Technische Universität München
Institut für Informatik
Prof. Tobias Nipkow, Ph.D.
L. Noschinski, L. Hupel, Dr. J. Blanchette, M. Eberl

WS 2014/15
19.12.2014
Abgabe: 09.01.2015

Einführung in die Informatik 2
11. Übung

Aufgabe G11.1 Assoziationslisten

Wir haben bereits in Aufgabe H2.2 Assoziationslisten `Eq k => [(k,v)]` als eine Möglichkeit kennengelernt, Maps mit Schlüsseltyp k und Werttyp v darzustellen.

Um zu verhindern, dass der Benutzer unsinnige Assoziationslisten erstellen kann (z.B. mit mehreren Einträgen für einen Schlüssel), wollen wir die Implementation in einem Modul verstecken.

Definieren Sie ein Modul `AssocList`, das einen Typ `Map k v` und die folgenden Funktionen bereitstellt. Andere Typen und Funktionen sollen nicht exportiert werden! Insbesondere sollen die Konstruktoren von `Map` nicht zur Verfügung stehen.

```
newtype Map k v = ...
empty :: Eq k => Map k v
insert :: Eq k => k -> v -> Map k v -> Map k v
lookup :: Eq k => k -> Map k v -> Maybe v
delete :: Eq k => k -> Map k v -> Map k v
keys :: Eq k => Map k v -> [k]
```

(Bei einem Aufruf von `insert` mit bereits existierendem Schlüssel soll der alte Wert überschrieben werden.)

Die Maps sollen intern als Assoziationslisten dargestellt werden.

Hinweis: Die Prelude bietet auch eine `lookup`-Funktion an. Um Namenskonflikte zu vermeiden, können Sie mittels `import Prelude hiding (lookup)` die vordefinierte Funktion in Ihrem Modul verstecken.

Aufgabe G11.2 Na warte, Schlange!

Eine *Warteschlange* ist eine klassische Datenstruktur mit FIFO-Semantik ("first in, first out"). Neue Elemente werden am Ende der Schlange hinzugefügt, alte Elemente werden am Kopf entfernt. Schreiben Sie ein Modul, dass einen passenden Datentyp `Queue` mit geeigneten Operationen definiert. Die Implementationsdetails sollen versteckt werden.

Aufgabe G11.3 Korrektuz (optional)

Schreiben Sie eine Funktion

```
fixTypo :: [String] -> String -> Either [String] String
```

die als Eingabe eine Vokabelliste *vocabs* und ein Wort *word* nimmt.

Das übergebene Wort gilt als *falsch geschrieben*, wenn es nicht in der Vokabelliste *vocabs* steht. Zwei Wörter sind *quasi-identisch*, wenn sie die gleiche Länge haben und das eine Wort aus dem anderen hergeleitet werden kann, indem genau ein Buchstabe durch einen anderen ersetzt wird.

Ist nun *word* falsch geschrieben, so soll `Left` *vs* zurückgegeben werden, wobei *vs* die Liste der in *vocabs* vorkommenden Wörter ist, die quasi-identisch zu *word* sind. Andernfalls wird `Right` *word* zurückgegeben.

Beispiele:

```
vocabs = ["i", "more", "more", "now", "want", "won", "wow"]

fixTypo vocabs "i"    == Right "i"
fixTypo vocabs "mose" == Left ["more"]
fixTypo vocabs "and"  == Left []
fixTypo vocabs "wob"  == Left ["won", "wow"]
```

Hinweis für den Wettbewerb: Am 09.01.2015 ist Abgabeschluss für die zweiwöchige Wettbewerbsaufgabe, die auf dem vorherigen Blatt gestellt worden ist. Nutzen Sie für Ihre endgültige Abgabe bitte die speziell auf der Übungsseite bereitgestellte Schablone und laden diese im Übungssystem unter *Wettbewerb 10/11* hoch. **Wettbewerbsbeiträge, die als Teil der Hausaufgaben auf diesem Blatt hochgeladen werden, werden nicht berücksichtigt.**

Aufgabe H11.1 Prioritäten setzen

Schreiben Sie ein Haskell-Modul `PrioQueue`, welches eine Prioritätswarteschlange implementiert. Dieses Modul soll einen abstrakten Typ `Queue a` und die Funktionen

```
empty :: Queue a
insert :: a -> Integer -> Queue a -> Queue a
pull :: Queue a -> Maybe (a, Queue a)
```

exportieren.

Die Semantik soll wie folgt sein:

- `empty` ist die leere Warteschlange
- `insert` x p q fügt x mit Priorität p in die Warteschlange q ein. Ein Element kann mehrfach in einer Warteschlange enthalten sein.
- `pull` q: Ist q leer, so wird `Nothing` zurückgegeben. Andernfalls ist das Ergebnis `Just (`x`,` q'`)`, wobei x ein Element aus q ist, dessen Priorität kleiner oder gleich der Prioritäten der anderen Elemente in q ist. q' ist die Warteschlange, die aus q entsteht, wenn man das Element x entfernt.

Aufgabe H11.2 CAPTCHA II

In Aufgabe H10.3 haben Sie einen Client für ein einfaches CAPTCHA-Protokoll implementiert. In dieser Aufgabe implementieren Sie den zugehörigen Server:

```
server :: IO (String, String) -> IO ()
```

Die Eingabe ist eine IO-Aktion, die jederzeit ein Paar aus auszugebendem Captcha und erwarteter Lösung liefert. Die Rückgabe ist eine IO-Aktion, die einen Server auf dem Port 8080 startet, auf einen Client wartet, das letzte Woche beschriebene Protokoll implementiert und im Erfolgsfall den Benutzernamen auf die Standardausgabe schreibt. Im Fehlerfall soll nichts passieren. Nach Beendigung der Transaktion soll der Server einen neuen Client akzeptieren.

Im Gegensatz zur Protokollspezifikation von letzter Woche brauchen Sie nur die Kommandos `REGISTER` und `SOLUTION` zu implementieren. Sie brauchen sich außerdem nicht um den 10-Sekunden-Timeout vom Client zu kümmern. Sofern allerdings ein ungültiges Kommando empfangen wird, muss die Verbindung getrennt werden.