Funktionale Programmierung

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

Übungsblatt 1 (Erste Schritte)

Do, 2013-10-24

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, 2013-10-29

Aufgabe 1 (Fingerübungen)

1. Definieren Sie eine Funktion maxi und eine Funktion mini, die jeweils das Maximum/Minimum von zwei **Int**s berechnen. Geben Sie auch eine entsprechende Typsignatur an. (Das Verwenden der vordefinierten Funktionen **min** und **max** ist übrigens geschummelt!)

2. Definieren Sie weiterhin eine Funktion max3, die das Maximum von drei **Int**s berechnet.

3. Definieren Sie eine Funktion med, die den Median von drei **Int**'s berechnet.

4. Testen Sie ihre Definitionen mit QuickCheck Properties. Versuchen Sie nur die Funktion Data.**List**.**sort** als Referenz zu benutzen. (Falls es nicht klappen sollte, dann verwenden Sie **max** und **min**).

Aufgabe 2 (Stapelrechner)

Wir implementieren die Logik eines Stack-basierten Rechners. Er rechnet auf **Int**s und soll die Operationen push n, pop, dup, add, substract, multiply und neg unterstützen.

Der Stack wird als Liste von **Int**s modelliert. Der initiale Stack ist „unendlich tief" und mit Nullen gefüllt. Das heißt, das Programm

```
pop
push 8
add
```

auf dem initialen Stack ergibt das Ergebnis `8` (anstatt eines Fehlers, dass nicht genug Elemente vorhanden sind)

1. Implementieren Sie die Stack Operationen als Funktionen auf einem Stack (vom Typ [**Int**]); sie erhalten den aktuellen Stack als Argument und liefern den veränderten Stack zurück.

2. Testen Sie ihre Definitionen mit QuickCheck. Achtung! QuickCheck generiert mitunter sehr lange Listen. Falls das beim Testen zu Problemen führt, fangen Sie diese Fälle in den Properties ab (siehe Vorlesung)

3. Stellen Sie sich vor, Ihr Stack-Rechner soll standardmäßig bei iOS8 mitgeliefert werden. Apple hat es allerdings verschlafen auch eine Version von GHC zu integrieren. Die Benutzer müssen also die Operationen als Strings an den Rechner schicken. Schreiben Sie eine Funktion readCommand :: **String** −> [**Int**] −> [**Int**], die anhand des übergebenen Strings die richtige Operation ausführt. Wird der String nicht erkannt, soll gar nichts passieren (noop). Wie Sie die Strings auf Operationen abbilden, bleibt Ihnen überlassen.

   Tipp: Strings sind Listen von **Char**s. Schauen Sie sich also einmal die Funktionen in Data.**Char** an.

Aufgabe 3 (Installation von Zusatzpaketen)
Im Laufe der Vorlesung wollen wir auch einige Zusatzpakete verwenden, die noch nicht in der Haskell-Platform enthalten sind. Damit wir Probleme möglichst früh aus der Welt schaffen, wäre es gut, Sie würden Installation und Lauffähigkeit der Pakete schon jetzt testen. Bitte berichten Sie von allen Problemen, die Sie beim Testen hatten, z.B. in einer Text-Datei in Ihrem Repository.

1. Installieren Sie die folgenden Pakete

   ```
   # ,,update'' muss nur einmal zur Initialisierung ausgeführt werden
   cabal update
   cabal install enumerator gloss lens netwire
   ```

2. Laden Sie sich die Datei `GlossClock.hs` von der Vorlesungsseite herunter. Compilieren Sie diese mit

   ```
   ghc --make GlossTest.hs
   ```

   Es sollte ein ausführbares Programm `GlossTest` erstellt worden sine. Versuchen Sie es zu starten.

   ```
   ./GlossTest
   ```

3. Laden Sie sich die Datei `CalcMain.hs` von der Vorlesungsseite herunter. Laden Sie diese in `ghci`.

   ```
   Prelude> :l CalcMain.hs
   *CalcMain>
   ```

   Laden Sie nun die Definitionen des Stack Rechners aus Aufgabe 2.

   ```
   *CalcMain> :l Ex01.hs
   *Ex01>
   ```

   Wenn Sie Aufgabe 2 nicht gelöst haben, definieren Sie stattdessen folgende Funktion in einer Datei und laden diese.

   ```
   1 readCommand str st = length str : st
   ```

   Importieren Sie nun `CalcMain` nochmals und versuchen Sie, dass Programm `calcMain` wie folgt auszuführen:

   ```
   *Ex01> import CalcMain
   *Ex01 CalcMain> calcMain readCommand "42" []
   [... es folgt eine Menge Output ...]
   Listening on http://0.0.0.0:10000/
   [24/Oct/2013:13:43:18 +0200] Server.httpServe: START, binding to [http://0.0.0.0:100
   ```

   Laden Sie URL `http://localhost:10000` in einem Webbrowser. Sie sollten nun eine magere GUI für den Stack-Taschenrechner sehen und benutzen können.