Arbeitsblatt: Fehlerbehandlung in Haskell

Maybe

Haskell definiert den Datentyp Maybe:

```
data Maybe a = Nothing | Just a
```

Maybe beschreibt Werte, die an- oder abwesend sein können. Maybe a hat zwei Wertkonstruktoren: Nothing steht für einen nicht vorhanden Wert und Just a steht für einen vorhandenen Wert vom generischen Typ a.

Wir betrachten nun eine Funktion, die gegeben eine Emailadresse aus einer Liste von Studierenden die Matrikelnummer des entsprechenden Studierenden raussucht:

```
data Student = Student { email :: String, matrikelNr :: Int}

findMatrikelNr :: String -> [Student] -> Int
```

Es gilt zu beachten, dass z.B. infolge eines Tippfehlers in einer Websuchmaske, nach einer Emailadresse gesucht werden könnte, für die kein Student in der Liste ist. In manchen Sprachen ist es nun üblich mit dem Wert -1 zu signalisieren, dass nichts gefunden werden konnte. **Was für ein Schwachsinn!**
Diese sonderbare Eigenheit muss irgendwo in der Dokumentation festgehalten werden und kann gerne vergessen werden. Gleiches gilt übrigens für den Rückgabewert null in Java! Tony Hoare[1], Turing Award Gewinner, Erfinder von Quicksort ☺ und dem Nullpointer ☹, hat sich für die Erfindung des Nullpointers entschuldigt und spricht von seinem "billion dollar mistake"!

Um präziser und maschinencheckbar auszudrücken, dass möglicherweise keine Matrikelnummer gefunden werden könnte, schreibt man in Haskell:

```
findMatrikelNr :: String -> [Student] -> Maybe Int
```

Jetzt ist vom Typ her klar, dass möglicherweise kein Wert gefunden werden kann und der Aufrufer ist gezwungen mit dieser Situation umzugehen.

Der Maybe Typ findet in der Haskell Standard Bibliothek häufig Verwendung. Ein Klassiker ist die Funktion find, die mit einem Prädikat die Liste durchsucht und den ersten passenden Wert zurückgibt, oder aber melden muss, dass kein Wert gefunden werden konnte:

```
find :: (a -> Bool) -> [a] -> Maybe a
```

Aufgaben

a) Die Funktion head :: [a] -> a wirft eine Exception falls Sie auf eine leere Liste angewendet wird. Das Programm wird in Folge abgebrochen (ungünstig wenn Sie damit eine Rakete steuern). Viel robuster wäre es im Typ klar zu sagen, dass ein Wert nicht vorhanden sein kann!
Implementieren Sie zum Aufwärmen die Listenfunktion safeHead.

```
safeHead :: [a] -> Maybe a
```

b) Wenn Sie Aufgabe a) gelöst haben, sollte es ein leichtes Spiel sein jetzt die Funktion safeTail zu implementieren:

```
safeTail :: [a] -> Maybe [a]
```

c) Die Listenfunktion maximum hat dasselbe Problem wie head und tail: Bei einer leeren Liste kann kein grösstes Element zurückgeben werden. Implementieren Sie die Funktion safeMax rekursiv:

```
safeMax :: (Ord a) => [a] -> Maybe a
```

---

[1] http://en.wikipedia.org/wiki/Tony_Hoare