Ficha 10

Programaçao Funcional

2015/16

```
module FichaTP10 where

import FichaTP9
import FichaTP7 --é precciso tirar os
"deriving Show"
```

1. Considere o seguinte tipo de dados para representar fraccões

```
data Frac = F Integer Integer
```

   (a) Defina a funcão `normaliza :: Frac -> Frac`, que dada uma fraccão calcula uma fracçao equivalente, irredutível, e com o denominador positivo.
   Por exemplo: `normaliza (F (-33) (-51)) = (F 11 17)`
   `normaliza (F 50 (-5)) = (F (-10) 1)`
   Sugestao: defina primeiro a funcão `mdc :: Integer -> Integer -> Integer` que calcula o maximo divisor comum entre dois números baseada na seguinte propriedade (atribuida a Euclides): `mdc x y = mdc (x + y) y = mdc x (y + x)`

```
mdc :: Integer -> Integer -> Integer
mdc a b | a > b = mdc (a - b) b
        | a < b = mdc a (b - a)
        | a == b = a


normaliza :: Frac -> Frac
normaliza (F n 0) = error "denominador nulo"
normaliza (F 0 d) = F 0 1
normaliza (F n d) = F ((signum d) * (n `div` m)) ((abs d) `div` m) where m = mdc (abs n) (abs d)
```

   (b) Defina `Frac` como instancia da classe `Eq`.

```
instance Eq Frac where -- ver "slides 06" para
a definição da classe.
(==) x y = (a1 == a2) && (b1 == b2) where
    (F a1 b1) = normaliza x
    (F a2 b2) = normaliza y
```

   (c) Defina `Frac` como instancia da classe `Ord`.

```
instance Ord Frac where -- ver "slides 06" para
a definição da classe.
compare (F n1 d1) (F n2 d2) = compare (n1 *
d2) (d1 * n2)
```

   (d) Defina `Frac` como instancia da classe `Show`, de forma a que cada fracção seja apresentada por (numerador/denominador).

```
instance Show Frac where -- ver "slides 06" para a definição da classe.
    show (F num den) = "(" ++ show num ++ "/" ++ show den ++ ")"
```

(e) Defina Frac como instancia da classe Num. Relembre que a classe Num tem a seguinte definicao

```
class (Eq a, Show a) => Num a where
  (+), (*), (-) :: a -> a -> a
  negate, abs, signum :: a -> a
  fromInteger :: Integer -> a
```

```
instance Num Frac where
    (+) (F n1 d1) (F n2 d2) = normaliza (F ((n1 * d2) + (n2 * d1)) (d1 * d2))
    (*) (F n1 d1) (F n2 d2) = normaliza (F (n1 * n2) (d1 * d2))
    (-) (F n1 d1) (F n2 d2) = normaliza (F ((n1 * d2) - (n2 * d1)) (d1 * d2))
    abs (F n d) = (F (abs n) (abs d))
    signum (F n d) = let (F a b) = normaliza (F n d) in if (a == 0) then 0 else if (a > 0)
    then 1 else (-1)
    fromInteger x = (F x 1)
```

(f) Defina uma funcao que, dada uma fraccão f e uma lista de fracções l, selecciona de l os elementos que sao maiores do que o dobro de f.

```
twiceBiggerThen :: Frac -> [Frac] -> [Frac]
twiceBiggerThen _ [] = []
twiceBiggerThen f (h:ts) = if (h > (2 * f)) then h : (twiceBiggerThen f ts) else
    (twiceBiggerThen f ts)
```

2. Relembre o tipo definido na Ficha 9 para representar expressões inteiras. Uma possível generalizaçao desse tipo de dados, será considerar expressões cujas constantes são de um qualquer tipo numérico (i.e., da classe Num).

```
data Exp a = Const a
           | Simetrico (Exp a)
           | Mais (Exp a) (Exp a)
           | Menos (Exp a) (Exp a)
           | Mult (Exp a) (Exp a)
```

(a) Declare Exp a como uma instancia de Show.

```
instance Show ExpInt where
    show x = infixx x
```

-- import da Ficha 9 feita no inicio

(b) Declare Exp a como uma instancia de Eq.

```
instance Eq ExpInt where
    (==) x y = (calcula x) == (calcula y)
```

(c) Declare Exp a como instancia da classe Num.

```
instance Num ExpInt where
    (+) x y = (Mais x y)
    (-) x y = (Menos x y)
    (*) x y = (Mult x y)
    abs x = if (calcula x) < 0 then (Simetrico x) else x
    signum x = if (c == 0) then (Const 0) else if (c > 0) then (Const 1) else (Const (-1))
where c = calcula x
    fromInteger y = (Const (fromInteger y))
```

3. Relembre o exercício da Ficha 7 sobre contas bancarias, com a seguinte declaração de tipos

-- Ex 3 - Retoma da Ficha 7

```
data Movimento = Credito Float | Debito Float
data Data = D Int Int Int       -- Dia Mes Ano
data Extracto = Ext Float [(Data, String, Movimento)]
```

(a) Defina Data como instancia da classe Ord.

```
instance Eq Data where
    (==) (D d1 m1 a1) (D d2 m2 a2) = (a2
== a1) && (m2 == m1) && (d2 == d1)

instance Ord Data where
    compare (D d1 m1 a1) (D d2 m2 a2) = if
((a1 == a2) && (m1 == m2) && (d1 == d2)) then
EQ else if ((a2 > a1) || ((a2 == a1) && (m2 > m1)) ||
((a2 == a1) && (m2 == m1) && (d2 > d1))) then
GT else LT
```

(b) Defina Data como instancia da classe Show.

```
instance Show Data where
    show (D d m a) = (show a) ++ "/" ++ (show
m) ++ "/" ++ (show d)
```

(c) Defina a funçao ordena :: Extracto -> Extracto, que transforma um extracto de modo a que a lista de movimentos apareça ordenada por ordem crescente de data.

```
ordena :: Extracto -> Extracto
ordena (Ext st moves) = Ext st (oAux moves []) where
    oAux [] new = new
    oAux (h:ts) new = oAux ts (insOrdAux h new)
    insOrdAux x [] = [x]
    insOrdAux (dt,x,y) ((date,strg,move):ts) = if (dt > date) then (date,strg,move) :
    (insOrdAux (dt,x,y) ts) else (dt,x,y) : (date,strg,move) : ts
```

(d) Defina Extracto como instancia da classe Show, de forma a que a apresentação do extracto seja por ordem de data do movimento com o seguinte, e com o seguinte aspecto

```
Saldo anterior: 300
---------------------------------------
Data       Descricao    Credito    Debito
---------------------------------------
2010/4/5   DEPOSITO     2000
2010/8/10  COMPRA                  37,5
2010/9/1   LEV                     60
2011/1/7   JUROS        100
2011/1/22  ANUIDADE                8
---------------------------------------
Saldo actual: 2294,5
```

```
instance Show Extracto where
show (Ext st list) = "Saldo anterior: " ++
(show st) ++ "\n-----------------------------------
---------
\nData\t\tDescricao\tCredito\tDebito\n--------
-----------------------------------\n" ++ (concat
(map auxMovimento list)) ++ "----------------
----------------------------\nSaldo actual: " ++
(show (saldo (Ext st list))) where

auxMovimento (dt,strg,Credito x) = if
(length strg) < 7 then (show dt) ++ "\t" ++
strg ++ "\t\t" ++ (show x) ++ "\n" else (show
dt) ++ "\t" ++ strg ++ "\t" ++ (show x) ++
"\n"

auxMovimento (dt,strg,Debito x) = if (length
strg) < 7 then (show dt) ++ "\t" ++ strg ++
"\t\t\t" ++ (show x) ++ "\n" else (show dt) ++
"\t" ++ strg ++ "\t\t" ++ (show x) ++ "\n"
```