WiSe 2015/2016

Funktionale Programmierung

4. Übungsblatt (Abgabe: Mo., den 16.11. um 10:10 Uhr)

Prof. Dr. Margarita Esponda

Ziel: Auseinandersetzung mit Listengeneratoren und Funktionen höherer Ordnung.

1. Aufgabe (2 Punkte)

Was ist der Wert folgendes Ausdrucks? Schreiben Sie mindestens 3 Zwischenschritte auf, um Ihre Berechnungen zu begründen.

```
[(n,m) | n<-[1..3], m<-[3,2..0], n/=m]
```

2. Aufgabe (3 Punkte)

Schreiben Sie eine Haskell-Funktion **trueDivisor**, die bei Eingabe einer natürlichen positiven Zahl **n** die Liste aller echten Teiler von **n** berechnet und als Ergebnis der Funktion zurückgibt. Die Zahl **n** ist kein echter Teiler von **n** und soll deswegen nicht in die Ergebnisliste zurückgegeben werden. Verwenden Sie dafür Listengeneratoren.

Anwendungsbeispiel:

```
trueDivisor 250    =>   [1, 2, 5, 10, 25, 50, 125]
```

3. Aufgabe (3 Punkte)

Schreiben Sie eine Funktion **allPairMults**, die unter sinnvoller Verwendung von Listengeneratoren die Liste mit der Produkten aller Zweier-Zahlenkombinationen einer Eingabeliste berechnet. Die Elemente der Liste können mit sich selber kombiniert werden.

Anwendungsbeispiel:

```
allPairMults  [2, 3, 1]  => [4, 6, 2, 6, 9, 3, 2, 3, 1]
```

4. Aufgabe (6 Punkte)

Programmieren Sie eine Haskell-Funktion **allFriends**, die bei Eingabe einer natürlichen Zahl **n** alle befreundeten Zahlenpaare **(a, b)** berechnet, die kleiner **n** sind. D.h. mit **a<b<n**.

Zwei natürliche Zahlen **(m, n)** werden als "Befreundetes Zahlenpaar" bezeichnet, wenn jede Zahl gleich der Summe der echten Teiler der anderen Zahl ist. Definieren Sie zuerst eine Hilfsfunktion **friends**, die bei Eingabe zweier natürlicher Zahlen entscheidet, ob die Zahlen befreundet sind oder nicht.

Anwendungsbeispiel:

```
allFriends 300    =>   [(220,284)]
```

5. Aufgabe (6 Punkte)

Die Goldbachsche Vermutung sagt, dass jede gerade Zahl größer als 2 als Summe zweier Primzahlen geschrieben werden kann.

Definieren Sie eine Funktion **goldbachPairs**, die bei Eingabe einer geraden Zahl die Liste aller Goldbachschen Tupel ermittelt. Sie können in Ihrer Definition die **primzahlen**-Funktion aus den Vorlesungsfolien verwenden.

Anwendungsbeispiel:

```
goldbachPairs 80 => [(7,73), (13,67), (19,61), (37,43)]
```

6. Aufgabe (3 Punkte)

Definieren Sie eine **myAny** polymorphe Funktion, die entscheidet, ob mindestens ein Element einer Liste eine gegebene Bedingung erfüllt. Sie dürfen keine Listengeneratoren dafür verwenden.

Anwendungsbeispiel:

```
myAny (/=7) [2, 1, 7, 8, 3, 0] => True
```

7. Aufgabe (7 Punkte)

Definieren Sie eine polymorphe Funktion **allPositionsOf**, die die Positionen eines Elements innerhalb einer Liste wiederum in einer Liste zurückgibt.

Anwendungsbeispiel:

```
allPositionsOf 'n' "Funktionale Programmierung" => [2, 7, 24]
```

a) Definieren Sie zuerst die Funktion unter Verwendung von expliziter Rekursion und Akkumulator-Technik.

b) Definieren Sie die Funktion unter sinnvoller Verwendung von Listengeneratoren.

Wichtige Hinweise:

1) **Zur jeder Funktion sollen Testfunktionen geschrieben werden.**

2) Verwenden Sie geeignete Namen für Ihre Variablen und Funktionsnamen, die den semantischen Inhalt der Variablen oder die Semantik der Funktionen wiedergeben.

3) Verwenden Sie vorgegebene Funktionsnamen, falls diese angegeben werden.

4) Kommentieren Sie Ihre Programme.

5) Verwenden Sie geeignete lokale Funktionen und Hilfsfunktionen in Ihren Funktionsdefinitionen.

5) Schreiben Sie in allen Funktionen die entsprechende Signatur.