3. Übungsaufgabe zu
Fortgeschrittene funktionale Programmierung
Thema: Generator/Filter/Selektor/Transformer-Prinzip;
Stromprogrammierung und Memoization
ausgegeben: Mi, 06.04.2016, fällig: Mi, 13.04.2016

Für dieses Aufgabenblatt sollen Sie Haskell-Rechenvorschriften zur Lösung der im folgenden angegebenen Aufgabenstellungen entwickeln und für die Abgabe in einer Datei namens `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 folgendes Springerproblem aus dem Schachspiel: Kann ein Springer in genau n Zügen von einem vorgegebenen Start- zu einem vorgegebenen Zielfeld gelangen?

  Wir wollen diese Aufgabe mit dem Generator-/Filter-/Selektor-/Transformer-Prinzip lösen und betrachten dazu folgende Datentypen

```
data Reihe = A | B | C | D | E | F | G | H
                                        deriving (Eq,Ord,Enum,Show)
data Linie = Eins | Zwei | Drei | Vier | Fuenf
           | Sechs | Sieben | Acht  deriving (Eq,Ord,Enum,Show)
type StartPosition = (Reihe,Linie)
type ZielPosition  = (Reihe,Linie)
type AnzahlZuege   = Int
type VonPosition   = (Reihe,Linie)
type NachPosition  = (Reihe,Linie)
type Zug           = (VonPosition,NachPosition)
type Zugfolge      = [Zug]
```

  Damit stellt sich ein Schachbrett wie folgt dar:

```
         A   B  C  D  E  F  G  H
Acht                              Acht
Sieben                            Sieben
Sechs                             Sechs
Fuenf                             Fuenf
Vier                              Vier
Drei                              Drei
Zwei                              Zwei
Eins                              Eins
         A   B  C  D  E  F  G  H
```

Gesucht ist nun eine Funktion

```
springer :: StartPosition -> AnzahlZuege -> ZielPosition -> [Zugfolge]
springer =
  spgTransformer . spgSelektor . spgFilter . spgGenerator . transform
```

die alle Zugfolgen bestimmt, die das Gewünschte leisten. Dazu operieren die Funktionen `spgGenerator`, `spgFilter`, `spgSelektor` und `spgTransformer` auf einem virtuellen Schachbrett nicht begrenzter Größe, wobei die Position A-Eins durch das Zahlenpaar $(1, 1)$ vom Typ `(Integer,Integer)` bis hin zur Position H-Acht durch das Zahlenpaar $(8, 8)$ vom Typ `(Integer,Integer)` repräsentiert wird. Auf diesem virtuellen Schachbrett kann ein Springer auch Züge vollführen, die ihn aus dem üblichen 8×8-Schachbrett herausführen und nicht zu einer zulässigen Zugfolge gehören.

Im einzelnen leisten die Funktionen also folgendes:

- `transform`: wandelt Start- und Zielposition in ihre Darstellungen auf dem virtuellen Schachbrett um und reicht diese zusammen mit dem vorgegebenen Wert für die Anzahl der Züge an die Funktion `spgGenerator` weiter.
- `spgGenerator`: baut ausgehend von der Startposition alle Zugfolgen der vorgegebenen Länge auf dem virtuellen Schachbrett auf. Positionen sind dabei Integer-Paare, Züge sind Paare von Integer-Paaren und Zugfolgen sind Listen derart dargestellter Züge.
- `spgFilter`: streicht alle Zugfolgen, die nicht zulässig sind, weil mindestens ein Zug der Zugfolge aus dem 8×8-Schachbrett herausgeführt hat.
- `spgSelektor`: wählt aus der Menge der zulässigen Zugfolgen diejenigen aus, die an der gewünschten Zielposition enden. Dies können eine, mehrere oder auch keine Zugfolge sein.
- `spgTransformer`: transformiert die Zugfolgen auf dem virtuellen Schachbrett in Zugfolgen auf dem wirklichen Schachbrett, wo Positionen durch Reihen- und Linienwerte der Typen `Reihe` und `Linie` gegeben sind.

Hinweis: Sie können davon ausgehen, dass die Funktion `springer` ausschließlich mit positiven Werten für das Argument `AnzahlZuege` aufgerufen wird, d.h. mit Werten größer oder gleich 0.

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

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