Funktionale Programmierung

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

Übungsblatt 6 (GUI, Auswertung, Parsen)

Mi, 2013-11-27

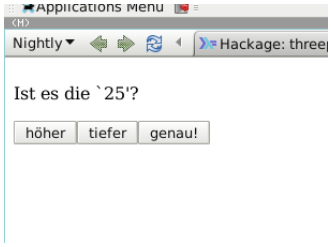
Hinweise

- 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: Mi, 2013-12-04

Aufgabe 1 (Zahlenraten mit GUI)

Implementieren Sie „Zahlenraten" (siehe Ex05) mit einer grafischen Benutzeroberfläche. Dazu sollen Sie die Bibliothek `threepenny-gui` benutzen. Die Benutzeroberfläche kann in etwa so aussehen:



(Sie können natürlich auch eine aufwändigere GUI erstellen)

Installieren Sie zuerst das Paket `threepenny-gui` (falls noch nicht geschehen). Führen Sie dazu folgendes Kommando in der System-Shell aus:

```
cabal install threepenny-gui-0.3.0.1
```

Achtung: nicht die neueste Version nehmen (0.4), sondern (0.3.0.1)

Laden Sie sich die Datei `ThreepennyExample.hs` von der Vorlesungsseite herunter. Darin ist die Verwendung der für diese Aufgabe wichtigen Elemente von `threepenny-gui` beschrieben und demonstriert. Die API-Dokumentation von `threepenny-gui` [1] enthält weitere Informationen; `ThreepennyExample.hs` sollte aber zum Bearbeiten der Aufgabe ausreichen.

Ein threepenny-gui Programm wird gestartet, in dem man die Hauptaktion (meist `main :: IO ()`) ausführt, und dann die URL `http://localhost:10000/` in einem Browser öffnet.

```
Main> main
Listening on http://0.0.0.0:10000/
```

In `ghci` kann man das Programm durch Drücken der Tastenkombination `Strg-C` beenden.

Bitte wenden Sie sich an das Forum, wenn etwas unverständlich sein sollte.

Aufgabe 2 (Striktheit)

Geben Sie für die folgenden Funktionen an, in welchen Parametern sie strikt bzw nicht-strikt sind. Geben Sie für die nicht-strikten Fälle eine Gegenbeispiel an.

[1] `http://hackage.haskell.org/package/threepenny-gui-0.3.0.1`

1. **foldr**
2. **foldl**
3. **take**
4. **zipWith**
5. **unfoldr**

Aufgabe 3 (Newton-Verfahren zur Wurzelberechnung)
In dieser Aufgabe implementieren wir den Newtonschen Algorithmus zur näherungsweisen Berechnung der Quadratwurzelfunktion. Wir versuchen dabei, die Implementierung durch Ausnutzung Haskells nicht-strikter Konstruktoren modular zu halten.

Algorithmus Die Eingabe sind der *Radikant* $r \in \mathbb{R}$ mit $r \geq 0$ und die erste Approximation $a_0 \in \mathbb{R}$ mit $a_0 \geq 0$. Die Ausgabe ist eine Näherung von $\sqrt{r}$.

Das Verfahren berechnet sukzessive die Glieder der Folge

$$a_{i+1} = \frac{(a_i + \frac{r}{a_i})}{2}$$

Das Resultat ist dann a_{n+1} für das kleinste $n \geq 0$ für das gilt $|a_n - a_{n+1}| < \varepsilon$ ($\varepsilon \in \mathbb{R}^+$ ist dabei eine frei-wählbare, kleine Konstante).

1. Implementieren Sie den Algorithmus als Haskell-Funktion **sqrt**, indem Sie die Folge a_n als unendliche Liste definieren und dann die gewünschte Näherung für ein gegebenes ε aus dieser Liste bestimmen.
2. Implementieren Sie auch eine Funktion rel_sqrt, die die Nährung für den *relativen Abstand* $|(1 - a_n/a_{n+1})| < \varepsilon$ bestimmt. Versuchen Sie für rel_sqrt möglichst viele Komponenten aus **sqrt** wiederzuverwenden.

Aufgabe 4 (Parsen)
Laden Sie die Datei `Parser.lhs` von der Vorlesungsseite herunter. Sie enthält das Parser Modul, das in der Vorlesung entwickelt wurde.

Definieren Sie die im Folgenden beschriebenen Parser:

- pmany :: Parser t r −> Parser t [r]
  pmany p akzeptiert p keine oder mehrere Male und fasst die Ergebnisse in einer Liste zusammen.
- pmany1 :: Parser t r −> Parser t [r]
  pmany1 p akzeptiert p ein oder mehrere Male und fasst die Ergebnisse in einer Liste zusammen.
- pIntList :: Parser **Char** [**Integer**]
  pIntList akzeptiert Listen im Haskell Syntax, die Integer-Literale enthalten. Z.B. pIntList "[1, 22,33\n, 44] "== ([1, 22, 33, 44], "")
- pPaliAB :: Parser **Char String**
  pPaliAB akzeptiert Palindrome aus den Zeichen 'a' und 'b'
- pPali :: (**Eq** r) =>Parser t r −> Parser t [r]
  pPali p akzeptiert die Palindrome, die aus Elementen bestehen, die p akzeptiert (z.B.: pPaliAB = pPali (lit 'a' `palt` lit 'b').
- pTwice :: (**Eq** t) =>Parser t [t] −> Parser t [t]
  Für alle ts die p akzeptiert, wird ts ++ ts von pTwice p akzeptiert.