2. Übungsaufgabe zu
Fortgeschrittene funktionale Programmierung
Thema: Ströme, Memoization
ausgegeben: Di, 24.03.2015, fällig: Di, 14.04.2015

Für dieses Aufgabenblatt sollen Sie Haskell-Rechenvorschriften zur Lösung der im folgenden angegebenen Aufgabenstellungen entwickeln und für die Abgabe in einer Datei namens `AufgabeFFP2.hs` in Ihrem Gruppenverzeichnis ablegen, wie gewohnt auf oberstem Niveau. Kommentieren Sie Ihre Programme aussagekräftig und benutzen Sie, wo sinnvoll, Hilfsfunktionen und Konstanten.

- Sei $p \geq 2$ natürliche Zahl. Paare der Form $(p, p+2)$ heißen *Primzahlpaar* genau dann, wenn p und $p+2$ beides Primzahlen sind.

  Schreiben Sie eine 0-stellige Haskell-Rechenvorschrift `pps :: [(Integer,Integer)]`, die den Strom der Primzahlpaare generiert.

  Beispiele:

  ```
  take 10 pps ->> [(3,5),(5,7),(11,13),(17,19),(29,31),(41,43),
                   (59,61),(71,73),(101,103),(107,109)]
  pps!!20 ->> (347,349)
  head (drop 30 pps) ->> (809,811)
  ```

- Die Haskell-Rechenvorschrift

  ```
  pow :: Int -> Integer
  pow 0 = 1
  pow n = pow (n-1) + pow (n-1)
  ```

  berechnet die Funktion 2^n, $n \geq 0$.

  Implementieren Sie analog zum Beispiel zur Berechnung der Fibonacci-Zahlen aus der Vorlesung (vgl. Chapter 2, Memoization) eine Variante `powFast :: Int -> Integer` der Rechenvorschrift `pow`, die die Memoizationsidee aus dem Fibonacci-Beispiel aufgreift und die Berechnung auf eine Memo-Tafel abstützt:

  ```
  powFast :: Int -> Integer
  powFast 0 = ...
  powFast n = ...
  ...
  ```

  Vergleichen Sie (ohne Abgabe!) anhand von nach und nach größer gewählten Argumenten das unterschiedliche Laufzeitverhalten der Rechenvorschriften `pow` und `powFast`.

- Gegeben sei die Funktion g:

$$g(z,k) = 1 + z + \frac{z^2}{2!} + \frac{z^3}{3!} + \cdots = \sum_{n=0}^{k} \frac{z^n}{n!}$$

Implementieren Sie zwei Haskell-Rechenvorschriften

```
f :: Int -> Int -> Float
```

```
fMT :: Int -> Int -> Float
```

zur Berechnung der Funktion g, wobei die Implementierung von `fMT` eine (oder mehrere) Memo-Tafel(n) verwendet (z.B. eine Memo-Tafel für jeden z-Wert), die von `f` nicht. Beide Funktionen `f` und `fMT` mögen sich auf die Hilfsfunktion h abstützen

$$h(z,i) = \frac{z^i}{i!}$$

mit deren Hilfe sich g wie folgt ausdrücken lässt:

$$g(z,k) = \sum_{i=0}^{k} h(z,i)$$

Vergleichen Sie (ohne Abgabe!) auch hier wieder das Laufzeitverhalten der beiden Implementierungen miteinander.

- Sei $n \geq 1$ eine natürliche Zahl und sei $d_1 d_2 \ldots d_k$ die Folge der Dezimalziffern von n. Sei weiters $p_1, p_2, p_3, \ldots, p_k, \ldots$ die Folge der Primzahlen. Dann heißt die Zahl

$$2^{d_1} 3^{d_2} 5^{d_3} \ldots p_k^{d_k}$$

die *Gödelzahl* von n. Z.B. hat die Zahl 42 die Gödelzahl $144 = 2^4 * 3^2$, die Zahl 402 die Gödelzahl $400 = 2^4 * 3^0 * 5^2$.

1. Schreiben Sie eine Haskell-Rechenvorschrift `gz :: Integer -> Integer`, die positive Argumente auf ihre Gödelzahl abbildet, nicht-positive Argumente auf 0.
2. Schreiben Sie eine 0-stellige Haskell-Rechenvorschrift `gzs :: [Integer]`, die den Strom der Gödelzahlen beginnend für 1 liefert, d.h. das erste Listenelement ist die Gödelzahl von 1, das zweite Listenelement die Gödelzahl von 2, usw.