Prof. Dr. Manfred Schmidt-Schauß
M.Sc. Nils Dallmeyer

Goethe-Universität Frankfurt am Main/ Institut für Informatik
Künstliche Intelligenz/Softwaretechnologie

Grundlagen der Programmierung 2

Sommersemester 2018

Aufgabenblatt Nr. 3

Abgabe: Mittwoch 2. Mai 2018 **vor!** der Vorlesung

Aufgabe 1 (25 Punkte)

Verwenden Sie im Wesentlichen `elem`, `head`, `tail`, `map`, `reverse`, `last`, `filter`, `concat`, `sum` und `(++)`, um die folgenden Funktionen zur Listenverarbeitung in Haskell zu programmieren – die Verwendung von Pattern-Matching ist **verboten**:

a) Eine Funktion, die einen String[1] erhält und zunächst folgende Ersetzungen durchführen soll:

- Entspricht ein Zeichen einer Ziffer, so soll es auf die entsprechende Zahl abgebildet werden.
- Entspricht ein Zeichen dem Buchstaben `a`, so soll dieses Zeichen auf die Zahl `1` abgebildet werden.
- Alle anderen Zeichen sollen auf die Zahl `0` abgebildet werden.

Danach soll die Summe dieser Zahlen berechnet werden. Zum Beispiel soll für `"138aza"` der Wert `14` berechnet werden. (7 Punkte)

b) Einige Leute geben heutzutage einer Aussage nochmal eine besonders freundliche Betonung, indem sie alles groß schreiben. Ebenso tippen einige Personen gefühlt 20.000 Zeichen pro Sekunde mit dem Smartphone, haben aber absolut keine Zeit, die Groß- und Kleinschreibung zu beachten. Geben Sie eine Funktion an, die eine Liste von Strings erhält und alle Kleinbuchstaben eines Strings, der mit einem Ausrufezeichen endet, in Großbuchstaben umwandelt. Falls ein String jedoch nicht mit einem Ausrufezeichen endet, sollen alle Großbuchstaben in Kleinbuchstaben umgewandelt werden. Zum Beispiel soll für

```
["Du bist so intelligent wie eine Bananeschale und halb so gutaussehend!",
"Ich schreibe jedes Wort klein Nebensaetze und Punkte kenne ich nicht"]
```

das folgende Ergebnis berechnet werden:

```
["DU BIST SO INTELLIGENT WIE EINE BANANESCHALE UND HALB SO GUTAUSSEHEND!",
 "ich schreibe jedes wort klein nebensaetze und punkte kenne ich nicht"]
```

(7 Punkte)

c) Eine Funktion, die eine Liste von Listen von Zahlen erhält und zunächst alle innersten Listen entfernt, die weniger als 4 Elemente enthalten. Die übriggebliebenen inneren Listen sollen umgedreht werden und außerdem sollen bei jeder inneren Liste alle Elemente außer den ersten Dreien entfernt werden. Danach sollen alle inneren Listen zu einer großen verschmolzen werden. Zum Beispiel soll für `[[1,2],[1..4],[6..8],[8..12]]` das Ergebnis `[4,3,2,12,11,10]` berechnet werden. (11 Punkte)

[1] Strings sind in Haskell nichts anderes als Listen von Zeichen, d.h. `"Haskell"` ist nur eine andere Darstellung für die Liste `['H','a','s','k','e','l','l']`.

Hinweis: Die Bibliothek `Data.Char` beinhaltet nützliche Funktionen, die Sie in Ihrer Lösung verwenden dürfen. Sie können die Bibliothek durch `import Data.Char` am Anfang des Quelltextes in Ihr Programm einbinden. Die Dokumentation von Bibliotheken finden Sie im Internet unter `http://www.haskell.org/ghc/docs/latest/html/libraries/`.

Aufgabe 2 (30 Punkte)

Implementieren Sie in Haskell die folgenden Funktionen auf Listen im Wesentlichen unter Verwendung von Pattern-Matching und Rekursion – d.h. Sie dürfen Hilfsfunktionen definieren und nutzen, allerdings ist die Verwendung von Listenfunktionen wie z.B. `map`, `concat`, `replicate` usw. **verboten**.

a) Eine Funktion, die eine Liste erwartet und je zwei benachbarte Elemente vertauscht. Enthält die Liste eine ungerade Anzahl an Elementen, so bleibt die Position des letzten Elements unverändert. Zum Beispiel soll für die Liste `[1..9]` das Ergebnis `[2,1,4,3,6,5,8,7,9]` berechnet werden. (7 Punkte)

b) Eine Funktion, die eine Liste von Listen von Listen von Zahlen erwartet und die Summe der jeweils ersten Elemente der innersten Listen berechnet. Zum Beispiel soll `44` für die Liste `[[[1,2],[4,5],[7,8,9]],[[13,15],[19]]]` berechnet werden. (10 Punkte)

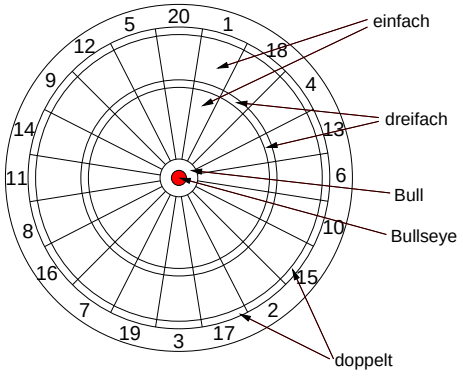
c) Eine Funktion, die eine Liste erwartet und eine Liste zurückgibt, die das k-te Element der Eingabeliste k-mal enthält. Dabei soll die Reihenfolge unverändert bleiben und der Listenindex erstreckt sich von `1` bis zur Länge der Liste. Zum Beispiel soll für die Liste `['a','b','c']` das Ergebnis `"abbccc"` berechnet werden. (13 Punkte)

Aufgabe 3 (45 Punkte)

Diese Aufgabe beschäftigt sich mit Darts.

Das Dart-Board ist in 20 Zahlensegmente unterteilt. Die Felder zählen jeweils soviel, wie außen am entsprechenden Segment notiert ist. Ausnahmen sind die schmalen Kreise. Im äußeren Kreis zählen die Felder das Doppelte (Double-Felder), im mittleren Kreis das Dreifache (Triple-Felder) der entsprechenden Zahl. Der Mittelpunkt des Boards, das Bullseye, zählt 50 Punkte (Double 25), der Ring darum herum (Bull) 25 Punkte. Gespielt wird die Variante 501 mit Double-Out:

Jeder Spieler beginnt bei 501 Punkten. Eine Aufnahme besteht aus (maximal) drei nacheinander geworfenen Darts. Die erzielten Punkte werden jedesmal von der verbleibenden Punktzahl abgezogen, wobei nach jeder Aufnahme der jeweilige Gegenspieler dran ist. Es dürfen nur diejenigen Darts gezählt werden, die im Board steckenbleiben. Gewonnen hat derjenige Spieler, der zuerst genau 0 Punkte erreicht hat, wobei der letzte Dart ein Doppel-Feld oder das Bullseye treffen muss. Erzielt der Spieler zu viele Punkte oder so viele, dass er nicht mehr mit einem Doppel abschließen kann (der Spieler überwirft sich), so werden alle Darts dieser Aufnahme nicht gezählt – der Spieler bleibt auf dem selben Rest wie vor der Aufnahme. In der letzte Aufnahme, in der die Punktzahl genau auf 0 reduziert wird, können auch weniger als 3 Darts geworfen werden. Falls man sich überwirft kann eine Aufnahme ebenso aus weniger als 3 Darts bestehen.



Ein einzelnes Spiel dieser Art nennt man Leg. Wer zuerst eine festgelegte Anzahl an Legs gewonnen hat, gewinnt das gesamte Spiel. Dabei beginnen die Spieler die Legs jeweils abwechselnd.

Ein Leg im obigen Modus sei in Haskell wie folgt dargestellt:

- Ein Liste mit genau zwei Elementen $[m, p]$ stellt die Punktzahl eines einzelnen geworfenen Darts dar, wobei für m eine 1, 2 oder 3 entsprechend Single, Double oder Triple darstellt und p die Punktzahl des Zahlensegments ist. Das heißt die Triple 20 entspricht der Darstellung $[3, 20]$, Double 20 $[2, 20]$ und Single 20 $[1, 20]$. Da das Bullseye als Doppelfeld gezählt wird, wird es als $[2, 25]$ dargestellt. Außerdem entspricht $[0, 0]$ einem Fehlwurf.
- Eine Aufnahme ist eine Liste solcher zweielementigen Listen, z.B. entspricht $[[3, 20], [3, 19], [2, 25]]$ einem Treffer in die Triple 20, gefolgt von einem Treffer in die Triple 19, gefolgt von einem Treffer in das Bullseye. Falls ein Spieler seine Aufnahme schon mit weniger als 3 Darts beendet hat, sich also überworfen oder das Leg gewonnen hat, so enthält die Liste entsprechend weniger Elemente.
- Ein Leg wird durch eine Liste von Aufnahmen (also einer Liste von Listen mit zweielementigen Listen) dargestellt, in der jede Aufnahme mit ungerader Listenposition dem Spieler zugeordnet ist, der das Leg begonnen hat und alle geraden dem Gegenspieler.

Hier einige Beispiel-Legs:

```
leg1 = [[[3,20],[3,20],[3,20]], [[3,20],[3,19],[2,25]], [[3,19],[3,19],[3,19]],
        [[3,20],[3,19],[2,25]], [[2,25],[2,25],[2,25]]]
leg2 = [[[3,20],[3,20],[3,20]], [[3,20],[3,19],[2,25]], [[3,19],[3,19],[3,19]],
        [[3,20],[3,19],[2,25]], [[2,25],[2,25],[1,25]], [[3,20],[3,19],[2,25]]]
leg3 = [[[3,20],[1,20],[1,20]], [[3,20],[1,20],[1,20]], [[3,20],[3,20],[3,20]],
        [[3,20],[3,20],[3,20]], [[3,20],[1,1] ,[2,20]], [[3,20],[1,1] ,[2,20]],
        [[3,20],[1,20],[2,20]]]
leg4 = [[[3,20],[1,20],[1,20]], [[3,20],[1,20],[1,20]], [[3,20],[3,20],[3,20]],
        [[3,20],[3,20],[3,20]], [[3,20],[1,1] ,[2,20]], [[3,20],[1,1] ,[2,20]],
        [[3,20],[1,20],[1,20]], [[3,20],[1,20],[2,20]]]
```

a) Implementieren Sie in Haskell eine Funktion `legSieger`, die ein Leg in der obigen Darstellung als Eingabe erwartet und `1` zurückliefert, falls der Spieler gewinnt, der das Leg begonnen hat, falls der andere Spieler gewinnt soll `2` zurückgegeben werden. Berücksichtigen Sie dazu die oben dargestellten Regeln.

Die Funktion soll für fehlerhafte Legs `0` zurückgeben. Zum Beispiel falls kein Spieler genau 0 Punkte erreicht hat, die angegebenen Felder gar nicht existieren (zum Beispiel Triple-Bull oder Single-21) oder falls ein Spieler bereits gewonnen hat und der Gegner danach trotzdem noch wirft.

Für die Beispiel-Legs soll also `legSieger leg1` als Ergebnis `1` liefern und `legSieger leg2` soll `2` berechnen.

Tipp: Verwenden Sie Hilfsfunktionen um die Regeln systematisch abzudecken. Wie kann das Abwechseln der Spieler einfach implementiert werden? (35 Punkte)

b) Die letzten maximal 3 Würfe eines Legs, die der Gewinner des Legs zum gewinnen benötigt, nennt man Finish. Implementieren Sie in Haskell eine Funktion `highOut`, welche eine Liste von Legs in der obigen Darstellung entgegennimmt und für den Spieler, welcher das erste Leg beginnt, über alle Legs sein höchstes Finish ausrechnet. Beachten Sie dass diese Information während des Spiels berechnet werden können soll, gegebenenfalls ist also das letzte Leg in der Eingabeliste noch nicht zu Ende gespielt. Falls der Spieler (noch) kein Leg gewonnen hat, soll `0` zurückgegeben werden.

Zum Beispiel soll `highOut [leg3,leg2,leg3,leg4,leg3]` das Ergebnis `167` zurückgeben, während `0` für `highOut [leg4,leg3]` zurückgegeben werden soll. (10 Punkte)