> Int -> Int -> [String]`. Das erste Argument bezeichnet den Fließtext, das zweite Argument die Position (m, n) des ersten Zeichens des Ausschnitts, das dritte Argument i die Breite und das vierte Argument j die Höhe des Ausschnitts. Als Ergebnis liefert die Funktion `fragment` den entsprechenden Ausschnitt wie in der obigen Abbildung illustriert. Das gilt auch, wenn der gewählte Ausschnitt teils oder ganz aus dem Fließtext herausfällt. Dabei ist die Höhe des Fließtextes durch die Anzahl von Zeilen (einschließlich aller Leerzeilen) und die Breite durch die Anzahl der Zeichen der längsten Zeile gegeben, wodurch ebenfalls eine rechteckige Begrenzung entsteht. Wie in der folgenden Abbildung illustriert, ist $(1, 1)$ die Position des ersten Zeichens der ersten Zeile eines Fließtexts. Felder außerhalb des durch den Fließtexts begrenzten Rechtecks betrachten wir als “virtuell”: sie werden nur zur Lageberechnung des gewünschten Ausschnitts relativ zum Fließtext herangezogen. Sind die Argumente für Ausschnittsbreite oder -höhe kleiner oder gleich 0, so liefert die Funktion die leere Liste als Ergebnis.



Mit dem Fließtext aus den obigen Abbildungen liefert der Aufruf von `fragment` mit Position $(1, 1)$, Breite 5 und Höhe 2 das Ergebnis `["Diese","das A"]`, der Aufruf mit Position $(0, -1)$, Breite 5 und Höhe 2 das Ergebnis `["Die"]`, der Aufruf mit Position $(1, 28)$, Breite 3 und Höhe 2 das Ergebnis `["rt"]` (da Zeile 2 nicht mit Leerzeichen aufgefüllt ist; sonst `["rt","   "]`, d.h. Leerzeichen ist zu unterscheiden von “nicht beschriftet”).