Prof. Dr. Visvanathan Ramesh
PD. Dr. Arne Nägel
Lukas Müller

Goethe-Universität Frankfurt am Main
Institut für Informatik

Programmierparadigmen und Compilerbau

Sommersemester 2022

Aufgabenblatt Nr. 2

Abgabe Mittwoch 18.05.2022, 10 Uhr in Moodle!

Aufgabe 1 - Rekursion **(1.5 + 2.5 = 4 Punkte)**

Die Eulersche Zahl e ist eine sehr wichtige Konstante in der Mathematik. Die Zahl e kann durch verschiedene Funktionen angenähert werden werden. Eine der Funktionen ist diese:

$$e(n) = \sum_{i=0}^{n} \frac{1}{i!} = 1 + \frac{1}{1!} + \frac{1}{2!} + ... + \frac{1}{n!}$$

a) Schreiben Sie eine Funktion `e1`, die die Fakultäts-Funktion aus der Vorlesung verwendet. Die Fakultäts-Funktion soll hier als Hilfs-Funktion genutzt werden. Die Funktion `e1` soll eine Zahl übergeben bekommen, bis dahin soll die Eulersche Zahl angenähert werden.

b) Schreiben Sie die Funktion aus a) so um, dass diese die Kriterien der Endrekursion erfüllt. Schreiben Sie hierfür eine einzige Funktion `e2`.

Aufgabe 2 - Listen Funktionen **(4 Punkte)**

Definieren Sie in Haskell eine Funktion `g` die einen String erwartet und einen neuen String wie folgt erzeugt:

- Zuerst alle kleinen Buchstaben aus dem eingegeben String entfernt.
- Dann alle Großbuchstaben durch Kleinbuchstaben ersetzt. (Diese sollen nicht entfernt werden)
- Jede Zahl durch entsprechend viele Zeichen der Zahl ersetzten
- Als Rückgabe wird ein String erwartet.

Zum Beispiel soll `"h22f55555"` für die Eingabe `"Hall0F52F5"` berechnet werden. Sie dürfen `map`, `filter` und weite Listenfunktionen verwenden, aber auf Pattern Matching verzichten. Die elementaren Funktionen (z.B. `digitToInt`) dürfen verwendet werden.

Aufgabe 3 - Pattern Matching **(1 + 1 + 2 + 2 = 6 Punkte)**

Implementieren Sie in Haskell die folgenden Funktionen auf Listen im Wesentlichen unter Verwendung von Pattern-Matching und Rekursion– d.h. Sie dürfen Hilfsfunktionen definieren und nutzen, allerdings ist die Verwendung von Listenfunktionen wie z.B. `map`, `concat`, `replicate` usw. **verboten**.

a) Eine Funktion `f1`, die eine Liste erwartet und je zwei benachbarte Elemente vertauscht. Enthält die Liste eine ungerade Anzahl an Elementen, so bleibt die Position des letzten Elements unverändert.
Zum Beispiel soll für die Liste `[1..9]` das Ergebnis `[2,1,4,3,6,5,8,7,9]` berechnet werden.

b) Eine Funktion `f2`, die eine Liste von Listen von Zahlen erwartet und die immer das erste Element jeder Teilliste in einer neuen Liste zurückgibt.
Zum Beispiel soll für eine Liste `[[1,2,3],[4,5,6],[7,8,9]]` das Ergebnis `[1,4,7]` berechnet werden.

c) Eine Funktion `f3`, die eine Liste von Listen von Zahlen erwartet und immer das erste und letzte Element der Teilliste in eine Liste packt. Wenn nur ein Element in der Teilliste ist, soll nur das Element in die Liste gepackt werden.
Zum Beispiel soll für die Eingabe `[[1,2,3],[4..10],[3],[7..9]]` das Ergebnis `[[1,3],[4,10],[3],[7,9]]` berechnet werden.

d) Eine Funktion `f4`, die eine Liste von Listen von Zahlen erwartet und alle Zahlen zusammenaddiert, aber alle Listen die weniger als 3 Elemente haben nicht berücksichtigt werden.
Zum Beispiel soll für die Eingabe `[[1,2],[1,2,3],[],[4..6]]` das Ergebnis `21` berechnet werden.

Aufgabe 4 - Selbstdefinerte Datentypen (1 + 1 + 2 = 6 Punkte)

Sie wollen ein kleines Spiel programmieren. Die Spieler werden in Haskell als `Charakter`[1] dargestellt. Jeder Spieler hat einen `Namen`, eine `Waffe` und ein `Inventar`. Hier die Datenstruktur wie ein Charakter in Haskell dargestellt wird:

```
data Inventar = Inventar [Object] deriving(Show)
data Waffe = Waffe Name Staerke deriving(Show)
data Charakter = Charakter Name Waffe Inventar deriving(Show)
type Name = String
type Staerke = Int
type Object = String
```

a) Erstellen Sie sich anhand der oben definierten Datenstruktur einen Charakter. Dieser soll dabei mindestens 3 Objekte in seinem Inventar haben. Der Name des Charakter soll ihrem Vornamen entsprechen. Den Rest dürfen Sie sich selber aussuchen.
Speicher Sie Ihren Charakter in einer Funktion `mein_Charakter :: Charakter`.

b) Erstellen Sie eine Funktion `wechsel_waffe :: Charakter -> Waffe -> Charakter`, die den Waffenwechsel des Charakters simulieren soll.

c) Erstellen Sie eine Funktion `hole_aus_inventar :: Charakter -> String -> Charakter` die ein Object aus dem Inventar holt und dann den Charakter zurückgibt mit dem kleineren Inventar. Wenn das übergebene Object nicht in dem Inventar ist, soll es unverändert zurückgegeben werden.

[1] Der Ausdruck `deriving(Show)` führt dabei dazu, dass Objekte des entsprechenden Datentyps angezeigt werden kann