Universität Bremen

Praktische Informatik 3 WS 20/21

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

2. Übungsblatt

Ausgabe: 16.11.20
Abgabe: 23.11.20 12:00

2.1 *Kalter Kaffee.* *5 Punkte*

Die Geschäfte des kleinen Cafés *Happy Coffecup* florieren, denn der örtliche Barrista ist im Umgang mit Kaffeeschaum und Espresso so talentiert, dass er moderne Kunstwerke in die Tasse zaubert, die man kaum trinken mag. Da fällt es dann auch nicht weiter auf, dass der Besitzer den guten Arabico-Kaffee mit Fichtensägespänen streckt, um der Gewinnspanne etwas auf die Sprünge zu helfen — tatsächlich loben Kritiker den "intensiven Vanille-Geschmack" des Kaffees.

Wie dem auch sei, der Andrang ist groß. Um ihn zu bewältigen, soll ein System zur Verwaltung der Bestellungen implementiert werden. Gelockt vom Versprechen auf Gratiskaffee eilen wir gerne zur Hilfe.

1. Modellieren Sie die Getränke und das Gebäck von der Karte in Abbildung 1 mit entsprechenden Datentypen `Getraenke` und `Gebaeck`, und die Größe der Getränke als `Groesse`.

   Implementieren Sie zwei Funktionen

   ```
   getraenk_preis :: Getraenk→ Groesse→ Int
   gebaeck_preis :: Gebaeck→ Int
   ```

   welche den Preis eines Getränkes in einer gegebenen Größe bzw. eines Gebäcks berechnen. (Nicht jedes Getränk ist in jeder Größe verfügbar — wie modellieren Sie diesen Fehler?)

   Modellieren Sie jetzt den Bestellvorgang. Dazu implementieren Sie einen algebraischen Datentyp `Bestellung` (vergleiche den `Einkaufskorb` aus der Vorlesung), und drei Funktionen zum Start der Bestellung, und um ein Getränk (von einer bestimmten Größe) bzw. ein Gebäck einer Bestellung hinzuzufügen:

   ```
   start :: Bestellung
   order_getraenk :: Bestellung→ Getraenk→ Groesse→ Bestellung
   order_gebaeck :: Bestellung→ Gebaeck→ Bestellung
   ```

   Die Funktion `zwischensumme` berechnet die Summe der Preise der Bestellungen ohne Berücksichtigung die Sonderangebote (*Specials*).

   ```
   zwischensumme :: Bestellung→ Int
   ```

   Nicht jedes Getränk kann in jeder Größe bestellt werden (siehe Preisliste); wenn das Getränk nicht in der bestellten Größe verfügbar ist, verändert sich die Bestellung nicht.

   3 Punkte

2. Jetzt können Sie die Behandlung der Sonderangebote implementieren. Hierzu implementieren Sie eine Funktion

   ```
   rabatt :: Bestellung→ Int
   ```

   welche berechnet, wie viel bei einer Bestellung durch die Sonderangebote eingespart wird (s. die Rabatt-Tabelle in Abbildung 2, sie ergibt sich aus der Karte in Abbildung 1). Die Funktion soll die für den Kunden günstigsten Sonderangebote ausrechnen — das Café rühmt sich seiner guten Beziehung zu seinen Kunden. Das ist nicht ganz trivial, was ist beispielsweise der günstigste Rabatt für zwei große und einen mittleren Cappuccino mit zwei Croissants? (€0.30).

   Kombinieren Sie zum Schluss beide Funktionen zu

   ```
   rechnung :: Bestellung→ Int
   ```

   welche die zu zahlende Gesamtsumme ausrechnet (Zwischensumme abzüglich Rabatt).

   2 Punkte

Willkommen im *Happy Coffeecup!*

Wärmende Getränke:

Kaffee	S	M	L	XL
Café Creme		€1.90	€2.10	
Latte Machiato		€2.30	€2.60	
Milchkaffee		€1.80	€2.00	€2.20
Cappuccino		€2.20	€2.50	
Espresso	€1.80	€1.90	€2.50	
Kakao		€ 2.90	€ 3.10	€3.50

Lecker Gebäck:

Muffins:	
Blaubeer	€1.60
Salted Caramel	€1.80
Cheesecake Cream	€1.70
Double Chocolate	€1.90
Croissant	€1.65
Cookies:	
Vanille	€0.95
Schoko	€1.15

Specials:

- *Happy Breakfast*: Cappuccino mit Croissant: € 3.80 (M), € 4.00 (L)
- *Kaffee und Kuchen*: Café Creme mit Muffin — €0.50 Rabatt!
- *Cookie Binge*: Nimm zwei Cookies, der dritte ist *umsonst*!

Abbildung 1: Die Karte des Café *Happy Coffeecup*

Happy Breakfast M: Cappuccino M mit Croissant	€0.05
Happy Breakfast L: Cappuccino L mit Croissant	€0.15
Cookie Binge Schoko: drei Cookies Schokolade	€1.15
Cookie Binge Vanille: drei Cookies, davon mindestens einer Vanille	€0.95
Kaffee und Kuchen: Café Creme und Muffin	€0.50

Abbildung 2: Die Rabatt-Tabelle des Café *Happy Coffeecup*

Hinweise:

1. Für die Sonderangebote ist es sinnvoll, Hilfsfunktionen zu implementieren, die Getränke oder Gebäck zählen:

   ```
   anzahl_gebaeck :: Bestellung→ Gebaeck→ Int
   anzahl_getraenk :: Bestellung→ Getraenk→ Groesse→ Int
   anzahl_muffins :: Bestellung→ Int
   ```

   Damit lassen sich dann Funktionen implementieren, die zählen, wie oft ein Sonderangebot auftritt: eine *Cookie Binge* Schokolade zum Beispiel tritt bei 3 Schokoladen-Cookies auf — teilen Sie einfach die Anzahl der Schokoladen-Cookies durch 3. Für die *Cookie Binge* Vanille teilen Sie die Anzahl der Vanille-Cookies durch 3, aber Sie können den Rest der Schokoladen-Cookies mit 3 dazuzählen bevor Sie teilen — zwei Cookies Schokolade und ein Cookie Vanille ergeben eine *Cookie Binge* Vanille. Ein *Happy Breakfast L* tritt so oft auf, wie das Minimum aus der Anzahl Croissants und Cappuccini L, aber ein *Happy Breakfast M* tritt so of auf wie das Minimum aus der Anzahl der Croissants minus der *Happy Breakfast L* und Cappuccini M, da ein Croissant nicht für zwei Sonderangebote genutzt werden darf.

   ```
   special_happy_breakfast_l :: Bestellung→ Int
   special_happy_breakfast_m :: Bestellung→ Int
   ...
   ```

   Jetzt müssen Sie nur noch jedes Auftreten mit der Sonderangebote mit dem jeweiligen Rabatt multiplizieren, und alles addieren.

2. Damit Sie Datentypen vergleichen und (auf der Kommandozeile des Interpreters) anzeigen können, müssen Sie hinter der Deklaration des Datentyps die vorerst magische Formel `deriving (Eq, Show)` einfügen (s. Beispiele aus der Vorlesung).

3. Damit die Tests aus der Vorlage sich übersetzen lassen müssen die Konstruktoren folgende Namen haben:

- für `Getraenk`: `CafeCreme`, `LatteMachiato`, `Milchkaffee`, `Cappuccino`, `Espresso`, `Kakao`;
- für `Groesse`: `S`, `M`, `L`, `XL`;
- für `Gebaeck`: `Muffin`, `Croissant`, `Cookie`;
- für `Muffin`: `Blaubeer`, `SaltedCaramel`, `CheesecakeCream`, `DoubleChocolate`;
- für `Cookie`: `Vanille`, `Schokolade`.

2.2 *Die Rechnung bitte!* *5 Punkte*

Seit dem 01.01.2020 müssen Bäckereien einen Bon ausdrucken, um der Steuerhinterziehung vorzubeugen. Wir implementieren dazu die nötigen Funktionen, den Rest kann der Cafébesitzer selber machen.[1]

- Eine Rechnung ist entweder leer oder besteht aus einem Rechnungsposten: ein Name, ein Betrag (in Cent), und der restlichen Rechnung. Implementieren Sie dies in einem linear rekursiven algebraischen Datentyp `Rechnung`, mit den Konstruktoren `Leer` und `Posten`.
- Schreiben Sie jetzt drei Hilfsfunktionen:

  ```
  summe   :: Rechnung→ Int
  center  :: Int→ String→ String
  formatPosten :: Int→ String→ Int→ String
  ```

  `summe` soll die Summe der Posten einer Rechnung berechnen.

  `center` soll eine Zeichenkette innerhalb einer Zeile der angegebenen Länge zentrieren, indem davor und dahinter Leerzeichen eingefügt werden:

  `center 10 "abc"` ⇝ `"␣␣␣abc␣␣␣␣\n"`

  `formatPosten` soll einen Posten (aus Namen und Betrag) innerhalb einer Zeile der angegeben Länge formatieren, indem zwischen Namen und Betrag genügend (aber mindestens ein) Leerzeichen eingefügt werden:

  `formatPosten 20 "Foobaz" 1302` ⇝ `"Foobaz␣␣␣␣␣␣13.02␣EU\n"`

  Das Ergebnis von `center` und `formatPosten` soll nicht länger als `n` Zeichen sein, wobei Ihnen überlassen bleibt wo sie überflüssige Zeichen abschneiden.
- Damit können Sie dann die Hauptfunktion schreiben:

  ```
  rechnung :: String→ String→ Int→ Rechnung→ String
  ```

  `rechnung header footer w r` formatiert eine Rechnung `r` auf die Breite `w`, berechnet die Summe der Rechnung und fügt `header` als zentrierte Kopfzeile und `footer` als zentrierte Fußzeile hinzu:

  ```
  r= Posten "Foo" 1302 (Posten "Baz" (-92) (Posten "Hugo" 0 Leer))
  putStrLn (rechnung "RECHNUNG␣#372" "Die␣Firma␣dankt!" 20 r)  ⇝
     RECHNUNG #372

  Foo          13.02 EU
  Baz          -0.92 EU
  Hugo          0.00 EU
  ====================
  Summe:       12.10 EU

    Die Firma dankt!
  ```

  Das Ergebnis von `rechnung` ist ein String aus Zeilen der Länge `n` (das gilt auch für die Leerzeilen— sie bestehen aus `n` Leerzeichen!).

Hinweis: Die folgenden Funktionen (bekannt aus der Vorlesung, sind in der Vorlage enthalten) können helfen:

```
formatL  :: Int→ String→ String
showEuro :: Int→ String
repeat   :: Int→ String→ String
take     :: Int→ String→ String  -- vordefiniert
```

[1] Der alte Geizkragen, für unsere Mühen aus der ersten Übung gab es gerade mal einen Milchkaffee, und den nur in M.