Berthold Hoffmann <hof>
Christian Maeder <maeder>
Klaus Lüttich <luettich>
Pascal Schmidt <pharao90>
Felix Beckwermert <foetus>
Dennis Walter <dw>
Franck Ngueuleu <nngf>

2. Übungsblatt

Ausgabe: 03.11.03

Bearbeitungszeit: Zwei Wochen

3 *Geradengleichungen* — *4 Punkte*

Eine Gerade in der Ebene kann auf unterschiedliche Weise repräsentiert werden:

- mit Hilfe von zwei verschiedenen Punkten
- durch eine Gleichung der Form $ax + by + c = 0$ (mit $a \neq 0$ oder $b \neq 0$).

1. Ausgehend von einem Typ `Point`:

   ```
   type Point = (Double, Double)
   ```

   geben Sie zwei Typsynonyme `LineByPoints` und `LineByEq` für die obigen zwei Geradenrepräsentationen an.

2. Implementieren Sie eine Konvertierungsfunktion

   ```
   toLineByEq :: LineByPoints -> LineByEq
   ```

   mit Hilfe der Zweipunktegleichung: $(y - y_1)(x_2 - x_1) = (x - x_1)(y_2 - y_1)$ für zwei Punkte (x_1, y_1) und (x_2, y_2)

3. Implementieren Sie eine Funktion, die testet, ob ein Punkt auf einer Geraden vom Typ `LineByPoints` liegt. Dazu definieren Sie eine Vergleichsfunktion über `Double`, die nur eine gewisse Genauigkeit verlangt, um Rundungsfehler auszugleichen.

4 *Polygonzüge* — *6 Punkte*

Ein Polygonzug ist durch eine (nicht leere) Liste von Punkten gegeben:

```
type Polygon = [Point]
```

1. Implementieren Sie eine Funktion, die die minimale und maximale x- und y-Koordinaten berechnet, also das umfassende Rechteck als ein Punktepaar liefert.

2. Formen Sie ein Polygon äquivalent so um, dass zwei benachbarte Punkte immer verschieden sind (bzgl. obiger Genauigkeit).

3. Normieren Sie ein Polygon weiter so, dass drei benachbarte Punkte nicht auf einer Geraden liegen. Ggf. wird der mittlere Punkt weggelassen. Geben Sie ein Beispiel dafür an, dass sich durch diese Normierung das umfassende Rechteck ändern kann.

$\boxed{5}$ *Warenlager* *10 Punkte*

Wir stellen uns ein Warenlager wie folgt vor. Jedes Gut (`Item`) hat eine eindeutige Artikelbezeichnung. Das gesamte Lager (`store`) wird durch zwei Listen repräsentiert: die erste Liste ordnet jedem Artikel einen Preis (`Double`) zu und die zweite jedem Artikel seine Häufigkeit (`Int`) im Lager.

```
type Item    = String
type Price   = Double
type Prices  = [(Item, Price)]
type Items   = [(Item, Int)]
type Store   = (Prices, Items)
```

1. Implementieren Sie

   ```
   lookupItem   :: Store -> Item -> Bool
   lookupPrice  :: Store -> Item -> Price
   lookupNumber :: Store -> Item -> Int
   ```

   Die erste Funktion soll testen, ob der Preis eines Artikels bekannt ist, die zweite liefert den Preis, und die dritte die Häufigkeit (im Zweifel 0) des Artikels im Lager.

2. Implementieren Sie

   ```
   remove :: Store -> (Item, Int) -> Store
   ```

   die einen Artikel n-mal aus dem Lager entfernt

3. Implementieren Sie

   ```
   total :: Prices -> Items -> Price
   ```

   die den Gesamtpreis der Artikel (multipliziert mit ihren Häufigkeiten) ermittelt. Als vordefinierte Konvertierungsfunktion (von `Int` nach `Double`) kann `fromIntegral` verwendet werden.

4. Erzeugen Sie aus der Preisliste eines Lagers eine List von Textzeilen.

   ```
   prnt :: Store -> [String]
   ```

   Jede Zeile (vom Typ `String`) soll geeignet formatiert die Artikelbezeichnung, den Einzelpreis und die Häufikgeit des Artikels im Lager enthalten.

Dieses ist Version 2.0 von 5. November 2003.