SoSe 2014
Prof. Dr. Margarita Esponda

ProInformatik II: Funktionale Programmierung

3. Übungsblatt (4. Tag)

1. Aufgabe

In der Vorlesung wurde die *randList*-Funktion besprochen (siehe Vorlesungsfolien), die in der Lage ist, bei Eingabe einer positiven Zahl n eine Liste mit n Pseudo-Zufallszahlen zu erzeugen. Schreiben Sie eine Funktion randUntilRepeat, die, mit Hilfe der gleichen random-Funktion aus der Vorlesung, bei Eingabe eines Startwertes (*seed*) die Liste aller Pseudo-Zufallszahlen, bis eine Wiederholung vorkommt, berechnet.

2. Aufgabe

Definieren Sie eine Haskell-Funktion, die bei Eingabe einer positiven Zahl **n** alle Primzahlen zwischen 1 und **n** berechnet, deren Quersumme wiederum eine Primzahl ist.

Anwendungsbeispiel:

querSumPrimes 100 => [2,3,5,7,11,23,29,41,43,47,61,67,83,89]

3. Aufgabe

Definieren Sie eine Funktion **groupEquals**, die aufeinanderfolgende gleiche Elemente in eine Liste gruppiert und als Ergebnis eine Liste von Listen zurückgibt.

Beispiel:

groupEquals [1,1,2,1,2,2,1,1,1] = [[1,1],[2],[1],[2,2],[1,1,1]]

4. Aufgabe

Definieren Sie eine eigene Funktion **myZip**, die immer der Reihe nach zwei Elemente aus den Listen nimmt und eine Liste von Tupeln konstruiert.

Beispiel:

myZip [1,2,3] [2,4,6] = [(1,2),(2,4),(3,6)]

5. Aufgabe

Definieren Sie Funktionen, die mit Listen von 8 Elementen (jedes Element ist 0 oder 1) arbeiten und folgende binäre Operationen simulieren:

a) die Summe von zwei 8-Bit-Registern (d.h. Das Ergebnis soll auch eine 8-Bit-Zahl sein).
b) die Subtraktion (Zweierkomplement) von zwei 8-Bit-Registern
c) das Produkt von zwei 8-Bit-Registern (16-Bit-Resultat)

6. Aufgabe

Definieren Sie ein Typ-Synonym **Menge** als [**Int**] und definieren Sie damit folgende Mengen-Operationen:

elementOf :: Int -> **Menge** -> Bool

gleich :: **Menge** -> **Menge** -> Bool

vereinigung :: **Menge** -> **Menge** -> **Menge**

teilmenge :: **Menge** -> **Menge** -> Bool

schnittmenge :: **Menge** -> **Menge** -> **Menge**

mengendifferenz :: **Menge** -> **Menge** -> **Menge**

Es wird angenommen, dass die Zahlen sortiert sind und keine Verdoppelung der Elemente vorhanden ist.

7. Aufgabe

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

a) Schreiben Sie eine Funktion **listOfSums**, die unter sinnvoller Verwendung von Listengeneratoren, die Liste mit der Summe aller zweier Zahlenkombinationen einer Eingabeliste berechnet. Die Elemente der Liste können mit sich selber kombiniert werden.

Anwendungsbeispiel: **listOfSums** [1, 2, 0] => [2, 3, 1, 3, 4, 2, 1, 2, 0]

b) Definieren Sie eine Funktion, 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** 32 => [(3, 29), (13, 19)]

8. Aufgabe

Wenn wir eine Menge ***M*** mit ***n*** verschiedenen Objekten haben, kann die Anzahl der verschiedenen ***k***-elementigen Teilmengen aus ***M*** mit Hilfe des bekannten Binomialkoeffizienten wie folgt berechnet werden.

Binomialkoeffizient (*n*, *k*) $= \begin{pmatrix} n \\ k \end{pmatrix} = \frac{n!}{k!\,(n-k)!}$ mit $0 \leq k \leq n$

a) Schreiben Sie eine Haskell-Funktion mit folgender Signatur

binom_naiv :: Integer -> Integer -> Integer

die mit Hilfe der vorherigen Definition und ohne Rekursion (außer innerhalb der Fakultätsfunktion) für beliebige natürliche Zahlen ***n*** und ***k*** den Binomialkoeffizienten berechnet.

Eine rekursive Definition der gleichen Funktion sieht wie folgt aus:

$$\binom{n}{k} = 0 \text{ , wenn } k > n$$

$$\binom{n}{0} = \binom{n}{n} = 1$$

$$\binom{n}{1} = \binom{n}{n-1} = n$$

$$\binom{n+1}{k+1} = \binom{n}{k} + \binom{n}{k+1} \quad \text{für alle } n,k \in \mathrm{N} \quad \text{mit } 0 < k \leq n-1$$

b) Definieren Sie eine rekursive Haskell-Funktion dafür.

c) Aus der ersten Definition von a) kann folgende Gleichung abgeleitet werden:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} = \begin{cases} 1 & falls \quad k = 0 \\ \frac{n}{1} \cdot \frac{(n-1)}{2} \cdot \frac{(n-2)}{3} \cdots \frac{(n-k+2)}{(k-1)} \cdot \frac{(n-k+1)}{k} & falls \quad k > 0 \end{cases}$$

Definieren Sie eine möglichst effiziente Haskell-Funktion, die diese Definition des Binomialkoeffizienten verwendet.

d) Schreiben Sie eine Test-Funktion, die überprüft, dass alle drei Funktionen das gleiche Ergebnis liefern.

Wichtige Hinweise:

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

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

3) Kommentieren Sie Ihre Programme.

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

5) Schreiben Sie in alle Funktionen die entsprechende Signatur.