5. Aufgabenblatt zu Funktionale Programmierung vom 9.11.2011.
Fällig: 23.11.2011 / 30.11.2011 (jeweils 15:00 Uhr)

Themen: *Funktionen über Strings*

Für dieses Aufgabenblatt sollen Sie Haskell-Rechenvorschriften für die Lösung der unten angegebenen Aufgabenstellungen entwickeln und für die Abgabe in einer Datei namens `Aufgabe5.hs` ablegen. Sie sollen dieses Mal also wieder ein "gewöhnliches" Haskell-Skript schreiben. Versehen Sie wieder 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 die im folgenden beschriebenen Problemstellungen bearbeiten.

Um einen Text zeilen- und wortweise zu bearbeiten gibt es im Prelude die Funktionen `lines :: String -> [String]` und `words :: String -> [String]` sowie die dazugehörigen Funktionen in umgekehrter Richtung, `unlines` und `unwords`. Verwenden Sie diese Funktionen in den folgenden Beispielen.

1. Ist die Funktion `unlines` die exakte Umkehrfunktion von `lines`? Dokumentieren Sie drei Gegenbeispiele mit `ctxlines :: [String]`, sodass die folgenden Ausdrücke wahr sind (also `True` sind).

   ```
   all (\x->(unlines.lines)x/=x) ctxlines

   (((<)2).length) ctxlines

   any (((<)2).length.lines) ctxlines
   ```

   Analog für die umgekehrte Richtung: `(lines.unlines)`. Gegenbeispiele in `ctxunlines :: [[String]]`.

   Analog für `ctxwords :: [String]`.

   Zerteilen Sie die Zeichenketten in den folgenden Beispielen **nicht** mit einer eigenen Funktion sondern nur mit `lines` und `words`.

2. Implementieren Sie die folgenden den Unix-Kommandos `tac`, `head`, `tail`, `grep` nachempfundenen Funktionen.

   `unixtac :: String -> String` die Zeilen in umgekehrter Reihenfolge

   `unixhead :: Int -> String -> String` die ersten n Zeilen

   `unixtail :: Int -> String -> String` die letzten n Zeilen

   `unixgrep :: String -> String -> String` alle Zeilen, in denen das 1. Argument als Teilzeichenreihe vorkommt.

3. Heben Sie das gemeinsame der Funktionen heraus, indem Sie eine allgemeinere Funktion `aslines :: ([String]->[String]) -> String -> String` definieren.

   Definieren Sie nun analog `unixtac'`, `unixhead'`, `unixtail'` und `unixgrep'`, die alle `aslines` verwenden.

4. `unixrev :: String -> String` dreht jede Zeile zeichenweise um

   `wordrev :: String -> String` dreht jede Zeile wortweise um

5. Definieren Sie analog zum UNIX-Kommando `wc(1)`:

   `unixwcw :: String -> Int` die Anzahl der Wörter

   `unixwc :: String -> (Int,Int,Int)` Zeilen, Wörter und Zeichen eines Texts, wobei jedes Element des Strings als ein Zeichen zählt.

(Nicht bewertet) Die Funktionen dieser Woche lassen sich direkt als einfache Shell-Kommandos verwenden. Mit `runhugs` bzw. `runghc` können Funktionen der Signatur `String -> String` direkt als Shell-Kommandos verwendet werden. Dazu wird lediglich eine "toplevel" Funktion `main` benötigt. Z.B.:

```
main :: IO ()
main = getContents >>= putStr.show.unixwc
```

Vergleichen Sie die Laufzeiten von Haskell mit jenen der Unix-eigenen Kommandos!

Haskell Live

An einem der kommenden *Haskell Live*-Termine, der nächste ist am Freitag, den 11.11.2011, werden wir uns u.a. mit der Aufgabe *World of Perfect Towers* beschäftigen.

World of Perfect Towers

In diesem Spiel konstruieren wir Welten perfekter Türme. Dazu haben wir n Stäbe, die senkrecht auf einer Bodenplatte befestigt sind und auf die mit einer entsprechenden Bohrung versehene Kugeln gesteckt werden können. Diese Kugeln sind ebenso wie die Stäbe beginnend mit 1 fortlaufend nummeriert.

Die auf einen Stab gesteckten Kugeln bilden einen Turm. Dabei liegt die zuerst aufgesteckte Kugel ganz unten im Turm, die zu zweit aufgesteckte Kugel auf der zuerst aufgesteckten, usw., und die zuletzt aufgesteckte Kugel ganz oben im Turm. Ein solcher Turm heißt *perfekt*, wenn die Summe der Nummern zweier unmittelbar übereinanderliegender Kugeln eine Zweierpotenz ist. Eine Menge von n perfekten Türmen heißt *n-perfekte Welt*.

In diesem Spiel geht es nun darum, n-perfekte Welten mit maximaler Kugelzahl zu konstruieren. Dazu werden die Kugeln in aufsteigender Nummerierung, wobei mit der mit 1 nummerierten Kugel begonnen wird, so auf die n Stäbe gesteckt, dass die Kugeln auf jedem Stab einen perfekten Turm bilden und die Summe der Kugeln aller Türme maximal ist.

Schreiben Sie in Haskell oder einer anderen Programmiersprache ihrer Wahl eine Funktion, die zu einer vorgegebenen Zahl n von Stäben die Maximalzahl von Kugeln einer n-perfekten Welt bestimmt und die Türme dieser n-perfekten Welt in Form einer Liste von Listen ausgibt, wobei jede Liste von links nach rechts die Kugeln des zugehörigen Turms in aufsteigender Reihenfolge angibt.