2. Aufgabenblatt vom Dienstag, den 21. Juli 2009 zur Vorlesung

Proinformatik

Marco Block

Dienstag, den 21. Juli 2009

1. **Integral** Machen Sie den Typen `Nat` aus der Vorlesung zu einer Instanz der Typklasse **Integral**

```
class (Real a, Enum a) => Integral a where
  quot, rem, div, mod :: a -> a -> a
  quotRem, divMod  :: a -> a -> (a,a)
  toInteger       :: a -> Integer
  toInt          :: a -> Int

  -- Minimal complete definition: quotRem and toInteger
  n `quot` d        = q where (q,r) = quotRem n d
  n `rem` d      = r where (q,r) = quotRem n d
  n `div` d      = q where (q,r) = divMod n d
  n `mod` d      = r where (q,r) = divMod n d
  divMod n d        = if signum r == - signum d then
    (q-1, r+d) else qr
         where qr@(q,r) = quotRem n d
  toInt         = toInt . toInteger

-- Mit diesen Klassen als Vorbedingung
class (Num a, Ord a) => Real a where
  toRational    :: a -> Rational
  -- Benutzen Sie hier toRational = toRational . fromEnum
  -- Warum funktioniert das?


class Enum a where
  succ, pred   :: a -> a
  toEnum   :: Int -> a
  fromEnum :: a -> Int
  enumFrom :: a -> [a]          -- [n..]
  enumFromThen :: a -> a -> [a]     -- [n,m..]
  enumFromTo :: a -> a -> [a]     -- [n..m]
  enumFromThenTo :: a -> a -> a -> [a]  -- [n,n'..m]

  -- Minimal complete definition: toEnum, fromEnum
  succ = toEnum . (1+) . fromEnum
  pred = toEnum . subtract 1 . fromEnum
  enumFrom x = map toEnum [fromEnum x ..]
  enumFromTo x y = map toEnum [fromEnum x .. fromEnum y]
  enumFromThen x y = map toEnum [fromEnum x, fromEnum y ..]
  enumFromThenTo x y z  = map toEnum [fromEnum x, fromEnum y ..
     fromEnum z]
```

2. **Listen** Definieren Sie folgende Methoden für Listen und Überlegen Sie sich möglichst allgemeine Typen:

   - `drehUm`, dreht eine Liste um
   - `istPalindrom`, testet, ob eine Liste ein Palindrom ist
   - `kleinstes`, gibt das kleinste Element einer Liste zurück
   - `fuegEin`, nimmt eine sortierte Liste und ein Element und fügt es so ein, dass die Ergebnisliste wieder sortiert ist

     ```
     fuegEin [1,3,5] 4 = [1,3,4,5]
     ```

   - `selectSort`, nimmt wiederholt das Minimum einer Liste und fügt es an eine andere Liste an, um eine sortierte Liste zu bauen
   - `insertSort`, nimmt nacheinander die Elemente einer Liste und fügt sie mit `fuegEin` in eine Ergebnisliste ein. Liefert eine sortierte Liste als Ergebnis