2. Übungsaufgabe zu
Fortgeschrittene funktionale Programmierung
Thema: Ströme, Memoization
ausgegeben: Mi, 16.03.2016, fällig: Mi, 06.04.2016

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.

- Schreiben Sie eine 0-stellige Haskell-Rechenvorschrift `ddps :: [Integer]`, die den Strom der natürlichen Zahlen liefert, bei denen die Zahl der Einsen ihrer Dualdarstellung eine Primzahl ist (Hinweis: 1 ist keine Primzahl!).

  Beispiele:

  ```
  take 10 ddps ->> [2,3,5,6,7,9,10,11,12,13]
  ddps!!20 ->> 26
  head (drop 15 ddps) ->> 21
  ```

- Implementieren Sie eine Variante `ddpsMT :: [Integer]` der Rechenvorschrift `ddps`, die sich für den Primzahltest auf eine Memotafel abstützt (vgl. Chapter 2, Memoization). Vergleichen Sie (ohne Abgabe!) anhand von nach und nach größer gewählten Argumenten das Laufzeitverhalten der Rechenvorschriften `ddps` und `ddpsMT` miteinander.

- Die Haskell-Rechenvorschrift

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

  berechnet die Funktion 3^n, $n \geq 0$.

  Implementieren Sie analog zum Beispiel zur Berechnung der Fibonacci-Zahlen aus der Vorlesung eine Variante `powMT :: Int -> Integer` der Rechenvorschrift `pow`, die die Memoizationsidee aus dem Fibonacci-Beispiel aufgreift und die Berechnung auf eine Memo-Tafel abstützt:

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

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

- Gegeben sei die Funktion g:

$$g(x,k) = x + \frac{x^3}{3!} + \frac{x^5}{5!} + \cdots = \sum_{n=0}^{k} \frac{x^{2n+1}}{(2n+1)!}$$

  Implementieren Sie zwei Haskell-Rechenvorschriften

```
f :: Integer -> Integer -> Double

fMT :: Integer -> Integer -> Double
```

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

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

  mit deren Hilfe sich g wie folgt ausdrücken lässt:

$$g(x,k) = \sum_{i=0}^{k} h(x, 2i+1)$$

  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.
  3. Schreiben Sie eine Variante `gzMT` der Rechenvorschrift `gz`, die die Beschleunigung der Berechnung der Gödelzahlen durch Verwendung einer (oder mehrerer) Memo-Tafel(n) erreicht (z.B. für die Werte von $p_i^{d_j}$).
  4. Schreiben Sie eine Variante `gzsMT` von `gzs`, die sich auf `gzMT` statt `gz` abstützt. Vergleichen Sie auch hier wieder (ohne Abgabe!) die verschiedenen Laufzeiten.