Universität Bremen

Praktische Informatik 3 WS 2023/24

Thomas Barkowsky
Tede von Knorre
Yaroslav Purgin
Tim Sperling

5. Übungsblatt

Ausgabe: 20.11.2023
Abgabe: 27.11.2023

Achtung: Die Aufgaben dieses Übungsblatts sollen mit Funktionen höherer Ordnung (`map`, `filter`, `foldr`, ...) gelöst werden. Lösungsvorschläge, die auf die Verwendung von Funktionen höherer Ordnung verzichten führen zu **Punktabzügen**!

Das Szenario: Ein kleiner Supermarkt führt seine Warenliste mit Stift und Papier. Zurzeit läuft das Geschäft leider schlecht, da mittlerweile das ganze Warenlager bis unter die Decke randvoll mit Papier gefüllt ist. Der Lagerplatz wird aber dringend für die Verkaufsware benötigt. Letztendlich werden wir beauftragt, einige Funktionen zu programmieren, damit zukünftig weniger Papierstapel im Supermarkt anfallen.

5.1 *Warehouse* *5 Punkte*

Ein Artikel im Warenlager wird durch einen eindeutigen Namen repräsentiert; zwei Artikel haben also niemals denselben Namen. Außerdem hat jeder Artikel eine Quantität.

```
type Item = (String, Int)
```

Das Warenlager des Supermarkts wird durch eine Artikelliste `itemList` repräsentiert, in der die Verkaufsware mit der aktuell vorrätigen Menge gespeichert ist. Wir haben zwei Beispielwarenlager, `il1` und `il2`.

```
type ItemList = [Item]
il1 = [("Honey",5), ("Milk",14), ("Vinegar",14), ("Egg", 37)]
```

Implementieren Sie eine Funktion `isItem`, welche prüft, ob ein Artikel im aktuellen Sortiment enthalten ist. Dafür ist die Funktion `any` hilfreich.

```
isItem "Milk" il1 ⇝ True
isItem "Windshield_Wiper" il1 ⇝ False
```

Implementieren Sie eine Funktion `sortIL`, die eine Artikelliste nach dem Namen aufsteigend sortiert.

```
sortIL il1 ⇝ [("Egg",37),("Honey",5),("Milk",14),("Vinegar",14)]
```

Die Funkion `sumIL` summiert die Mengen einer Artikelliste. So können wir zum Beispiel die Artikelanzahl im Warenlager bestimmen. Die Funktion `sum` könnte hier hilfreich sein.

```
sumIL il1 ⇝ 70
```

Nun wollen wir eine Funktion `showItem` implementieren, die einen Artikel hübsch in einen String mit fester Länge formatiert. An den Rändern und in der Mitte werden Pipe-Symbole und je nach Breite Leerzeichen eingefügt.
Falls die Breite kleiner als die Länge eines Eintrags ist, werden nur die ersten Zeichen dargestellt, und wenn der Name die leere Liste ist, oder die Breite 0 oder kleiner ist, wird die leere Liste zurückgegeben.
In der ersten Spalte stehen die Leerzeichen hinter dem Namen, in der zweiten Spalte stehen sie vor dem Wert. Hinweis: Die Funktion `show` ist hilfreich.

```
putStr (showItem 6 ("Sugar",4)) ⇝ |Sugar |      4|
putStr (showItem 4 ("Mayonnaise", 400)) ⇝ |Mayo| 400|
```

Jetzt können wir die Funktion `showIL` implementieren, welche eine Artikelliste in einen String verwandelt. Diese Tabelle soll die Breite des längsten Artikelnamens haben. Für die leere Liste gibt die Funktion eine leere Liste zurück.

```
putStr $ showIL il1 ⇝
#################
|Honey  |      5|
|Milk   |     14|
|Vinegar|     14|
|Egg    |     37|
#################
```

5.2 *Shopkeeping* *5 Punkte*

Der Supermarkt wird regelmäßig mit neuer Ware beliefert, damit das Warenlager aufgefüllt wird. Der Lieferant tätigt leider keine Warenrücknahme, das heißt, jeder bestellte Artikel wird im Supermarkt verkauft. Unsere Aufgabe ist es, die richtige Bestellmenge zu berechnen, damit die Warenbestellung in Zukunft automatisiert getätigt werden kann. In der Liste `target1` ist der gewünschte Lagerbestand verzeichnet, der erreicht werden soll.
Wir können uns darauf verlassen, dass immer alle Artikelnamen unseres Sortiments enthalten sind, und dass beide Listen immer die gleiche Länge haben.

```
target1 = [("Milk", 30), ("Egg", 100), ("Vinegar", 12), ("Honey", 12)] :: ItemList
```

Die Funktion `refill` erhält als ersten Parameter den Soll-Wert (`target1`), und als zweiten Parameter das aktuelle Warenlager (`il1`). Die Differenz zwischen diesem Soll-Wert und dem Ist-Wert, sind die Artikel, die nachbestellt werden müssen. Wir sind nur an Artikeln interessiert, die tatsächlich nachgeliefert werden müssen. Die zurückgegebene Liste ist sortiert. Zwei Listen können mit `zipWith` verknüpft werden.

```
refill target1 il1 ⇝ [("Egg",63),("Honey",7),("Milk",16)]
```

Wir benötigen eine Funktion `sellItem`, welche einen zu verkaufenden Artikel aus dem Warenlager entnimmt, also einfach die jeweilige Artikelanzahl reduziert, und die resultierende, sortierte Artikelliste zurückgibt. Der Lagerbestand kann nicht negativ werden, da maximal die im Lager vorrätige Menge verkauft werden kann (der Artikel wäre dann 0-mal in der Artikelliste). Falls der Artikel nicht in der Artikelliste enthalten ist, wird sie unverändert zurückgegeben. Ist der Artikelname leer, geben wir die leere Liste zurück.

```
sellItem ("Milk", 2) il1 ⇝ [("Egg",37),("Honey",5),("Milk",12),("Vinegar",14)]
sellItem ("Honey", 7) il1 ⇝ [("Egg",37),("Honey",0),("Milk",14),("Vinegar",14)]
```

Nun wollen wir eine entsprechende Funktion für eine Artikelliste implementieren. `sellIL` hat zwei Paramter, an erster Stelle die Einkaufsliste (Artikel die aus dem Lager entnommen werden), und an zweiter Stelle das aktuelle Warenlager. Die zurückgegebene Liste ist sortiert. Hier könnte Listenfaltung hilfreich sein.

```
sellIL [("Milk",1),("Egg",2),("Honey",1)] il1 ⇝ [("Egg",35),("Honey",4),("Milk",13),("Vinegar",14)]
```

Wir benötigen außerdem eine Funktion, die Rechnungen erstellt. In einer Preisliste `priceList1` ist der zugehörige Preis (in Eurocent) von jedem Artikel im Supermarkt gespeichert. Diese Obermenge enthält auch Artikel, die aktuell nicht im Sortiment geführt werden.

```
priceList1 = [("Milk", 99), ("Vinegar", 129), ("Egg", 28), ("Rice", 149), ("Honey", 299), ("Juice", 199)]
```

Die Funktion `invoiceIL` hat zwei Parameter in folgender Reihenfolge, die Preisliste `priceList`, und eine Artikelliste. Die Funktion berechnet für jedes Element in der Einkaufsliste den zu zahlenden Betrag anhand der Menge. Die zurückgegebene Liste ist sortiert. An dieser Stelle machen negative Zahlen keinen Sinn, daher müssen wir diesen Fall nicht weiter beachten.

```
invoiceIL priceList1 il1 ⇝ [("Egg",1036),("Honey",1495),("Milk",1386),("Vinegar",1806)]
```

Jetzt können wir den Warenwert der nächsten Bestellung berechnen.

```
sumIL $ invoiceIL priceList2 $ refill target2 il2 ⇝ 69371
```

Falls Ihr hier angekommen seid noch was zum ausprobieren:

```
putStr . showIL . invoiceIL priceList2 $ refill target2 il2
```