4. Aufgabenblatt zu Funktionale Programmierung vom 06.11.2013.
Fällig: 13.11.2013 / 20.11.2013 (jeweils 15:00 Uhr)

Themen: *Funktionen auf algebraischen Datentypen*

Für dieses Aufgabenblatt sollen Sie die zur Lösung der unten angegebenen Aufgabenstellungen zu entwickelnden Haskell-Rechenvorschriften in Form eines "**literate Script**" in einer Datei mit Namen `Aufgabe4.lhs` ablegen. Versehen Sie wie auf den bisherigen Aufgabenblättern alle Funktionen, die Sie zur Lösung brauchen, mit ihren Typdeklarationen und kommentieren Sie Ihre Programme aussagekräftig. Benutzen Sie, wo sinnvoll, Hilfsfunktionen und Konstanten. Im einzelnen sollen Sie die im folgenden beschriebenen Problemstellungen bearbeiten.

Hinweis: Wenn Sie einzelne Rechenvorschriften aus früheren Lösungen wiederverwenden möchten, so kopieren Sie diese in die neue Abgabedatei ein. Ein `import` schlägt für die Auswertung durch das Abgabeskript fehl, weil Ihre alte Lösung, aus der importiert wird, nicht mit abgesammelt wird. Deshalb: Kopieren statt importieren zur Wiederwendung!

Denken Sie bitte daran, dass Sie für die Lösung dieses Aufgabenblatts ein "literate" Haskell-Skript schreiben sollen!

Auf diesem Aufgabenblatt beschäftigen wir uns mit Bäumen, Suchbäumen und Operationen darauf. Dazu führen wir folgende Datenstrukturen ein:

```
data Person = Pers {
  vorname,
  nachname    :: String,
  geburtstag  :: Geburtstag } deriving (Eq, Show)

type Tag = Int

data Monat = Jan | Feb | Mar | Apr | Mai | Jun
             | Jul | Aug | Sep | Okt | Nov | Dez deriving (Eq, Show)

type Jahr = Int

data Geburtstag = Gtg Tag Monat Jahr deriving (Eq, Show)

type PLZ = Int

data Anschrift = Ans {
  stadt,
  land  :: String,
  plz   :: PLZ } deriving (Eq, Show)

newtype SozNr = Sznr Int deriving (Eq, Show)

type Eintrag = (Person, Anschrift, SozNr)

type Adressbuch = [Eintrag]

data Tree = Nil
            | Node Eintrag Tree Tree deriving (Eq, Show)
```

Für diese Aufgabe gehen wir davon aus, dass unabhängig vom Jahr der Februar stets 28 Tage hat, alle anderen Monate ihre übliche Zahl von Tagen. Weiters gehen wir davon aus, dass es in unserem Kalender ein Jahr 0 gibt. Sozialversicherungsnummern können positiv, negativ oder Null sein.

Ein Baum heißt *Binärbaum*, wenn jeder Knoten einen, zwei oder keinen Nachfolger hat. Wir bezeichnen einen Binärbaum als *Suchbaum*, wenn jeder Knoten (neben möglicherweise weiterer Informationen) einen Schlüsselwert k enthält und alle Schlüsselwerte im linken Teilbaum kleiner als k sind und alle Schlüsselwerte im rechten Teilbaum größer als k sind. Schlüsselwerte können deshalb in Suchbäumen nicht mehrfach auftreten.

Werte des oben eingeführten algebraischen Datentyps `Tree` sind Binärbäume. Abhängig von den Knotenmarkierungen können sie auch Suchbäume sein.

Die `deriving`-Klauseln bei einigen der Typdeklarationen sind ergänzt, um Werte der entsprechenden Typen auf Gleichheit überprüfen und auf dem Bildschirm ausgeben zu können. Im Rahmen der Vorlesung wird die `deriving`-Klausel in Kapitel 8 besprochen.

Schreiben Sie Haskell-Rechenvorschriften für folgende Aufgabenstellungen:

1. Eine Haskell-Rechenvorschrift `erzSB :: Adressbuch -> Tree`. Angewendet auf ein Adressbuch `a` werden die Einträge in einen Suchbaum `t` überführt. Der Resultatbaum soll dabei so aufgebaut sein als wären die Einträge von `a` von links nach rechts durchgegangen und sukzessive in den Resultatbaum eingetragen worden. Als Schüsselkomponente wird dabei die Sozialversicherungsnummer verwendet. Enthält das Adressbuch mehrere Einträge mit gleicher Sozialversicherungsnummer, wird keiner dieser Einträge in den Suchbaum übernommen.

2. Eine Haskell-Rechenvorschrift `istSB :: Tree -> Bool`. Angewendet auf einen Baum t liefert die Wahrheitswertfunktion `istSB` den Wert `True`, falls `t` ein Suchbaum bezüglich des Schlüssels Sozialversicherungsnummer ist, sonst `False`.

3. Eine Haskell-Rechenvorschrift `eintragen :: Eintrag -> Tree -> Tree`. Angewendet auf einen (Adressbuch-) Eintrag `e` und einen Suchbaum `t`, liefert der Aufruf von `eintragen` einen Suchbaum zurück, in den `e` als neues Blatt eingefügt ist, falls es in `t` noch keinen Eintrag mit dem Schlüsselwert von `e` gibt. Kommt der Schlüsselwert von `e` im Suchbaum `t` schon vor, so liefert der Aufruf von `eintragen` den Wert `t`. Ist `t` kein Suchbaum, so liefert der Aufruf von `eintragen` den Wert `Nil`.

4. Eine Haskell-Rechenvorschrift `loeschen :: SozNr -> Tree -> Tree`. Angewendet auf eine Sozialversicherungsnummer `s` und einen Suchbaum `t`, liefert der Aufruf von `loeschen` einen Suchbaum, in dem der Knoten mit Schüsselwert `s` nicht mehr vorkommt. Kommt der Schlüsselwert `s` im Suchbaum `t` nicht vor, so liefert der Aufruf von `loeschen` den Wert `t`. Ist `t` kein Suchbaum, so liefert der Aufruf von `loeschen` den Wert `Nil`.

5. Eine Haskell-Rechenvorschrift `linearisieren :: Tree -> Adressbuch`. Angewendet auf einen Suchbaum `t` liefert der Aufruf von `linearisieren` ein Adressbuch `a`, in dem die Einträge aufsteigend nach Sozialversicherungsnummern angeordnet sind. Ist `t` kein Suchbaum, so liefert der Aufruf von `linearisieren` die leere Liste als Resultat.

6. Eine Haskell-Rechenvorschrift `filtereNachGebTag :: Geburtstag -> Tree -> Adressbuch`. Angewendet auf einen Geburtstag `g` und einen Suchbaum `t` liefert der Aufruf von `filtereGebTag` aufsteigend nach Sozialversicherungsnummern geordnet ein Adressbuch mit den Einträgen aller derjenigen Personen, deren Geburtstag `g` ist. Ist `g` kein gültiges Datum oder `t` kein Suchbaum, so liefert der Aufruf von `filtereNachGebTag` die leere Liste als Resultat.

7. Eine Haskell-Rechenvorschrift `filtereNachPLZundGebTag :: PLZ -> Geburtstag -> Tree -> Adressbuch`. Angewendet auf eine Postleitzahl `p`, einen Geburtstag `g` und einen Suchbaum `t` liefert der Aufruf von `filtereNachPLZundGebTag` aufsteigend nach Sozialversicherungsnummern geordnet ein Adressbuch mit den Einträgen aller derjenigen Personen, bei denen Postleitzahl und Geburtsjahr übereinstimmen. Ist `g` kein gültiges Datum oder `t` kein Suchbaum, so liefert der Aufruf von `filtereNachPLZundGebTag` die leere Liste als Resultat.

Haskell Live

Am Freitag, den 08.11.2013, werden wir uns in *Haskell Live* mit Lösungsvorschlägen für Aufgabenblatt 1 beschäftigen, die (gerne auch) von Ihnen eingebracht werden können, sowie mit einigen der schon speziell für *Haskell Live* gestellten Aufgaben.