Übung 4: Funktionen

Information: Whitespace und Indentation

Leerzeichen in Haskell Source Code haben eine Bedeutung! Das kann zu verwirrenden Compiler Meldungen führen. Als goldene Regel der Einrückung sollten Sie sich folgendes merken:

1. **Code which is part of some expression should be indented further in than the beginning of that expression.**
2. **All grouped expressions must be exactly aligned!**

Folgende Definition kompiliert:

```
a = let
     b = 12
     c = 13
    in
     b + c
```

Ein zusätzliches Leerzeichen führt aber zu einer Fehlermeldung beim Kompilieren:

```
a = let
     b = 12
      c = 13
    in
     b + c

.../AS04Offside.hs:3:9:
    parse error on input ‘=’
Failed, modules loaded: none.
```

Das Problem ist die Verletzung der Regel 2. Die vollständigen, formalen Regeln sind etwas umfangreich. Wichtig ist, dass Sie solche Fehlermeldungen als Einrückungsprobleme erkennen und diese beseitigen können. Weitere Informationen finden Sie im Haskell Wiki http://en.wikibooks.org/wiki/Haskell/Indentation .

Aufgabe 1: Conditional Syntax

Sie haben in einem Arbeitsblatt gesehen, dass bedingte Ausführungen ganz unterschiedlich implementiert werden können. Als weiteres Beispiel diene hier die `compare`-Funktion. Sie ist eine Funktion der Klasse `Ord` und ermittelt in welchem Verhältnis zwei Werte stehen. Das Ergebnis ist vom Typ `Ordering`. Der Typ `Ordering` ist wie folgt definiert:

```
data Ordering = LT | EQ | GT
```

Es gilt folgendes: wenn x und y jeweils vergleichbar sind, dann ist das Ergebnis von `x `compare` y`

- `LT` falls `x < y`
- `EQ` falls `x = y`
- `GT` falls `x > y`

Der Typ von `compare` ist:
```
compare :: (Ord a) => a -> a -> Ordering
```

Aufgabe:
Implementieren Sie die Funktion `compare` mit

a) if-then-else
b) guards
c) case expression
d) Begründen Sie, warum es nicht möglich ist, die Fallunterscheidung mittels Pattern Matching in der Funktionsdefinition zu implementieren.

Hinweis:
Sie können die Konstrukte auch verschachtelt anwenden. Also z.B. eine if-Expression in einer if-expression. Es ist auch möglich die Konstrukte beliebig zu mischen, also eine if-Expression in einer case-Expression, die wiederum mit guards...

Aufgabe 2: Pattern Matching

Der Compiler kann nicht automatisch erkennen, ob die Liste der Muster abschliessend ist, oder ob es auch Fälle gibt, die nicht erkannt werden.

a) Überprüfen Sie was passiert, wenn kein Muster passt.
b) Was passiert, wenn die Muster sich überlappen, wenn also ein Muster schon im anderen (teilweise) enthalten ist? Stellen Sie entsprechende Definitionen auf und überprüfen Sie diese mit GHCi.

Aufgabe 3: Matrix Library

Definieren Sie eine Typ Synonym um 2x2 Matrizen mit Int Komponenten zu definieren.

type M22 = ...

Definieren Sie dann basierend darauf folgende Funktionen:

add :: M22 -> M22 -> M22 -- Addiert zwei Matrizen
sub :: M22 -> M22 -> M22 -- Subtrahiert die zweite Matrix von der ersten
mulS :: M22 -> Int -> M22 -- Multipliziert eine Matrix mit einem Skalar
mul :: M22 -> M22 -> M22 -- Multipliziert zwei Matrizen