Funktionale Programmierung

`http://proglang.informatik.uni-freiburg.de/teaching/functional-programming/2013/`

Übungsblatt 3 (Datentypen, Listen, Typklassen)

Di, 2013-11-05

Hinweise

- Lösungen sollen als Haskell Quellcode in das persönliche Subversion (svn) Repository hochgeladen werden. Die Adresse des Repositories wird per Email mitgeteilt.
- **Alle** Aufgaben müssen bearbeitet und pünktlich abgegeben werden. Falls das sinnvolle Bearbeiten einer Aufgaben nicht möglich ist, kann eine stattdessen eine Begründung abgegeben werden.
- Wenn die Abgabe korrigiert ist, wird das Feedback in das Repository hochgeladen. Die Feedback-Dateinamen haben die Form `Feedback-<user>-ex<XX>.txt`.
- Allgemeinen Fragen zum Übungsblatt können im Forum (`http://proglang.informatik.uni-freiburg.de/forum/viewforum.php?f=38`) geklärt werden.

Abgabe: Di, 2013-11-12

Aufgabe 1 (Schere, Stein, Papier)

Dem Provider für ein Schere-Stein-Papier [1] Online-Spiel ist bei einem Brand im Datencenter die Highscore-Tabelle abhanden gekommen. Glücklicherweise konnten die Logs der Spieler-Aktivitäten gerettet werden. Sie wurden nun angeheuert die Highscore-Tabelle anhand der Logs zu rekonstruieren.

Die Datentypen der Logs und der Highscore-Tabelle sind im Modul `LogTypes` definiert, dass Sie auf der Vorlesungs-Homepage finden. Geloggt wurden sog. „Game-Events", die angeben wann Spiele zwischen zwei Teilnehmern gestartet oder beendet wurden. Weiterhin wurde aufgezeichnet, wann ein Spieler seine Wahl (Schere, Stein, Papier) änderte bzw. gesetzt hat. Wer ein Spiel gewonnen hat entscheidet sich durch die aktuelle Wahl beider Spieler zum Zeitpunkt der „Stop" Game-Events.

Die Highscore-Tabelle besteht aus einer Liste von Spielern und der Anzahl ihrer Siege (natürlich absteigend geordnet nach der Sieges-Anzahl)

Implementieren Sie eine Funktion reconstructHighScore :: Log −> HighScore, die die Highscore-Tabelle anhand der Logs rekonstruiert.

Aufgabe 2 (Vectoren)

Definieren Sie einen Datentyp für 2D Vektoren mit **Double** Komponenten. Geben Sie für den Datentyp eine möglichst sinnvolle Instanz der **Num**-Typklasse an.

Verallgemeinern Sie den Datentyp und die Operationen, so dass auch andere Komponententypen außer **Double** verwendet werden können. Geben Sie die Typsignaturen für die Operationen an.

Hinweis: Die notwendigen Methoden für **Num** und andere Typklassen können Sie leicht mit Hoogle (`http://www.haskell.org/hoogle/`) herausfinden.

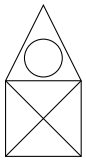
Aufgabe 3 (Zeichenprogramm)

In einem Vektor-Zeichenprogramm werden Bilder nicht durch den Farbwert von Pixeln beschrieben, sondern durch die Anordnung und Skalierung verschiedener geometrischer Elemente (wie Kreise, Rechtecke, Linien, Bezier-Kurven, ...). Beispielsweise ist das folgende Bild

[1] `http://de.wikipedia.org/wiki/Schere,_Stein,_Papier`

ein „Quadrat mit Kantenlänge 1 über einem gleichschenkligem Dreieck mit Höhe 1 und Basislänge 1". (In Programmen muss diese Beschreibung natürlich präzisiert werden)

1. Definieren Sie einen Datentyp Picture mit dem man Linien, Rechtecke, Kreisen und Kombinationen daraus beschreiben kann. Definieren Sie Operationen zum Verschieben und Skalieren von Bildern. Definieren Sie mindestens die oben gezeigte Figur und das „Haus vom Nikolaus mit Dachfenster". (Die Verhältnisse müssen nicht genau gleich sein, nur die ungefähre Figur soll reproduziert werden).



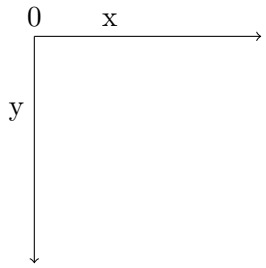
2. Mit dem Datentyp Picture können wir Bilder als Werte repräsentieren. In anderen Systemen, wie „HTML-5 Canvas", werden Grafiken hingegen imperativ beschrieben. D.h. man ruft eine Reihe von Zeichen-Kommandos auf, die das Bild produzieren (`http://www.w3schools.com/html/tryit.asp?filename=tryhtml5_canvas_tut_path2`).

   Im Modul `Canvas`, dass Sie auf der Vorlesungs-Homepage finden, sind Funktionen definiert, die die Javascript Befehle ausgeben, welche zum Zeichnen unserer Picture Werte notwendig sind. (Den Code in `Canvas.hs` hätten Sie auch selbst schreiben können. Er ist nur sehr langweilig und wird deshalb zum Blatt „mitgeliefert"). Implementieren Sie eine Funktion draw, die, mithilfe der Funktionen aus `Canvas`, ein gegebenes Bild in ein Javascript-Programm übersetzt, das dieses Bild zeichnet. Sie können in ghci mit dem Befehl:

   ```
   writeFile "Dateiname" "String"
   ```

   einen String in eine Datei schreiben, und sich so das Ergebnis in einem Browser ansehen.

   Hinweis: Das Canvas-Koordinatensystem zeigt mit der y-Achse „nach unten":



Aufgabe 4 (Monoide)

Eine weitere Typklasse, die in Haskell Programmen öfter Verwendung findet, ist Monoid. Sie ist der gleichnamigen Algebraischen Struktur *Monoid*[2] nachempfunden: eine Menge mit einer assoziativen Operation (mappend) und einem neutralem Element (**mzero**). Die Typklasse Monoid ist in `Data.Monoid` definiert; hooglen Sie diese.

Die prominenteste Instanz für Monoid sind Listen. Dort ist mappend = (++) und **mzero** = [].

Versuchen Sie für die Datentypen aus den vorherigen Aufgaben Monoid-Instanzen zu definieren, falls möglich.

[2] `http://de.wikipedia.org/wiki/Monoid`