1. Übung (06.–10. April 2020)

Programmierung

Aufgabe 1 (AGS 12.1.10)
In den folgenden Aufgaben sollen Haskell-Funktionen implementiert werden. Es bietet sich an einen Haskell-Compiler (z.B. `ghc`) oder -Interpreter (z.B. `ghci` (Teil von `ghc`)) zu verwenden um die Funktionen zu testen. Mittels `ghci` lassen sich solche Funktion importieren und interaktiv ausführen, indem entweder

- `ghci` mit dieser Datei gestartet wird (von der Kommandozeile `ghci <Pfad zu dieser Datei>`), oder
- nach dem Start in `ghci` der Befehl `:load <Pfad zu dieser Datei>` ausgeführt wird.

Wenn man so beispielsweise eine Datei mit dem Inhalt

```
sum3 :: Int -> Int -> Int -> Int
sum3 x y z = x + y + z
```

lädt, kann man danach im Interpreter das Ergebnis von `4 + 5 + 6` mit dem Befehl `sum3 4 5 6` berechnen lassen. Solche Datein, die Haskell-Funktionen beinhalten, nennen wir *Module*; man verwendet für Module üblicherweise die Dateiendung `.hs`.
Neben `:load` unterstützt `ghci` noch weitere Befehle, die mit einem `:` beginnen, diese kann man mit `:help` anzeigen. Installieren Sie `ghci` und machen Sie sich damit vertraut! Was leisten die Befehle `:type`, `:info`, `:browse` und `:?`?

Hinweis Sie können die Funktionen der folgenden Aufgaben in einem Modul implementieren. Bitte notieren Sie über jeder Funktion die Aufgabe bzw. den Aufgabenteil, den sie lösen soll, in einem Kommentar. Z.B. soll in der Zeile über `sumFacs :: Int -> Int -> Int` der Kommentar `-- Aufgabe 1 (b)` stehen.

Aufgabe 2 (AGS 12.1.5)

(a) Schreiben Sie eine Funktion `fac :: Int -> Int`, so dass `fac n` die Fakultät von `n`, also $\texttt{n}! = \prod_{i=1}^{\texttt{n}} i$, berechnet.

(b) Schreiben Sie nun eine Funktion `sumFacs :: Int -> Int -> Int`, so dass `sumFacs n m` den Wert $\sum_{i=\texttt{n}}^{\texttt{m}} i!$ berechnet.

Aufgabe 3 (AGS 12.1.6)
Die Folge der Fibonacci-Zahlen f_0, f_1, ..., 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 Funktion, die für die Eingabe i die Zahl f_i berechnet.

Zusatzaufgabe 1 (AGS 12.1.9)
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.