Datum: 30. November 2021

Funktionale Programmierung mit Haskell

Wichtige Quellen:

- https://www.haskell.org (Was sonst?)
- Simon Thompson. Haskell: The Craft of Functional Programming. Addison Wesley, 3rd edition, 2011.
- Bird, R.; Wadler, P.: Einführung in die funktionale Programmierung. München: Hanser 1992.

Warum Haskell?

- rein funktional, d.h. Definition von Ausdrücken, die ausgewertet werden (Definiendum:= „das, was bestimmt werden soll“ und Definiens:= „das, was das Vorgenannte beschreibt“)
- Pattern Matching `(x:xs)` oder `[x|x<-[1..10], x<5]`
- Lazy Evaluation
- stark (strict) und statisch typisiert
- keine Seiteneffekte (referenzielle Transparenz): eine Auswertung (Reduktion) liefert immer bei gleichen Werten das gleiche Ergebnis und der Wert des Ergebnisses hängt nur vom Wert der Eingaben ab, keine Änderungen von Objekten

```
1 public static void Main (string[] args) {
2       int i = 0, j = 0;
3       if ((i++ == 1) && (j++ == 1)) {
4       }
5       Console.WriteLine(i);
6       Console.WriteLine(j);
7     }
```

... und in Haskell:

```
1 -- foo bekommt zwei Werte 0 und 0 und prueft, ob beide Werte nach
     dem Inkrementieren genau 1 sind und gibt diese dann aus
2 foo:: Int -> Int -> (Int,Int)
3 foo i j | (i+1 == 1) && (j+1 == 1) = (i+1,j+1)
4         | otherwise = error "i␣und␣j␣sind␣nicht␣Null"
```

- eine Vielzahl von Packages (Open Source!)
- Currying
- Lambda-Kalkül

Am besten wir machen uns die Sachen beim Üben klar. ;-))

Los geht's!

1. Aufgabe:
Einfache Funktionen, Basistypen

Die Aufgaben dienen der Erinnerung
und Übung. Haskell ist schon lange her. ☺

- Schreibe jeweils Funktionen für die Berechnung der Summe, Produkt, Durchschnitt, geometrisches Mittel, harmonisches Mittel zweier und dreier Zahlen. (Signatur nicht vergessen!)
- Schreibe eine Funktion, die für gegebenes n die Summe der Zahlen von 1 bis n zurück gibt (Rekursion!).
- Programmiere die boolschen Funktionen, AND, EXOR (wer mag kann auch XNOR nehmen) und die Äquivalenz.
- Es ist ein Halb- und Volladdierer zu programmieren.
- Schreibe eine Funktion, die für gegebene n und k den Binomialkoeffizienten $\binom{n}{k}$, n über k berechnet.
- Fakultäts- und Fibonaccizahl berechnen (Das muss natürlich sein. Zum Vergleichen gegenüber C #.)
- Die Ackermann-Funktion ist definiert wie folgt: `ack(0,m) = m+1, ack(n+1,0) = ack(n,1), ack(n+1,m+1) = ack(n,ack(n+1,m))`. Gib `ack` in Haskell an. (Hinweis: Die Ackermannfunktion wird sehr schnell größer. Zum Testen sollte etwas größeres als `ack(2,2)` nur auf eigenes Risiko berechnet werden.)
- Schreibe eine Funktion, die für gegebene b und k die k-fache Potenz von b berechnet.
- Der ggt (n,m) ist zu ermitteln.
- Schreibe eine Funktion, die für gegebene m und n die Zahl m genau n mal aufsummiert.
- Benutze die beiden vorherigen Funktionen die für gegebene r, s und t die Summe $\sum_{t=1}^{r} = s^t$ berechnet.
- Die Türme von Hanoi sind rekursiv zu programmieren. Man wird staunen, es sind nur ein paar Zeilen nötig.

Und, kommen die Erinnerungen wieder? Na dann können wir ja loslegen. Ohne Tupel und Listen ist Haskell nur der halbe Spaß!

- Schaue dir Tupel an an und schreibe fst, snd für Paare neu als `ourfst` und `oursnd`. Dann schreiben wir `triplefst, triplesnd und tripletrd` für Tripel (pattern matching).
- Schaue dir Listen an und schreibe deine eigenen Versionen von head und tail.
- Schreibe eine Funktion, die Tupel der Länge 3 in Listen konvertiert.

Und? Schon auf den Geschmack gekommen? Na, es sind ja jetzt zwei Wochen Zeit. Da braucht ihr noch mehr „Stoff“. Vorher aber noch eine kleine Erinnerung aus Klasse 9. Die schnelle Rekursion:
Beispiel:

```
1 fak 0 = 1
2 fak n = n * fak (n-1) -- So haben wir das frueher programmiert
3
4 fak2 n = fak' 1 n
5            where fak' acc 1 = acc
6                  fak' acc n = fak' (acc*n) (n-1)
```

Es sind weniger Speicherzugriffe auf dem Stack (LIFO) notwendig und die Ausführung der Operationen wird beschleunigt. Diese verkürzte Rekursion nennt man **Endrekursion**.
Aufgaben: Gesucht sind die folgende Funktionen (Signatur nicht vergessen!):

a) `malzwei` – multipliziert jedes Listenelement mit dem Faktor 2
b) `zwischensetzen` – setzt zwischen jedes Element der Liste ein gegebenes, Bsp: zwischensetzen 'a' ['c','d','f','t'] ⇝ ['c','a','d','a','f','a','t']
c) `erste` – bekommt einen Parameter m und gibt die ersten m Elemente, soweit vorhanden, aus Bsp: `erste` 3 ['c','d','f','t','a'] ⇝ ['c','d','f']
d) `letzten` – gibt die letzten m Element aus
e) `zaehlen` – zählt das Vorhandensein bestimmter Elemente in der Liste Bsp: `zaehlen` 'd' ['c','d','f','t','a','d'] ⇝ 2
f) `ohnevokal` – gibt einen übergebenen String ohne Vokale zurück
g) `ersetzen` – arbeitet wie ohnevokal, nur werden bestimmte Elemente durch andere ersetzt Bsp: `ersetzen` 3 8 [2,5,6,4,5,3,4,2,1,3,4,5,6,7,8] ⇝ [2,5,6,4,5,8,4,2,1,8,4,5,6,7,8]
h) `verbinden` – verknüpft zwei Listen zu ein, aber nach dem Reißverschlussprinzip Bsp: `verbinden` [2,5,6,4] [3,4,1,7] ⇝ [2,3,5,4,6,1,4,7]
i) Es ist eine Funktion `teilerliste` gesucht, die zu einer Zahl die Menge aller Teiler ermittelt.