Praktische Informatik 3 WS 10/11

Christoph Lüth
Bernd Gersdorf
Christian Maeder
René Wagner
Dennis Walter
Diedrich Wolter
Simon Ulbricht

2. Übungsblatt

Ausgabe: 11.11.10
Abgabe: 22.11.10

3 *Uncle Bob's Aulde Grocery Shoppe* *13 Punkte*

Onkel Robert hat einen klassischen Lebensmittelladen, Roberts Schlemmerparadies, wo die Milch noch offen verkauft wird und man die Eier einzeln kaufen kann. Aber auch in dieser Einkaufsidylle hält die EDV Einzug: Robert braucht ein POS-System, welches die *sales force* (Robert und seine Nichte Selma, 15, in den Schulferien) effizient beim *accounting* unterstützt. Ein Vertreter einer großen deutschen Softwarefirma mit drei Buchstaben hat ihm ein entsprechendes System ("*state of the art*, auch bei Karstadt im Einsatz") für einen günstigen Betrag im niedrigen sechstelligen Bereich angeboten. Als Robert bei seiner örtlichen Sparkasse nach einem Bankdarlehen über diesem Betrag anfragte, bekam der Berater einen Lachanfall, von dem er sich bis heute nicht wieder erholt hat. Deshalb machen wir uns daran, ein solches System zu implementieren; unser Lohn ist eine Tüte Äpfel (und 13 Punkte).

Äpfel	*ct.* 35/Stk
Birnen	*ct.* 30/Stk
Bananen	*ct.* 40/Stk
Zucker	€ 1,00/kg
Mehl	€ 1,35/kg
Salz	€ 1,30/kg
Eier	*ct.* 25/Stk
Milch	€ 1,10/l
Bier	€ 1,00/0.5l (Flasche)
Butter	€ 5,40/kg
Gouda	€ 12,00/kg
Emmentaler	€ 8,00/kg
Salami (ungar.)	€ 17,00/kg
Katenschinken	€ 19,00/kg

Tabelle 1: Preisliste in *Robert's Schlemmerparadies*

1. Robert's Laden bietet verschiedene Delikatessen, die entweder im Stück oder per Mengeneinheit verkauft (siehe Tabelle 1). Modellieren Sie den Datentyp Food für die angebotenen Waren, und den Datentyp Item für einen Einzelposten (z.B. 3 Äpfel, ein halber Liter Milch).

2. Definieren Sie eine Funktion

   ```
   price :: Food→ Integer
   ```

   die den Preis der Ware per Grundeinheit in Eurocent definiert.

3. Definieren Sie einen rekursiven Datentyp Purchase, der einen Einkauf modelliert. Definieren Sie darauf Funktionen

   ```
   empty  :: Purchase
   add    :: Item→ Purchase→ Purchase
   remove :: Item→ Purchase→ Purchase
   ```

   welche einen leeren Einkauf zurückgeben, und einen Einzelposten hinzufügen oder wieder entfernen. Beachten Sie, dass mehrere Einzelposten zusammengefaßt werden können, beispielsweise wenn erst 2 und später 3 Äpfel hinzugefügt werden, sind das insgesamt 5 Äpfel.

4. Zum Schluss definieren wir zwei Funktionen

   ```
   bill  :: Purchase→ String
   total :: Purchase→ Integer
   ```

```
Uncle Bob's Aulde Grocery Shoppe
        ... where salmonella outbreaks are thankfully rare

************************************************
Milk                          1 @ 1,10 EU/l : 1.10
Apple                         5 @ 35 Ct/Stk : 1.75
Eggses                        6 @ 25 Ct/Stk : 1.50
Ham                    0.23 @ 19,00 EU/kg : 4.37
************************************************
                                      Total : 8.72

Thank you for shopping at Bob's.
```

Abbildung 1: Rechnung von Robert's Schlemmerparadies (Muster)

welche zum eine Rechnung (als String) erzeugen, und zum anderen den zu zahlenden Gesamtbetrag berechnen (hierzu Abb. 1).

Der Einkauf wird dann durch eine Folge von add und remove Operationen modelliert, und mit bill oder total abgeschlossen.

5. Zur Korrektheit definieren Sie wie immer Testfunktionen, welche ausgewählte Eigenschaften der oben angeführten Funktionen testen. bill braucht nicht getestet zu werden.

Hinweis: Mit putStrLn können Sie eine Zeichenkette (Datentyp String) auf der Kommandozeile ausgeben. Für den Datentyp String können ferner folgende Funktionen hilfreich sein:

```
(++)      :: String→ String→ String
replicate :: Int→ Char→ String
take      :: Int→ String→ String
drop      :: Int→ String→ String
```

4 *Noch mehr Zeichenketten selbstgemacht* — *7 Punkte*

Die Funktionalität der in der Vorlesung vorgestellten Eigenimplentation von Zeichenketten, **data** Mystring, ist dringend ausbaufähig. Deshalb definieren wir folgende Funktion

```
toU :: MyString→ MyString
toU Empty       = Empty
toU (Cons c s) = Cons (toUpper c) (toU s)
```

welche die Zeichen in der Zeichenkette in Großbuchstaben wandelt. Der Name ist natürlich eine Abkürzung; der volle Name, toUppercase, wird zu schreibaufwändig. (Die Funktion toUpper wird aus dem Modul Char importiert.)

Beweisen Sie folgende Eigenschaften dieser Funktion formal, wie in der Vorlesung gezeigt:

$$\mathsf{len}\ (\mathsf{toU}\ s) = \mathsf{len}\ s \tag{1}$$

$$\mathsf{cat}\ (\mathsf{toU}\ s)\ (\mathsf{toU}\ t) = \mathsf{toU}\ (\mathsf{cat}\ s\ t) \tag{2}$$

$$\mathsf{rev}\ (\mathsf{toU}\ s) = \mathsf{toU}\ (\mathsf{rev}\ s) \tag{3}$$