PHILIPPS-UNIVERSITÄT MARBURG
Fachbereich Mathematik und Informatik
Prof. Dr. R. Loogen

D-35032 Marburg
Hans Meerwein Straße
Lahnberge

Übungen zur „Praktischen Informatik III“, WS 2005/06

Nr. 5, Abgabe: 29. November 2005 vor der Vorlesung

A. Hausaufgaben

Die Lösungen sollten grundsätzlich schriftlich, Programme zusätzlich auf Diskette oder per E-Mail an Ihre Tutorin abgegeben werden. Die Abgabe ist in Gruppen bis zu zwei Personen erlaubt.

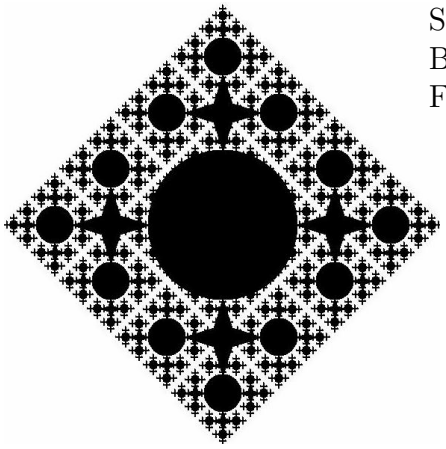
20. Interaktive Ein-/Ausgabe — 4 Punkte

Schreiben Sie ein interaktives Haskell-Programm, das solange eine Zahlenliste in Haskell-Syntax einliest und die Summe der Zahlen ausgibt, bis eine Leerzeile eingegeben wird.
Bei den Zahlenlisten soll es sich um endliche Aufzählungen von ganzen Zahlen handeln.
Nebenstehend ist eine Beispielausführung gezeigt.

```
Main> main
Bitte eine Zahlenliste eingeben!
[100,-34,200,300,-35,-100]
Die Summe beträgt: 431.
Bitte eine Zahlenliste eingeben!
[-123,300,400,-234,100,-234]
Die Summe beträgt: 209.
Bitte eine Zahlenliste eingeben!

Ende!
```

21. Fraktale Bilderzeugung — 5 Punkte



Schreiben Sie ein Programm, das ein fraktales Bild der angegebenen Form erzeugt und in einem Fenster ausgibt.

22. Testfunktion höherer Ordnung — 3 Punkte

Die Funktion `exists :: (a -> Bool) -> [a] -> Bool` soll bei einem Aufruf `(exists p xs)` überprüfen, ob mindestens ein Element der Liste `xs` das Prädikat `p` erfüllt. Zum Beispiel soll der Ausdruck `exists even [3,5,7,6]` den Wert `True` ergeben.

(a) Erstellen Sie eine herkömmliche rekursive Definition der Funktion. / 1

(b) Implementieren Sie die Funktion mittels `foldl` oder `foldr`. / 2

B. Mündliche Aufgaben

23. Anwendung von Funktionen höherer Ordnung

    Definieren Sie die folgenden Funktionen unter Verwendung vordefinierter Funktionen höherer Ordnung:

    (a) `pairAndSquare :: Num a => [a] -> [(a,a)]` paart jedes Element einer Liste von Zahlen (Typklasse `Num a`) mit seinem Quadrat.

    Beispiel: `pairAndSquare [1,2,3]` $\Rightarrow^*$ `[(1,1), (2,4), (3,9)]`

    (b) `multNeighbours :: Num a => [a] -> [a]` multipliziert je zwei aufeinanderfolgende Zahlen einer Liste.

    Beispiel: `multNeighbours [1,2,3,4]` $\Rightarrow^*$ `[2,6,12]`

    (c) `substitute :: Eq a => a -> a -> [a] -> [a]` ersetzt in einer Liste alle Vorkommen eines Elementes.

    Beispiel: `substitute 'u' 'i'  "substitute"` $\Rightarrow^*$ `"sibstitite"`

    (d) `remove :: Eq a => a -> [a] -> [a]` löscht in einer Liste alle Vorkommen eines Elementes.

    Beispiel: `remove 'u'  "substitute"` $\Rightarrow^*$ `"sbstitte"`

    (e) `numTwins :: Eq a => [a] -> Int` zählt in einer Liste, wie oft benachbarte Elemente gleich sind.

    Beispiel: `numTwins  "Hallo, Otto!"` $\Rightarrow^*$ `2`

    (f) `addPointwise :: Num a => [(a,a)] -> (a,a)` addiert komponentenweise die Elemente einer Liste von Zahlenpaaren.

    Beispiel: `addPointwise [(1,2), (2,3), (3,4)]` $\Rightarrow^*$ `(6,9)`