

Softwaretechnik und Programmiersprachen SS11 Freiwillige Übung

Um diese Aufgaben zu bearbeiten müssen Sie auf Ihrem PC Haskell installieren. Wir empfehlen die Haskell-Plattform zu benutzen. Die Haskell-Plattform ist ein relative grosser Download, aber für den Einstieg am einfachsten. Die Download-Seite ist auf www.haskell.org verlinkt.

Haskell-Source-Code aus der Vorlesung findet man unter
http://www.stups.uni-duesseldorf.de/~fontaine/vl_fkt.

Aufgabe 1 (leicht)

Starten Sie `ghci`. Mit `:browse Prelude` können Sie sich die Funktionen anzeigen lassen im Prelude Modul definiert sind und standardmässig geladen werden. Mit `import Data.List` können Sie z.B. zusätzlich das List-Module importieren. Stellen Sie durch raten, nachdenken und ausprobieren fest was folgende Funktionen aus `Data.List` machen und programmieren Sie die Funktionen dann nach.

```
replicate :: Int -> a -> [a]
length :: [a] -> Int
take :: Int -> [a] -> [a]
drop :: Int -> [a] -> [a]
(!!) :: [a] -> Int -> a
(+++) :: [a] -> [a] -> [a]
map :: (a -> b) -> [a] -> [b]
filter :: (a -> Bool) -> [a] -> [a]
```

Tipp: Man kann auch auf <http://haskell.org/hoogle> nach Funktionen suchen und dort ist auch der "echte" Sourcecode verlinkt. Aber bitte, erst selbst programmieren.

Aufgabe 2 (leicht)

Die folgenden Funktionen sind Brot-und-Butter in Haskell.

```
foldr :: (a -> b -> b) -> b -> [a] -> b
foldr fkt st [] = st
foldr fkt st (h:t) = fkt h (foldr fkt st t)

foldl :: (a -> b -> a) -> a -> [b] -> a
foldl f z [] = z
foldl f z (x:xs) = foldl f (f z x) xs
```

Berechnen Sie von Hand die Ausdrücke `foldr (*) 1 [1,2,3]` und `foldl (+) 0 [5,2,4]` und geben Sie wichtige Zwischenschritte an. (Hinweis: Ihre Zwischenschritte beschreiben sehr wahrscheinlich nicht das, was Haskell "wirklich" macht (lazy Auswertung), aber für diese Aufgabe ist das erstmal nicht schlimm.) `foldl` ist *tail-rekursive*. Was heisst das in der Praxis ?

Aufgabe 3 (mittel, für Anfänger knobelig)

Implementieren Sie `length`, `map` und `(++)` mit Hilfe von `foldr`.

Aufgabe 4 (leicht)

Debuggen Sie folgende Quicksort Funktion:

```
module QuickSortBuggy
where
import qualified Data.List
import Test.QuickCheck

quickSort :: (Ord a) => [a] -> [a]
quickSort [] = []
quickSort (median:rest) = left ++ [median] ++ right
  where left = quickSort [e | e<-rest, e < median]
        right = quickSort [e | e<-rest, e > median]

prop_quickSort :: [Int] -> Bool
prop_quickSort l = quickSort l == Data.List.sort l

main = quickCheck prop_quickSort
```

Rufen Sie zum testen entweder `quickSort` direkt auf oder starten Sie `main`.

Aufgabe 5 (leicht)

Geben Sie eine Funktion an, die folgenden Type hat :

$(a \rightarrow b) \rightarrow (a \rightarrow c) \rightarrow a \rightarrow (b, c)$ Was macht Ihre Funktion ? Machen alle Funktionen, mit diesen Type, das gleiche ?

Aufgabe 6

$(\$)$:: $(a \rightarrow b) \rightarrow a \rightarrow b$ ist die Funktionsanwendung als Infixoperator:

$(\$)$ $f\ x = f\ x$

D.h. $f\ x$ kann auch als $f\ \$\ x$ geschrieben werden. Der $\$$ -Operator hat eine niedrige Priorität und ist rechts-assoziativ, wodurch man Klammern sparen kann.

Vereinfachen Sie den Ausdruck : `show (head (tail (tail [1,2,3])))` mit Hilfe von $\$$.