Programmierung

01. Übungsblatt

Zeitraum: 10. – 13. April 2017

Beachten Sie die Übungsverlegungen zum 14. April (s. LV-Website)

Übung 1

- Schreiben Sie in Haskell eine Funktion `fac :: Int -> Int`, so dass `fac n` die Fakultät `n!` berechnet, also $\prod_{i=1}^{n} i$.
- Schreiben Sie nun eine Funktion `sumFacs :: Int -> Int -> Int`, so dass `sumFacs n m` den Wert $\sum_{i=n}^{m} i!$ berechnet.

Übung 2

Die Folge der Fibonacci-Zahlen $f_0, f_1, \ldots$, ist definiert durch $f_0 = 1$, $f_1 = 1$, und $f_{i+2} = f_i + f_{i+1}$ für jedes $i \in \mathbb{N}$. Implementieren Sie eine Haskell-Funktion, die für die Eingabe i die Zahl f_i berechnet.

Übung 3

Schreiben Sie die folgenden Haskell-Funktionen:

- eine Funktion `prod :: [Int] -> Int`, die die Zahlen in einer gegebenen Liste aufmultipliziert.
- eine Funktion `rev :: [Int] -> [Int]`, welche eine gegebene Liste umkehrt.
- eine Funktion `rem :: Int -> [Int] -> [Int]`, so dass `rem x xs` die Liste ist, die aus `xs` hervorgeht, indem alle Vorkommen von `x` gelöscht werden.
- eine Funktion `isOrd :: [Int] -> Bool`, welche für eine gegebene Liste prüft, ob sie (aufsteigend) sortiert ist.
- eine Funktion `merge :: [Int] -> [Int] -> [Int]`, die zwei aufsteigend sortierte Listen zu einer aufsteigend sortierten Liste vereinigt.

Zusatzaufgabe 1

Implementieren Sie die (unendliche) Liste `fibs :: [Int]` der Fibonacci-Zahlen $f_0, f_1, \ldots$.

Zusatzaufgabe 2

In einem vollen Binärbaum ist jeder Knoten entweder ein Blatt, oder er hat zwei Kindknoten. Implementieren Sie eine Haskell-Funktion, welche für $n \in \mathbb{N}$ die Anzahl der vollen Binärbäume mit Knotenzahl n berechnet.

Zusatzaufgabe 3

Machen Sie sich mit ghc(i) (dem Glasgow Haskell Compiler, `https://www.haskell.org/ghc`) vertraut, insbesondere mit den Befehlen `:type`, `:info`, `:browse` und `:?`, und mit der Suchmaschine `https://haskell.org/hoogle`.