1. Übungsaufgabe zu
Fortgeschrittene funktionale Programmierung
Thema: Generator/Transformierer/Filter/Selektor-Prinzip;
Ströme, Memoization
Ausgegeben: Mi, 14.03.2018, abzugeben: Mi, 21.03.2018 (15:00 Uhr)

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` auf oberstem Niveau in Ihrem Gruppenverzeichnis ablegen. Kommentieren Sie Ihre Programme aussagekräftig und benutzen Sie, wo sinnvoll, Hilfsfunktionen und Konstanten.

- Wir betrachten folgende einfache Variante eines 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 dass ein vorgegebenes Höchstgewicht überschritten wird.

  Schreiben Sie für diese Aufgabe Funktionen `rs_generator`, `rs_transformer`, `rs_filter` und `rs_selector1/2`, deren Komposition

  ```
  'rs_selector1/2 . rs_filter . rs_transformer . rs_generator'
  ```

  die Aufgabe löst (`rs` erinnere dabei an Rucksack). Verwenden Sie dabei zur Modellierung des Rucksackproblems folgende Typen und Deklarationen :

  ```
  type Weight      = Int                -- Gewicht
  type Value       = Int                -- Wert
  type Item        = (Weight,Value)     -- Gegenstand als Gewichts/Wert-Paar
  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 der Auswahl
  type MaxWeight   = Weight             -- Hoechstzulaessiges Rucksackgewicht

  rs_generator   :: Items -> Loads
  rs_transformer :: Loads -> [LoadWghtVal]
  rs_filter      :: MaxWeight -> [LoadWghtVal] -> [LoadWghtVal]
  rs_selector1   :: [LoadWghtVal] -> [LoadWghtVal]
  rs_selector2   :: [LoadWghtVal] -> [LoadWghtVal]
  ```

Die einzelnen Funktionen leisten dabei folgendes:

- `rs_generator`: baut aus der gegebenen Menge von Gegenständen die Menge aller möglichen Auswahlen auf (ohne auf Zulässigkeit zu achten).
- `rs_transformer`: ergänzt die möglichen Auswahlen jeweils um deren Gesamtgewicht und -wert.
- `rs_filter`: streicht alle Auswahlen, die nicht zulässig sind, z.B. das zulässige Höchstgewicht einer Auswahl übersteigen.
- `rs_selector1/2`: wählen 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 `rs_generator`, `rs_transformer`, `rs_filter` zusammen mit einer Funktion `rs_selector1` derart, dass `rs_selector1` die oder diejenigen Auswahlen mit höchstem Gesamtwert liefert:

```
(rs_selector1 . (rs_filter 5) . rs_transformer . rs_generator) [(5,3),(2,7),(2,6),(10,100)]
            ->> [([(2,7),(2,6)],4,13)]
(rs_selector1 . (rs_filter 13) . rs_transformer . rs_generator) [(5,3),(2,7),(2,6),(10,100)]
            ->> [([(2,7),(10,100)],12,107)]
(rs_selector1 . (filter 1) . rs_transformer . rs_generator) [(5,3),(2,7),(2,6),(10,100)]
            ->> []
(rs_selector1 . (rs_filter 5) . rs_transformer . rs_generator) [(5,13),(2,7),(2,6),(10,100)]
            ->> [([(2,7),(2,6)],4,13),([(5,13)],5,13)]
```

2. Implementieren Sie zusätzlich eine Funktion `rs_selector2`, die bei gleichem Gesamtwert die oder diejenigen Auswahlen herausgreift, die diesen Wert mit geringstem Gesamtgewicht erreichen.

```
(rs_selector2 . (rs_filter 5) . rs_transformer . rs_generator) [(5,13),(2,7),(2,6),(10,100)]
            ->> [([(2,7),(2,6)],4,13)]
```

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

- Für Binomialkoeffizienten gilt folgende Beziehung ($0 \leq k \leq n$):

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

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: `binom_s :: (Integer,Integer) -> Integer`
2. Memoization: `binom_m :: (Integer,Integer) -> Integer`

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