

Softwaretechnik und Programmiersprachen WS-2011/2012 Freiwillige Übung

Um diese Aufgaben zu bearbeiten müssen Sie auf Ihrem PC Haskell installieren. Die Haskell-Plattform umfasst alles was man zum Einstieg benötigt (Glasgow-Haskell-Compiler (GHC), ghci, Standard-Libraries, Tool-Chain, cabal -tool, *ldots*). Wir empfehlen die Haskell-Plattform zu benutzen. Die Download-Seite zur Haskell-Plattform ist über www.haskell.org verlinkt.

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 `:m Data.List` können Sie z.B. zusätzlich das List-Module öffnen. 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]
```

Aufgabe 2 (leicht)

Gegeben seien folgende Funktionen:

```
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.)

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 = do
  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, die diesen Type, haben 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 $\$$.

Aufgabe 7 (leicht)

Um Programme zu schreiben die Eingaben und Ausgaben verarbeiten benötigt man die IO-Monad. Was das genau bedeutet wird noch in der Vorlesung erklärt (vielleicht). Hier erstmal ein Beispiel:

```

module Main
where
import System
import Control.Monad

main = do
  args <- getArgs
  putStrLn "CMD-Line Args :"
  mapM_ putStrLn args
  when ((length args) < 2) $ do
    putStrLn "Bitte mit zwei Argumenten starten"

```

```
    error "nicht genug Argumente"
let fname= args !! 0
    line= read (args !! 1)
putStrLn $ "lese Datei " ++ fname
contents <- readFile fname
-- putStrLn contents
putStrLn $ "Zeile " ++ show line ++ " von " ++ fname ++ ":"
putStrLn $ lines contents !! line
```

Testen Sie das Programm mit `ghci` (Man kann Commandline-Argumente simulieren, siehe `?`) und erzeugen Sie mit `ghc -O3 IODemo.hs -o iodemo` ein Binary-Executable und testen Sie dieses. Verbessern Sie die Effizienz des Programms indem sie statt Strings das Modul `Data.ByteString` verwenden.

Um IO actions mit `ghci` direkt auszuprobieren eignet sich die explizite Klammerung mit `{ ; }` besser.
Z.B.: `*Main> do { x<- getLine ; putStrLn "test" ; putStrLn x }`