Praktische Informatik 3 WS 04/05

Christoph Lüth <cxl>
Shi Hui <shi>
Klaus Lüttich <luettich>
Wolfgang Machert <wmachert>
Christian Maeder <maeder>
Hennes Märtins <maertins>
Kai-Florian Richter <richter>
Dennis Walter <dw>
Diedrich Wolter <dwolter>

2. Übungsblatt

Ausgabe: 15.11.04
Bearbeitungszeit: Zwei Wochen

3 *Lebenslang Grün-Weiß* *10 Punkte*

> *"Some people believe football is a matter of life and death.*
> *I'm very disappointed with that attitude.*
> *I can assure you it is much, much more important than that."*
>
> — Bill Shankly, 1913–1981.

Wir lernen ja eigentlich nicht für die Universität, sondern für's Leben. Und wie schon Bill Shankly wusste, ist Fußball das eigentliche Leben.

Deshalb werden wir in dieser Aufgabe eine Funktion zur Verwaltung von Tabellen erstellen. (Dieses Programm eignet sich natürlich im Prinzip für alle ähnlichen Wettkampsportarten, von Eishockey bis zu Tiddlywinks. Wer keinen Fußball mag, setze seine Lieblingssportart ein.)

Dabei werden wir eine Spieledatenbank verwalten, in die neue Spiele eingetragen werden können. Aus dieser Spieledatenbank können dann der momentane Tabellenstand und viele andere nützliche Informationen generiert werden.

Zuerst sollten Sie sich ein geeignete Repräsentation `DBase` für die Spieledatenbank überlegen. Damit wird dann eine Funktion implementiert, die ein Spiel zu dieser Datenbank hinzufügt: Ein Spiel besteht hierbei aus einer Nummer (nämlich dem aktuellen Spieltag), sowie zwei Mannschaften und ihrer jeweiligen Anzahl geschossener Tore. Die erste Mannschaft ist die Heimmannschaft, die zweite Mannschaft die Auswärtsmannschaft.

```
type Game  = (Int, String, Int, String, Int)
addGame :: DBase -> Game-> DBase
```

Hierbei ist sicherzustellen, dass jede Mannschaft pro Spieltag höchstens einmal spielt.

Nachdem solcherart Spiele in die Spieledatenbank eingetragen wurden, sollten folgende Abfragefunktionen implementiert werden, welche den Tabellenstand ausgeben, jeweils für alle Spiele oder nur für Heim- oder Auswärtsspiele, bzw. den bisherigen Saisonverlauf für eine gegebene Mannschaft.

```
table     :: DBase -> String
awayTable :: DBase -> String
homeTable :: DBase -> String
results   :: DBase -> String-> String
```

Zur Berechnung des Tabellenstandes gelten folgende Regeln:

- Für ein gewonnenes Spiel gibt es drei Punkte, für ein Unentschieden einen Punkt.
- Die Mannschaften werden der Anzahl der Punkte nach geordnet.
- Bei gleicher Anzahl Punkte wird die Tordifferenz (Anzahl erzielter minus Anzahl zugelassener Tore) herangezogen.
- Bei gleicher Tordifferenz gilt die Anzahl erzielter Tore.
- Ansonsten sind die Mannschaften gleichplaziert.

Hier ist eine Beispielausgabe für eine Tabelle, die wir immer wieder gerne anschauen. Die Spalten zeigen die Anzahl gespielter Spiele, gewonnener, unentschiedener und verlorener Spiele, erzielte und zugelassene Tore, Tordifferenz und zuletzt Punktezahl.

```
 1 Werder Bremen        32      22       8       2      76:29   +47     74
 2 Bayern München       32      19       8       5      67:36   +31     65
 3 VfB Stuttgart        32      17      10       5      49:21   +28     61
 4 Bayer Leverkusen     32      17       8       7      65:37   +28     59
 5 VfL Bochum           32      14      11       7      52:35   +17     53
 6 Borussia Dortmund    32      15       6      11      55:46    +9     51
 7 FC Schalke 04        32      12      10      10      44:40    +4     46
 8 Hamburger SV         32      13       6      13      45:59   -14     45
 9 Hansa Rostock        32      11       8      13      52:49    +3     41
10 VfL Wolfsburg        32      13       1      18      55:60    -5     40
11 SC Freiburg          32      10       7      15      42:65   -23     37
12 Bor. M'gladbach      32       9       9      14      36:45    -9     36
13 Hannover 96          32       9       9      14      48:60   -12     36
14 Hertha BSC           32       8      11      13      38:57   -19     35
15 1. FC Kaiserslautern 32      11       5      16      37:57   -20     35
16 1860 München         32       8       7      17      30:51   -21     31
17 Eintracht Frankfurt  32       8       5      19      32:49   -17     29
18 1. FC Köln           32       5       5      22      27:54   -27     20
```

(*8 Punkte*)

Wie schon der große Informatiker U. N. Bekannt wusste, sind Voraussagen schwierig, besonders wenn sie die Zukunft betreffen. Deshalb ist dieser Teil der Aufgabe hochspekulativ: wir wollen eine Funktion

```
predict :: DBase -> String-> String-> (Int, Int)
```

implementieren, die das Ergebnis der Begegnung zwischen den beiden angebenen Mannschaften auf der Basis der bereits stattgefunden voraussagt. Dabei können Sie folgende Annahmen zu Grunde legen:

- Der Durchschnitt der bisher zu Hause oder auswärts erzielten oder zugelassenen Tore ist eine verlässliche Annahme über die in diesem Spiel erzielten und zugelassenen Tore;

- Ebenso ist der Durchschnitt der bisher zu Hause oder auswärts erzielten Punkte eine verlässliche Annahme über die in diesm Spiel erzielten Punkte.

Das Problem hierbei ist, diese beiden Annahmen zu einer konsistenten Antwort zu kombinieren—wenn eine Mannschaft n Tore zulässt, aber die andere nur m Tore schießt, wieviel Tore fallen dann? (*2 Punkte*)

Tipp: Wenn Funktionen Zeichenketten zurückgeben, werden diese auf der Kommandozeile formatiert angezeigt, und nicht ausgegeben:

```
Prelude> "\n" ++ "Hallo" ++ "\n"
"\nHallo\n"
```

Mit Hilfe der Funktion `putStrLn` können (nur auf der Kommandozeile!) Zeichenketten ausgegeben und nicht nur angezeigt werden:

```
Prelude> putStrLn ("\n" ++ "Hallo" ++ "\n")

Hallo
```

4 *Haskell in Space!* *10 Punkte*

Wir schreiben das Jahr 2078, und Sie sind mit 95 nach einem langen und erfolgreichen Berufsleben kurz vor dem Eintritt in das inzwischen leicht erhöhte Rentenalter. Eine letzte Aufgabe wartet allerdings noch auf Sie.

Die Menschheit hat inzwischen den größeren Teil des Sonnensystem besiedelt, und auf den größeren Planetoiden des Asteroidengürtels Bergwerke für seltene Metalle (wie Molybdän und Selenium) errichtet. Von diesen Bergwerken fliegen regelmäßig Raketen zurück zur Erde.

Der Abbau des Erzes und die Verladung erfolgt vollautomatisch — die Erzklumpen werden einfach der Reihe nach in die bereitstehende Rakete verladen, bis sie voll ist. Das ist allerdings suboptimal — machmal beinhalten schwere Klumpen recht wenig Erz, und manchmal werden so die Raketen nur halbvoll.

Sie sollen deshalb ein Steuerungsprogramm schreiben, welches die Verladung besser steuert. Zum Glück hat sich Haskell inzwischen flächendeckend durchgesetzt. (Java ist seit einem Vorfall im Jahr 2038 verboten, als ein in Java geschriebenes Programm zu Steuerung der Erdklimakontrollanlage aufgrund eines falschen type casts beinahe die zweite Sintflut auslöste.) Das Problem stellt sich wie folgt dar:

Die Anzahl der zu verladenden Erzklumpen ist n; jeder Klumpen hat ein Gewicht w_i Tonnen und ein Erzgehalt von v_i Kilo (mit $1 \leq i \leq n$). Das Raumschiff hat eine Kapazität von W Tonnen. Zu finden ist eine Menge $\{i_1, \ldots, i_k\}$ von Erzklumpen (mit $1 \leq i_j \leq n$), so dass zum einen die Ladung *zulässig* ist, d.h. das Gesamtgewicht ist kleiner als die Kapazität der Rakete

$$\sum_{j=1}^{k} w_{i_j} \leq w$$

und zum anderen der Gesamterzgehalt der Ladung optimal ist, d.h. alle zulässigen Ladungen haben gleich viel oder weniger Erz.

In Haskell ausgedrückt, implementieren Sie eine Funktion

```
load :: [(Double, Double)]-> Double-> [Int]
```

wobei das erste Argument die Sequenz $[(w_1, v_1), \ldots, (w_n, v_n)]$ aus Paaren von Gewicht und Erzgehalt ist, das letzte Argument W (die Kapazität der Rakete), und das Ergebnis die Sequenz $[i_1, \ldots, i_k]$ der eingeladenen Erzblöcke. (*8 Punkte*)

Wie Sie wissen, verlangen die Technischen Sichervorschriften in der Raumfahrt (TeSiRaum) in der Fassung vom 21.07.2073, dass sicherheitsrelevante Steuerungsprogramme formal korrekt bewiesen sein müssen. Sicherheitsrelevant ist hier, dass die Kapazität der Rakete nicht überschritten wird. Geben Sie eine formale Spezifikation dieser Sicherheitseigenschaft an, und eine Beweisskizze, dass Ihre Implementation diese Sicherheitseigenschaft erfüllt.

Tipp: Wenn Sie Eigenschaften vordefinierter Funktionen (wie `filter`, `map` etc) zeigen wollen, benutzen Sie die Definitionen dieser Funktionen aus dem Kapitel I.8 der Sprachdefinition von Haskell98. (*2 Punkte*)

Dies ist Version 1.1 vom 2004/11/19 09:16:06.