PHILIPPS-UNIVERSITÄT MARBURG
Fachbereich Mathematik und Informatik
Prof. Dr. R. Loogen

D-35032 Marburg
Hans Meerwein Straße
Lahnberge

Übungen zur „Praktischen Informatik III“, WS 2005/06

Nr. 3, Abgabe: 15. November 2005 vor der Vorlesung

A. Hausaufgaben

Die Lösungen sollten grundsätzlich schriftlich, Programme zusätzlich auf Diskette oder per E-Mail an Ihre Tutorin abgegeben werden. Die Abgabe ist in Gruppen bis zu zwei Personen erlaubt.

10. Ähnlichkeit von Worten — 6 Punkte

Schreiben Sie Funktionen `similarX :: String -> String -> Bool` mit $\texttt{X} \in \{\texttt{A}, \texttt{B}, \texttt{C}\}$, die prüfen, ob zwei Worte (Typ `String`) *ähnlich* sind. Betrachten Sie als „Ähnlichkeit“ folgende unterschiedlichen Definitionen:

(a) Zwei Worte heißen ähnlich, wenn sie sich nur in einem Buchstaben unterscheiden. / 2

(b) Zwei Worte heißen ähnlich, wenn sie durch Auslassen bzw. Hinzufügen genau eines Buchstabens ineinander übergehen. / 2

(c) Zwei Worte heißen ähnlich, wenn sie durch Vertauschung zweier benachbarter Buchstaben ineinander übergehen. / 2

Wie ändert sich die Funktion, wenn man alle drei Möglichkeiten als Definition von Ähnlichkeit akzeptiert?

Hinweis: Die vordefinierte Funktion `elem :: Eq a => a -> [a] -> Bool` testet, ob ein Element in einer Liste vorkommt.

11. Mengenoperationen — 6 Punkte

Eine Menge kann in Haskell durch eine Liste repräsentiert werden. Implementieren Sie die folgenden Operationen auf Mengen.Setzen Sie, falls möglich, Listenabstraktionen ein.

```
schnitt         :: Eq a => [a] -> [a] -> [a]
vereinigung     :: [a] -> [a] -> [a]
differenz       :: Eq a => [a] -> [a] -> [a]
potenzmenge     :: [a] -> [[a]]
keine_duplikate :: Eq a => [a] -> [a]
```

Nützlich sind außerdem die folgenden Testfunktionen, die ebenfalls implementiert werden sollen:

```
disjunkt  :: Eq a => [a] -> [a] -> Bool
teilmenge :: Eq a => [a] -> [a] -> Bool
gleich    :: Eq a => [a] -> [a] -> Bool
```

B. Mündliche Aufgaben

12. Wortsuche

    Schreiben Sie eine Funktion `isIn :: String -> String -> Bool`, die zu einem Wort und einem Text (beide vom Typ `String`) prüft, ob das Wort in dem Text vorkommt.

13. Listenpermutationen

    Definieren Sie unter Verwendung von Listenabstraktionen

    (a) eine Funktion `removeList :: [a] -> [(a,[a])]`, die zu einer Liste alle Möglichkeiten bestimmt, ein Element zu entfernen. Als Ergebnis soll eine Liste aller möglichen Paare (entferntes Element, Restliste) zurückgegeben werden.

    Beispiel: `removeList [1,2,3] =>* [(1,[2,3]), (2,[1,3]), (3,[1,2])]`

    (b) eine Funktion `perms :: [a] -> [[a]]`, die zu einer Liste alle möglichen Permutationen der Listenelemente bestimmt.

    Beispiel:

    `perms [1,2,3] =>* [[1,2,3],[1,3,2],[2,1,3],[2,3,1],[3,1,2],[3,2,1]]`

14. Sparschwein

    Implementieren Sie einen abstrakten Datentyp Sparschwein mit den Operationen:

```
emptyPig :: PiggyBank                          -- leeres Sparschwein
add      :: Coin -> PiggyBank -> PiggyBank     -- Einwerfen einer Muenze
shake    :: PiggyBank -> (PiggyBank, Coin)     -- Herausschütteln einer Muenze
isEmpty  :: PiggyBank -> Bool                  -- Test auf leeres Sparschwein
break    :: PiggyBank -> Money                 -- Aufbrechen des Sparschweins
```

    Dabei seien die Datentypen `Coin` und `Money` wie folgt definiert:

```
data Coin  = OneCent | TwoCents | FiveCents | TenCents | FiftyCents
           | OneEuro | TwoEuros
type Money = Int -- Geldwert in Cents
```