Assignment I (25 pts)

Burak Ekici

Assigned : October the 13th, 23h55
Due : October the 20th, 23h55

Q1 (12 pts). It is possible to develop the type Pos of positive integers ($\mathbb{Z}^+$) in Haskell as

```
data Pos where
    XI :: Pos -> Pos
    XO :: Pos -> Pos
    XH :: Pos
```

where type instances are encoded in binary terms. That is, starting with the data constructor XH (encoding 1), one can insert new least significant digits employing data constructors XO (digit 0) and XI (digit 1), and build the type of positive integers entirely. E.g., the number $6 \in \mathbb{Z}^+$ is encoded to be XO (XI XH) while XI (XO (XI XH)) refers to $13 \in \mathbb{Z}^+$. Similarly, XO (XO (XO XH)) implements the number $8 \in \mathbb{Z}^+$ Notice that digits of a given Pos instance are read from right to left.

a) **(3 pts)** define the Pos type as a Show class instance.

   Hint: develop a function pos2Int :: Pos -> Int and then employ it when it comes to the instance construction:

```
instance Show Pos where
    show p = show (pos2Int p)
```

   implementing a function int2Pos :: Int -> Pos would facilitate your work by quite some margin.

b) **(2 pts)** define the Pos type as an Eq class instance.

   Hint: it suffices to implement a Boolean valued function posEq :: Pos -> Pos -> Bool and then plug it in as follows:

```
instance Eq Pos where
    p == q = posEq p q
```

c) **(2 pts)** define the Pos type as an Ord class instance.

   Hint: in a similar fashion with the equality testing function above, just implement a Boolean valued function posLeq :: Pos -> Pos -> Bool and benefit from it as cleared below:

```
instance Ord Pos where
    p <= q = posLeq p q
```

d) **(5 pts)** define the Pos type as a Num class instance.

   Hint: first implement below listed functions

```haskell
posAdd         :: Pos -> Pos -> Pos -- addition
posMult        :: Pos -> Pos -> Pos -- multiplication
posSubtr       :: Pos -> Pos -> Pos -- subtraction
posSignum      :: Pos -> Pos        -- sign calculation
posAbs         :: Pos -> Pos        -- absolute value calculation
posFromInteger :: Integer -> Pos    -- conversion from Integer
```

and make use of them as follows:

```haskell
instance Num Pos where
    n + m         = posAdd n m
    n * m         = posMult n m
    abs n         = posAbs n
    signum n      = posSignum n
    fromInteger n = posFromInteger n
    n - m         = posSubtr n m
```

Q2 (13 pts). Below given is a way to construct the type `Rat` of rational numbers ($\mathbb{Q}$) in Haskell

```haskell
data Rat where
    Frac :: Int -> Pos -> Rat
```

in which a single constructor named `Frac` formalizing fractions is employed. Observe that the first argument of `Frac` is an integer encoding the numerator while the second argument is a positive integer implementing the denominator of an arbitrarily given fraction. E.g., `Frac (-2) (XI (XO (XI XH)))` connotes the number $\frac{-2}{13} \in \mathbb{Q}$ while the number $\frac{5}{2} \in \mathbb{Q}$ is represented by `Frac 5 (XO XH)`.

a) **(2 pts)** define the `Rat` type as a `Show` class instance.

b) **(2 pts)** define the `Rat` type as an `Eq` class instance.

c) **(2 pts)** define the `Rat` type as an `Ord` class instance.

d) **(7 pts)** define the `Rat` type as a `Num` class instance.

Nota Bene (in general).

1. receive support from helper functions if needed;
2. the attached file (`Rationals.hs`) could be a good starting point;
3. do not remove or modify the line `{-#LANGUAGE GADTs#-}` contained in the file `Rationals.hs`.

Important Notice:

- Collaboration is strictly and positively prohibited; lowers your score to 0 if detected.
- Any submission after 23h55 on October the 20th will NOT be accepted. Please beware and respect the deadline!
- Implement your code within a file named `yourname_surname.hs`, and submit it either in the raw form as it is or in the ZIP compressed form. Do not RAR files.