Universität Bremen

Praktische Informatik 3 WS 18/19

Christoph Lüth
Thomas Barkowsky
Andreas Kästner
Gerrit Marquardt
Tobias Haslop
Matz Habermann
Berthold Hoffmann

4. Übungsblatt

Ausgabe: 06.11.18
Abgabe: 09.11.18

4.1 *Happy Birthday to You!* *10 Punkte*

Schon wieder den Geburtstag der Tante in Schneverdingen vergessen? Schon wieder die Abgabefrist für das Übungsblatt verpasst? Das Leben ist voller Termine und Fristen, da geht einem schnell mal einer durch die Lappen. Das ist allerdings keine Entschuldigung, die Abgabe der PI3-Übungsblätter zu verpassen, deshalb wollen wir hier einen kleinen Terminkalender programmieren.

Der Terminkalender besteht aus zwei Modulen: das Modul Date modelliert Daten, und das Modul BBook modelliert den eigentlichen Kalender.

(1) Ein Datum ist gegeben durch einen algebraischen Datentyp Date und besteht aus Tag, Monat und Jahr. Die Funktion date erzeugt aus zulässiger Kombination von Tag, Monat und Jahr[1] ein Datum:

```
data Date

date :: Int→ Int→ Int→ Maybe Date

date 31 3 1985 ⇝ Just...
date 29 2 1983 ⇝ Nothing
```

Implementieren Sie außerdem eine Vergleichsoperation für Date und eine Stringkonversion, welche das Datum im Format *dd.mm.yyyy* anzeigt, und instanziieren Sie damit die Typklassen Ord und Show:

```
show (fromJust (date 23 3 786)) ⇝ "23.03.0786"
date 15 1 1985 ≤ date 16 1 1985 ⇝ True
date 1 9 1985 ≤ date 11 7 1985 ⇝ False
```

(2) Nachdem wir die Daten im Griff haben können wir das eigentliche Kaldendarium implementieren. Ein Geburtstagskalender ist eine Liste von Einträgen aus Name und Datum. Ein einzelner Eintrag wird durch einen algebraischen Datentyp Entry modelliert, ein Kalendarium enhält dann eine List von diesen:

```
data Entry
data BBook = BBook [Entry]
```

Hierbei hat ein BBook zwei Invarianten:

(i) Die Einträge sind nach aufsteigendem Datum sortiert.

(ii) Jeder Name ist höchstens einmal vertreten.

Zum Erstellen und Verwalten des Geburtstagskalendars dienen die Funktionen

```
newBook :: BBook
addEntry :: BBook→ String→ Int→ Int→ Int→ Maybe BBook
```

Die erste Funktion erstellt einen leeren Kalender, der trivialerweise die Invariante erfüllt; die zweite Funktion muss die Invariante sicherstellen.

[1] Tag und Monat dürfen nicht null oder negativ sein, Monate nicht größer als 12 und Tage nicht mehr als Tage in dem jeweiligen Monat. Das Jahr darf nicht negativ oder größer als 9999 sein. Beachten Sie insbesondere die korrekte Behandlung von Schaltjahren.

Jetzt wollen wir nur noch fragen können, wer wann Geburtstag hat. Zuerst erstellen wir ein Geburtstagskalender:

```
Just b1 = addEntry newBook "Hans Meier" 27 3 1984
Just b2 = addEntry b1 "Horst Schultz" 25 9 1985
Just b3 = addEntry b2 "Tanja Eckhorst" 13 1 1985
Just b4 = addEntry b3 "Markus Felderding" 8 8 1985
Just b5 = addEntry b4 "Horst Schultz" 25 10 1984
```

Mit den folgenden Funktionen

```
getBirthday :: BBook → String → Maybe Date
getNamesBetween :: BBook → Date → Date → [String]
nextBirthday :: BBook → Date → Maybe (String, Date)
listAll :: BBook → [String]
```

erfragen wir den Geburtstag zu einem Namen, alle Personen, die zwischen zwei Daten Geburtstag haben, und nächsten Geburtstag *nach* dem gegeben Datum:

```
getBirthday b5 "Tanja Eckhorst" ⇝ date 13 1 1985
getBirthday b5 "Horst Schultz" ⇝ date 25 10 1984
getNamesBetween b5 (fromJust (date 1 6 1984)) (fromJust (date 28 02 1985))
    ⇝ ["Horst Schultz", "Tanja Eckhorst"]
nextBirthday b5 (fromJust (date 1 1 1985)) ⇝ Just("Tanja Eckhorst", fromJust (date 13 1 1985))
```

Die Funktion `listAll` gibt eine Liste von Strings, welche die einzelnen Einträge repräsentieren, in aufsteigendem Datum zurück:

```
listAll b5 ⇝
 [ "Hans Meier (27.03.1984)",
   "Horst Schultz (25.10.1984)",
   "Tanja Eckhorst (13.01.1985)",
   "Markus Felderding (08.08.1985)" ]
```

Hinweise: Die vorgegebenen Tests lassen sich nicht übersetzen, solange die Instanzen für Ord, Eq und Show für den Typ Date noch nicht definiert sind. Im Zweifelsfall können Sie erst **deriving**...benutzen, um zumindest die Übersetzbarkeit sicherzustellen.