6. Aufgabenblatt zu Funktionale Programmierung vom 21.11.2012.
Fällig: 28.11.2012 / 05.12.2012 (jeweils 15:00 Uhr)

Themen: *Typklassen, Polymorphie und Überladung*

Für dieses Aufgabenblatt sollen Sie Haskell-Rechenvorschriften für die Lösung der unten angegebenen Aufgabenstellungen entwickeln und für die Abgabe in einer Datei namens `Aufgabe6.hs` ablegen. Sie sollen für die Lösung dieses Aufgabenblatts also ein "gewöhnliches" Haskell-Skript schreiben. Versehen Sie wieder wie auf den bisherigen Aufgabenblättern alle Funktionen, die Sie zur Lösung brauchen, mit ihren Typdeklarationen und kommentieren Sie Ihre Programme aussagekräftig. Benutzen Sie, wo sinnvoll, Hilfsfunktionen und Konstanten.

Im einzelnen sollen Sie die im folgenden beschriebenen Problemstellungen bearbeiten.

1. Wir betrachten den Datentyp `Nat` von Aufgabenblatt 4:

   ```
   data Nat = Z | S Nat
   ```

   Machen Sie den Datentyp `Nat` zu einer Instanz der Typklasse `Show`. Werte des Datentyps `Nat` sollen dabei in Form römisch-artiger Zahlen ausgegeben werden. Dabei werden folgende Zahlzeichen verwendet: 'I' für 1, 'V' für 5, 'X' für 10, 'L' für 50, 'C' für 100, 'D' für 500 und 'M' für 1000. Der `Nat`-Wert `Z` soll durch die leere Zeichenreihe dargestellt werden. Grundsätzlich gilt: Die Werte gleicher Zahlzeichen hintereinander werden addiert (z.B. "III" für 3, "CC" für 200, "MMM" für 3000). Der Wert eines kleineren Zahlzeichens vor einem größeren wird von diesem subtrahiert (zB. ."IV" für 4, "IX" für 9, "XL" für 40, "IL" für 49, "IC" für 99). Darüberhinaus verabreden wir für die Zwecke dieser Aufgabe weiters, dass

   (1) mit Ausnahme von 'M' kein Zahlzeichen mehr als dreimal unmittelbar hintereinander verwendet werden darf (z.B. "CD" und nicht "CCCC" für 400);

   (2) kein Zahlzeichen mehr als einmal hintereinander verwendet werden darf, wenn diese sich zum Wert des nächstgrößeren Zahlzeichens addieren (z.B. "C" und nicht "LL" für 100);

   (3) Subtraktionen, so nötig, stets so weit rechts wie möglich durchgeführt werden (z.B. "CCIC" und nicht "ICCC" oder "CICC" für 299);

   (4) höchstens ein kleineres Zahlzeichen vor einem größeren steht (z.B. "XCVIII" und nicht "IIC" für 98)

   (5) die Zahlzeichen 'V', 'L' und 'D' höchstens einmal und nie vor einem größeren Zahlzeichen stehen dürfen (z.B. "XCV" und nicht "VC" für 95, "XIX" und nicht "XVIV" für 19).

   Mit diesen Vereinbarungen gilt: 8 wird dargestellt als "VIII", 9 als "IX", 19 als "XIX", 39 als "XXXIX", 84 als "LXXXIV", 89 als "LXXXIX", 98 als "XCVIII", 99 als "IC", 889 als "DCCCLXXXIX", 1899 durch "MDCCCIC", 5884 durch "MMMMMDCCCLXXXIV", 5999 durch "MMMMMIM". Die Darstellungen "IXX" für 19, "IXL" für 39, "IXC" für 89, "VCIII" für 98, "DCCCIXC" für 889 oder "IMMMMMM" für 5999 werden nach obigen Vereinbarungen für diese Aufgabe als nicht zulässig angesehen.

   Folgende Beispiele zeigen das gewünschte Ein-/Ausgabeverhalten:

   ```
   n1 = Z
   n2 = S (S Z)
   n3 = S (S (S (S Z)))
   n4 = S (S (S (S (S (S Z)))))
   n5 = S (S (S (S (S (S (S Z))))))
   n6 = S (S (S (S (S (S (S (S (S (S Z))))))))))
   show n1 ->> ""
   show n2 ->> "II"
   show n3 ->> "IV"
   show n4 ->> "VI"
   show n5 ->> "VII"
   show n6 ->> "XI"
   ```

Wichtiger Hinweis: Die obigen Festlegungen weichen von den üblichen Darstellungen römischer Zahlen ab! Sie heißen deshalb hier auch *römisch-artige* Zahlen und nicht römische Zahlen. Insbesondere sind übliche Konvertierungsroutinen zur Umwandlung in römische Zahlen deshalb hier nicht eins-zu-eins anwendbar.

2. Wir betrachten den Datentyp `RatNumbers`

```
data RatNumbers = Rat Numerator Denominator
type Numerator = Nat
type Denominator = Nat
```

für die Darstellung positiver rationaler Zahlen. Zwei `RatNumbers`-Werte `Rat m n` und `Rat p q` sind *wertegleich*, wenn die Beziehung `mq` = `np` erfüllt ist. Ein `RatNumbers`-Wert `Rat m n` ungleich 0 liegt in *Normalform* vor, wenn `m` und `n` teilerfremd sind. Die Zahl 0 hat als Normalform den `RatNumbers`-Wert `Rat Z (S Z)`.

- Machen Sie den Typ `RatNumbers` zu einer Instanz der Typklasse `Show`. Dabei sollen `RatNumbers`-Werte als (-2)-adische Zahlen getrennt durch einen Schrägstrich ausgegeben werden.

  Folgende Beispiele zeigen das gewünschte Ein-/Ausgabeverhalten:

  ```
  show (Rat Z (S Z)) ->> "0/1"
  show (Rat (S (S (S Z))) (S (S (S (S Z))))) ->> "111/100"
  show (Rat (S (S (S Z))) (S (S (S (S (S (S Z)))))))) ->> "111/11010"
  ```

- Wir führen den Typ

  ```
  newtype NatP = NP (Nat,Nat) deriving Show
  ```

  zur Darstellung ganzer (!) Zahlen in Analogie zum Datentyp `NatPair` von Aufgabenblatt 4 und die Typklasse `Nf` für Typen mit Normalform ein:

  ```
  class Nf a where
    t2nf :: a -> a
  ```

  wobei die Funktion `t2nf` ("transformiere in Normalform") ihr Argument in Normalform überführt.

  Machen Sie die Typen `RatNumbers` und `NatP` zu einer Instanz der Typklasse `Nf`. Die Normalform für Werte vom Typ `NatP` ist dabei analog zur kanonischen Darstellung von `NatPair`-Werten von Aufgabenblatt 4 gegeben.

  Folgende Beispiele zeigen das gewünschte Ein-/Ausgabeverhalten:

  ```
  t2nf (Rat Z (S (S (S Z)))) ->> Rat Z (S Z)
  t2nf (Rat (S (S (S Z))) (S (S (S (S Z))))) ->> Rat (S (S (S Z))) (S (S (S (S Z))))
  t2nf (Rat (S (S (S Z))) (S (S (S (S (S (S Z))))))) ->> Rat (S Z) (S (S Z))
  t2nf (NP (S Z,S Z)) ->> NP (Z,Z)
  t2nf (NP ((S (S Z)),(S (S (S Z))))) ->> NP (Z,S Z)
  t2nf (NP ((S (S (S Z))),(S (S Z)))) ->> NP (S Z,Z)
  ```

- Machen Sie ohne Verwendung der `deriving`-Klausel den Typ `RatNumbers` zu einer Instanz der Typklassen `Eq`, `Ord` und `Num`. Geben Sie für alle Funktionen dieser Typklassen eine Implementierung für `RatNumbers` an, die der natürlichen Bedeutung dieser Funktionen auf den Werten vom Typ `RatNumbers` entspricht, wobei Ergebnisse von Operationen vom Typ `RatNumbers` stets in Normalform sein sollen. Machen Sie von den Standardimplementierungen in den Klassen Gebrauch, wo möglich. Die Relatoren (gleich, ungleich, kleiner, größer, etc.) in den Typklassen vergleichen dabei Werte, nicht Strukturen. So liefert etwa der Gleichheitstest für `Rat Z (S Z)` und `Rat Z (S (S Z))` den Wert `True`. Die Funktion `fromInteger` der Klasse `Num` soll den entsprechenden `RatNumbers`-Wert des Absolutbetrags des Arguments in Normalform liefern.

Haskell Live

Der nächste *Haskell Live*-Termin ist am Freitag, den 23.11.2012. An diesem Termin haben Sie auch die Möglichkeit, Ihre Lösungsvorschläge für Aufgabenblatt 3 und 4 vorzustellen.