3. Übungsaufgabe zu
Fortgeschrittene funktionale Programmierung
Thema: Generator/Transformierer/Filter/Selektor-Prinzip;
Ströme, Memoization
ausgegeben: Mi, 18.04.2012, fällig: Mi, 25.04.2012

Für dieses Aufgabenblatt sollen Sie Haskell-Rechenvorschriften zur Lösung der im folgenden angegebenen Aufgabenstellungen entwickeln und für die Abgabe in einer Datei namens `AufgabeFFP3.hs` in Ihrem Gruppenverzeichnis ablegen, wie gewohnt auf oberstem Niveau. Kommentieren Sie Ihre Programme aussagekräftig und benutzen Sie, wo sinnvoll, Hilfsfunktionen und Konstanten.

- Wir betrachten folgende einfache Variante eines Knapsack- oder Rucksackproblems.

  Gegeben ist eine endliche Menge von Gegenständen, die durch ihr Gewicht und ihren Wert gekennzeichnet sind. Aufgabe ist es, den Rucksack unter verschiedenen Randbedingungen bestmöglich zu bepacken, z.B. so, dass die Summe der Werte der eingepackten Gegenstände maximal ist, ohne ein vorgegebenes Höchstgewicht zu überschreiten.

  Dazu sollen vier Funktionen `generator`, `transformer`, `filter`, und `selector` geschrieben werden, deren Komposition

  ```
  selector . filter . transformer . generator
  ```

  die gestellte Aufgabe erfüllt.

  Wir verwenden folgende Typen und Deklarationen zur Modellierung des Rucksackproblems:

  ```
  type Weight      = Int                -- Gewicht
  type Value       = Int                -- Wert
  type Item        = (Weight,Value)     -- Gegenstand als Gewichts-/Wertpaar
  type Items       = [Item]             -- Menge der anfaenglich gegebenen
                                           Gegenstaende
  type Load        = [Item]             -- Auswahl aus der Menge der anfaenglich
                                           gegebenen Gegenstaende; moegliche
                                           Rucksackbeladung, falls zulaessig
  type Loads       = [Load]             -- Menge moeglicher Auswahlen
  type LoadWghtVal = (Load,Weight,Value) -- Eine moegliche Auswahl mit
                                           Gesamtgewicht/-wert dieser Auswahl
  type MaxWeight   = Weight             -- Hoechstzulaessiges Rucksackgewicht

  generator   :: Items -> Loads
  transformer :: Loads -> [LoadWghtVal]
  filter      :: [LoadWghtVal] -> MaxWeight -> [LoadWghtVal]
  selector    :: [LoadWghtVal] -> [LoadWghtVal]
  ```

Die einzelnen Funktionen leisten dabei folgendes:

- `generator`: baut aus der gegebenen Menge von Gegenständen die Menge aller möglichen Auswahlen auf (ohne auf Zulässigkeit zu achten).
- `transformer`: ergänzt die möglichen Auswahlen jeweils um deren Gesamtgewicht und -wert
- `filter`: streicht alle Auswahlen, die nicht zulässig sind, z.B. das zulässige Höchstgewicht einer Auswahl übersteigen.
- `selector`: wählt aus der Menge der zulässigen Auswahlen alle diejenigen aus, die bezüglich des vorgegebenen Optimalitätsziels am besten sind. Dies kann eine, mehrere oder keine Auswahl sein, wenn z.B. keine Auswahl zulässig ist.

1. Implementieren Sie die Funktionen `generator`, `transformer`, `filter` zusammen mit einer Funktion `selector1` derart, dass `selector1` die oder diejenigen Auswahlen mit höchstem Wert liefert:

```
(selector1 . filter . transformer . generator) [(5,3),(2,7),(2,6),(10,100)] 5
            ->> [([(2,7),(2,6)],4,13)]
(selector1 . filter . transformer . generator) [(5,3),(2,7),(2,6),(10,100)] 13
            ->> [([(2,7),(10,100)],12,107)]
(selector1 . filter . transformer . generator) [(5,3),(2,7),(2,6),(10,100)] 1
            ->> []
(selector1 . filter . transformer . generator) [(5,13),(2,7),(2,6),(10,100)] 5
            ->> [([(2,7),(2,6)],4,13),([(5,13)],5,13)]
```

2. Implementieren Sie zusätzlich eine Funktion `selector2`, die bei gleichem Gesamtwert die oder diejenigen Auswahlen herausgreift, die diesen Wert mit geringstem Gesamtgewicht erreichen.

```
(selector2 . filter . transformer . generator) [(5,13),(2,7),(2,6),(10,100)] 5
            ->> [([(2,7),(2,6)],4,13)]
```

Hinweis: Die Reihenfolgen der Elemente in den Ergebnislisten spielt keine Rolle.

- Für Binomialkoeffizienten gilt folgende Beziehung:

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

Daraus lässt sich folgende Implementierung zur Berechnung der Binomialkoeffizienten ableiten:

```
binom :: (Integer,Integer) -> Integer
binom (n,k)
  | k==0 || n==k = 1
  | otherwise    = binom (n-1,k-1) + binom (n-1,k)
```

Schreiben Sie nach dem Vorbild aus Kapitel 2.4 der Vorlesung zwei effizientere Varianten zur Berechnung der Binomialkoeffizienten mithilfe von

1. Stromprogrammierung: `binomS :: (Integer,Integer) -> Integer`
2. Memoization: `binomM :: (Integer,Integer) -> Integer`

Vergleichen Sie (ohne Abgabe!) das Laufzeitverhalten der drei Implementierungen `binom`, `binomS` und `binomM` miteinander.