Weitere Programmiersprache
- Serie 4 -

Aufgabe 4-1 (Abgabe!)

a) Schreiben Sie eine Funktion `isPangram :: String -> Bool`, die überprüft, ob der eingegebene String alle Buchstaben des Alphabets mindestens einmal enthält. Einzelne Buchstaben können dabei mehrfach vorkommen. Das Alphabet besteht aus den Buchstaben a bis z (keine Sonderzeichen, Groß und Kleinschreibung wird ignoriert).

Besipiele:

```
the quick brown fox jumps over the lazy dog
```

```
franz jagt im komplett verwahrlosten taxi quer durch bayern
```

b) Schreiben Sie eine Funktion namens `abbreviate :: String - > String`, die einen gegebenen String (mit Leerzeichen) abkürzt. Sie können davon ausgehen, dass nur der Anfang jedes Worts groß geschrieben ist. Zum Prüfen, ob ein Buchstabe groß geschrieben ist, können Sie die Funktion `isUpper` nutzen. Importieren Sie die Funktion, indem Sie die Zeile `import Data.Char (isUpper)` einfügen.

Beispiele:

```
As Soon As Possible => ASAP
```

```
Computer-Aided Design => CAD
```

```
Erasable Programmable Read-Only Memory => EPROM
```

Aufg. 4.1 bitte bis 22.10. 23:59 Uhr in Moodle hochladen! Max. 2 Punkte!

Dateiname: Serie4.hs, als erste Zeile fügen Sie bitte ein: `module Serie4 where`

Aufgabe 4-2 (Typen)

Geben Sie den Typ der folgenden Ausdrücke/Funktionen an:

```
a) fun2 n = [\a -> (a, b, a*b) | b <- [1..n]]
b) ['a', 'b'] ++ "cd"
c) fun3 (x:xs) = x
d) fun4 (x:xs) = x + 1
e) fun5 (x:xs) = (+) x
f) fun6 (x:xs) = fun5 x
g) fun7 (x:xs) = fun5 xs
```

Aufgabe 4-3 (Mergesort)

Erstellen sie eine Funktion `merge`, die rekursiv zwei sortierte Listen zusammenfügt.

```
merge [1,3,5] [2,4,6] = [1,2,3,4,5,6]
```

Erstellen Sie eine Funktion `half`, die eine Liste rekursiv in der Mitte halbiert.

```
half [1..10] = ([1,2,3,4,5,6],[7,8,9,10,11,12])
```

Nutzen Sie die beiden Funktionen, um eine Funktion `msort` zu schreiben, die eine unsortierte Liste durch Mergesort sortiert, z.B. `msort [10,9..1] = [1,2,3,4,5,6,7,8,9,10]`

Besprechung: 23/28.10.2024

Aufgabe 4-4 (Quicksort)

Am Beispiel Quicksort werden die Eigenschaften von funktionalen Programmierparadigmen deutlich.

1. Implementieren Sie Quicksort in Java
2. Implementieren Sie Quicksort in Haskell

Vergleichen Sie die beiden Implementierungen.

Aufgabe 4-5 (Standard-Funktionen)

Schreiben Sie die folgenden Funktionen in Haskell ohne entsprechende vordefinierte Funktionen zu verwenden:

```
-- Gibt die ersten n Elemente einer Liste zurück
myTake :: Integer -> [a] -> [a]

-- Entfernt die ersten n Elemente einer Liste
myDrop :: Integer -> [a] -> [a]


-- Gibt alle Elemente zurück die das gegebene Prädikat erfüllen
myFilter :: (a -> Bool) -> [a] -> [a]
```

Aufgabe 4-6 (Funktionen)

Schreiben Sie die folgenden Funktionen in Haskell und überlegen Sie sich vorher die Signatur:

1. `moreOften x y xs` ist True wenn x öfter in xs ist als y.
2. `isPrime n` ist True, wenn n eine Primzahl ist. Verwenden Sie dazu das Sieb des Erathostenes.

Besprechung: 23/28.10.2024