Praktische Informatik 3 WS 20/21

Christoph Lüth
Thomas Barkowsky
Tobias Brandt
Alexander Krug
Tarek Soliman
Robert Sachtleben

5. Übungsblatt

Ausgabe: 07.12.2020
Abgabe: 14.12.2020 12:00

Dank unseres Kassensystems (und den Fichtenspänen) florieren die Geschäfte im *Happy Coffecup*[1], und zwar dergestalt, dass die deutschlandweite Expansion als Franchise-System ansteht. Der Inhaber (der alte Geizhals) fragte bei einem renommierten deutschen Softwarehaus mit drei Buchstaben nach einem „state-of-the-art ERP-System“, aber der aufgerufene Preis (vierstellig — für den Kostenvoranschlag) schreckte ihn ab. Unser Preis hingegen (drei Muffins nach Wahl und Kaffee satt) schien akzeptabel. Auf ans Werk!

5.1 *Kalter Kaffee 2.0* — *4 Punkte*

Zuerst wollen wir nur einzelne Filialen und Bezirke verwalten. Dabei sind die Bezirke hierarchisch aufgebaut (ähnlich einem Dateisystem): ein Bezirk enthält Filialen oder andere (Unter-)Bezirke.

Wir nutzen dafür folgende Datentypen:

```
data Filiale = Filiale { filialName :: String
                       , mitarbeiter :: Int
                       , umsatz :: Int
                       } deriving (Show, Eq)

data Bezirk = Bezirk { bezirksName :: String
                     , unterBezirke :: [Bezirkseinheit]
                     } deriving (Show, Eq)

data Bezirkseinheit = Bezirkseinheit Bezirk
                    | Filialeinheit Filiale
                    deriving (Show, Eq)
```

Eine Filiale hat einen Namen, eine Anzahl Mitarbeiter und Umsatz. Bezirke haben auch einen Namen, sowie Unterbezirke oder Filialen (Bezirkseinheiten). Die Namen der Filialen und der Bezirke sollen jeweils eindeutig sein. (Damit werden auch zyklische Bezirkstrukturen ausgeschlossen!)

Um die Bezirkstruktur aufzubauen, implementieren wir folgende wir drei Funktionen:

```
neuerBezirk :: String → Bezirk
neuerUnterbezirk :: Bezirk → Bezirk → Bezirk
neueFiliale :: Bezirk → String→ Int→ Int→ Bezirk
```

Die erste Funktion erzeugt einen neuen Bezirk ohne Bezirkseinheiten, die anderen beiden fügen einen neuen Unterbezirk bzw. eine neue Filiale zu dem Bezirk im ersten Argument hinzu.

Wenn bei `neueFiliale` eine Filiale mit einem Namen hinzugefügt wird, der schon irgendwo im ersten Argument enthalten ist[2], dann soll der erste Bezirk unverändert zurückgegeben werden. Analog soll, wenn bei `neuerUnterbezirk` ein Bezirk hinzugefügt wird, in dem ein Name (jeweils einer Filiale oder eines Bezirks) enthalten ist, der auch im ersten Argument enthalten ist, der erste Bezirk unverändert zurückgegeben werden.

Außerdem sollen Bezirke und Filialen entfernt werden können (wenn sie die Franchise-Gebühr nicht zahlen). Dazu implementieren wir die zwei Funktionen:

```
entferneFiliale :: String → Bezirk→ Bezirk
entferneBezirk :: String → Bezirk → Maybe Bezirk
```

Wenn der genannte Bezirk bzw. die genannte Filiale nicht gefunden wird, soll der Bezirk unverändert zurückgegeben werden. `entferneBezirk nm b` soll `Nothing` zurückgeben, wenn `nm` der Name des Bezirks `b` ist (d.h. der gesamte Bezirk wird gelöscht), ansonsten wird der genannte Bezirk rekursiv gesucht (und entfernt):

[1] Vgl. 2. Übungsblatt.
[2] Dabei die hierarchische Struktur beachten — der Name kann auch in einer der Unterbezirke enthalten sein.

```
entferneBezirk "Foo" (Bezirk "Foo" []) ⇝ Nothing
```

Hinweise:

1. Es ist nützlich, folgende Hilfsfunktionen zu implementieren:

   ```
   alleBezirke :: Bezirk→ [String]
   alleFilialen :: Bezirk→ [String]
   ```

   die jeweils alle in dem Bezirk enthaltenen Namen von (Unter-)Bezirken und Filialen zurückgeben.

2. Folgende vordefinierte Funktionen können nützlich sein:

   ```
   mapMaybe  :: (α → Maybe β) → [α] → [β]   -- aus Data.Maybe
   intersect :: Eq α⇒ [α]→ [α]→ [α]         -- aus Data.List
   ```

5.2 *Kaffee statista* *6 Punkte*

Außerdem möchte die Geschäftsleitung Statistiken über die KPIs (*Kaffee Performance Indicators*) auswerten können. Implementiert dazu folgende Funktionen:

```
maxUmsatz ::  Bezirk → (String, Int)
minUmsatz ::  Bezirk → (String, Int)

maxMitarbeiter ::  Bezirk → (String, Int)
minMitarbeiter ::  Bezirk → (String, Int)
```

Diese sollen den Namen der Filiale mit dem größten bzw. kleinsten Umsatz bzw. Mitarbeitern in dem Bezirk, sowie jeweils Umsatz bzw. Mitarbeiter.

Die folgenden KPIs zeigen die Profitabilität des Unternehmens und gehen direkt in den Performance-Bonus des CEO mit ein:

```
sumUmsatz ::  Bezirk → Int
maxProfit ::  Bezirk → (String, Double)
```

Dabei ist `sumUmsatz` der Gesamtumsatz in dem Bezirk. Die Funktion `maxProfit` soll die Filiale mit dem höchsten Profit, hier definiert als Umsatz pro Mitarbeiter (d.h. der Quotient Umsatz und Mitarbeiter), zurückgeben.

3 Punkte

Diese statistischen Angaben möchte die Geschäftsleitung übersichtlich dargestellt haben. ("Diese ganzen Zahlen kann doch kein Mensch lesen, ich bin eher der visuelle Typ."). Leider kann man sich nicht einigen, was wichtiger ist, Umsatz oder Mitarbeiter? Deshalb implementieren wir flexibel.

Implementieren Sie dafür die Funktion:

```
tabelle :: (Filiale→ Int)→ String→ Int→ Bezirk→ String
tabelle kpi_select kpi_name width b =
```

Hier sind die ersten beiden Parameter eine Funktion `kpi_select`, welche aus einer Filiale den KPI selektiert (Umsatz oder Gewinn, oder eine Kombination daraus), und der zweite die Bezeichnung `kpi_name` des KPI.

Die Tabelle enthält eine Zeile für jede Bezirkseinheit in dem fraglichen Bezirk. In jeder Zeile einen Bezirk oder eine Filiale und den KPI. Die Namen sollen dabei eingerückt werden, um die hierarchische Struktur darzustellen. Danach soll ein einfaches Balkendiagram implementiert werden, indem proportional zum KPI so viele `'#'` dargestellt werden, dass es maximal `width` (der dritte Parameter) viele sind. Wir müssen also den KPI mit dem Quotienten aus `width` und dem Maximum aller KPIs skalieren.

Die KPIs der Filialen ergeben sich direkt durch die Funktion `kpi_select`, während die KPIs der Bezirke durch das Aufsummieren der enthaltenen Unterbezirke und Filialen berechnet wird.

Die Tabelle enthält ferner Kopf- und Fußzeile.

Es folgen zwei einfache Beispiele (die Daten für die Beispiele sind in der Test-Datei in der Vorlage zu finden):

```
putStr (tabelle umsatz "Umsatz" 40 d) ⇝
Umsatz für den Bezirk Deutschland

Bezirk/Filiale             | Umsatz
Bez. Bremen                | ######################### 7656
  Fil. Neustadt            | ######## 2343
  Fil. Horn                | ################# 5234
  Fil. Viertel             |  79
Fil. Berlin-Hbf            | ############ 3742
Bez. Niedersachsen         | ######################################## 12236
  Fil. Aurich              |  50
  Bez. Oldenburg           | ################# 5321
    Fil. Oldenburg-Zentrum | ################# 5321
  Fil. Jever               | ######## 2302
  Bez. Hannover            | ############### 4540
    Fil. Hannover-Hbf      | ############## 4322
    Fil. Messe             | # 218
  Fil. Westochtersum       |  23

Umsatz gesamt: 23634

putStr (tabelle mitarbeiter "Mitarbeiter" 10 ns) ⇝
Mitarbeiter für den Bezirk Niedersachsen

Bezirk/Filiale           | Mitarbeiter
Fil. Aurich              | #### 3
Bez. Oldenburg           | ######### 7
  Fil. Oldenburg-Zentrum | ######### 7
Fil. Jever               | ######### 7
Bez. Hannover            | ########## 8
  Fil. Hannover-Hbf      | ###### 5
  Fil. Messe             | #### 3
Fil. Westochtersum       | ######## 6

Mitarbeiter gesamt: 31
```

Im nächsten Übungsblatt implementieren wir dann den Export nach Excel.

3 Punkte

Hinweise:

1. Für die ersten Funktionen ist es nützliche, eine Hilfsfunktion zu implementieren, welche die Filialen in einem Bezirk zurück liefert:

   ```
   filialen :: Bezirk→ [Filiale]
   ```

2. Folgende vordefinierte Funktion (aus `Data.List` zu importieren) sind eventuell hilfreich:

   ```
   maximumBy :: (α→ α→ Ordering) → [α] → α
   minimumBy :: (α→ α→ Ordering) → [α] → α
   ```

3. Für die Tabellen ist es nützlich, die Berechnung der Daten von der Darstellung zu trennen, konkret indem eine Hilfsfunktion implementiert wird

   ```
   statData :: (Filiale→ Int)→ Int→ Bezirkseinheit→ [(String, Int)]
   ```

   welche die Daten für die Zeilen in der Tabelle erzeugt (jedes Element der Liste ist eine Zeile, die Zeichenkette ist der Name, schon korrekt eingerückt).

4. Für die links/rechtsbündige Formatierungen von Zeichenketten enthält beispiels das Fallbeispiel zu Onkel Robert's Delikatessenparadies aus der Vorlesung geeignete Hilfsfunktionen (die schon in der Vorlage enthalten sind):

   ```
   formatL :: String→ Int→ String
   formatR :: String→ Int→ String
   ```