1. Übungsaufgabe zu
Fortgeschrittene Funktionale Programmierung
Themen: Ströme, Generator/Selektor-Prinzip
ausgegeben: 16.03.2005, fällig: 13.04.2005

Für dieses Aufgabenblatt sollen Sie Haskell-Rechenvorschriften zur Lösung der im folgenden angegebenen Aufgabenstellungen entwickeln und für die Abgabe in einer Datei namens `AufgabeFFP1.hs` in Ihrem Homeverzeichnis ablegen, wie gewohnt auf oberstem Niveau. Kommentieren Sie Ihre Programme aussagekräftig und benutzen Sie, wo sinnvoll, Hilfsfunktionen und Konstanten.

- Schreiben Sie eine Haskell-Rechenvorschrift `facs :: [Int]`, die den Strom der "Fakultätszahlen" ausgibt. Der Aufruf von `facs` soll also die Ausgabe `[1,1,2,6,24,120,720,...` liefern.
- Schreiben Sie unter Benutzung der Rechenvorschrift `facs` aus dem vorigen Aufgabenteil und des Generator/Selektor-Prinzips Haskell-Rechenvorschriften
  - `fac :: Int -> Int`, die angewendet auf $n \geq 0$, $n \in \mathbb{N}$, die Ausgabe $n!$ liefert,
  - `facN :: Int -> [Int]`, die angewendet auf $n \geq 0$, $n \in \mathbb{N}$, die Liste der ersten n Fakultätszahlen liefert, beispielsweise soll der Aufruf `facN 4` die Ausgabe `[1,1,2,6]` liefern,
  - `facVB :: Int -> Int -> [Int]`, die angewendet auf $b \geq v \geq 0$, $b, v \in \mathbb{N}$, v erstes, b zweites Argument von `facVB` die Liste der v-ten bis b-ten Fakultätszahlen liefert, beispielsweise soll der Aufruf `facVB 2 4` die Ausgabe `[1,2,6]` liefern.
- Natürliche Zahlen, in deren Primfaktorzerlegung nur die Primfaktoren 2, 3 und 5 (null- oder auch mehrmals) vorkommen, heißen *Hammingzahlen*, d.h. die Menge der natürlichen Zahlen der Form $2^i * 3^j * 5^k$, $i, j, k \geq 0$. Schreiben Sie eine Haskell-Rechenvorschrift `hamming :: [Int]`, die den Strom der Hamming-Zahlen in aufsteigender Folge als Ausgabe liefert. Der Aufruf von `hamming` soll also die Ausgabe `[1,2,3,4,5,6,8,9,10,12,15,...` liefern.
- Schreiben Sie analog zu den Funktionen `facN` und `facVB` auch Funktionen `hamN` und `hamVB`.

- Schreiben Sie eine Haskell-Rechenvorschrift `partSum :: Int -> Int`, die angewendet auf Argumente n größer oder gleich 0 die Summe der ersten n natürlichen Zahlen liefert. Dabei sei 1 als kleinste natürliche Zahl vorausgesetzt. Der Aufruf `partSum 4` soll also die Ausgabe `10` liefern. Nutzen Sie zur Definition von `partSum` wieder das Generator/ Selektor-Prinzip aus.

- Der Wert der Exponentialfunktion exp an der Stelle x wird durch die Reihe
$$exp\ x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + ...$$
approximiert. Schreiben Sie eine Haskell-Rechenvorschrift `exps :: Float -> [Float]`, die den Strom der k-Präfixe der Approximation von exp angewendet auf x liefert. Die Ausgabe von `exps` soll also der Strom `[1,1 +` $\frac{x}{1!}$`,1 +` $\frac{x}{1!}$ `+` $\frac{x^2}{2!}$`,...` sein.

- Sehen Sie auch eine Funktion `expskPref :: Int -> Float -> Float` vor, die angewendet auf n, $n \in \mathbb{N}$, und x den n-Präfix von `exp x` liefert, für $n = 1$ also 1, für $n = 3$ also $1 + \frac{x}{1!} + \frac{x^2}{2!}$.