Ficha 9

Programação Funcional

LEI 1º ano

1. Considere o seguinte tipo de dados para representar fracções

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

   (a) Defina a função `normaliza :: Frac -> Frac`, que dada uma fracção calcula uma fracção equivalente, irredutível, e com o denominador positivo.
   Por exemplo: `normaliza (F (-33) (-51)) = (F 11 17)` e `normaliza (F 50 (-5)) = (F (-10) 1)`. Relembre a função `mdc` que definiu na Ficha 6.

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

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

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

   (e) Defina `Frac` como instância da classe `Num`. Relembre que a classe `Num` tem a seguinte definição

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

   (f) Defina uma função que, dada uma fracção `f` e uma lista de fracções `l`, selecciona de `l` os elementos que são maiores do que o dobro de `f`.

2. Considere o seguinte tipo para representar expressões inteiras.

```
data ExpInt = Const Int
        | Simetrico ExpInt
        | Mais ExpInt ExpInt
        | Menos ExpInt ExpInt
        | Mult ExpInt ExpInt
```

   Os termos deste tipo `ExpInt` podem ser vistos como árvores cujas folhas são inteiros e cujos nodos (não folhas) são operadores.

   (a) Defina uma função `calcula :: ExpInt -> Int` que, dada uma destas expressões calcula o seu valor.

   (b) Defina `ExpInt` como uma instância da classe `Show` de forma a que `show (Mais (Const 3) (Menos (Const 2)(Const 5)))` dê como resultado `"(3 + (2 - 5))"`.

   (c) Defina uma outra função de conversão para strings `posfix :: ExpInt -> String` de forma a que quando aplicada à expressão acima dê como resultado `"3 2 5 - +"`.

   (d) Defina `ExpInt` como uma instância da classe `Eq`.

   (e) Defina `ExpInt` como instância desta classe `Num`.

3. Uma outra alternativa para representar expressões é como o somatório de factores em que cada factor é o produto de constantes.

```
data ExpN = N [Parcela]
type Parcela = [Int]
```

   (a) Defina uma função `calcN :: ExpN -> Int` de cálculo do valor de expressões deste tipo.

   (b) Defina uma função de conversão `normaliza :: ExpInt -> ExpN`.

   (c) Defina `ExpN` como instância da classe `Show`.

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

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

   (a) Defina `Data` como instância da classe `Ord`.

   (b) Defina `Data` como instância da classe `Show`.

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

   (d) Defina `Extracto` como instância 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
```

   (e) A função `dmaxDebito`, a seguir apresentada, calcula a data e o montante do maior débito de um extracto.

```
dmaxDebito :: Extracto -> Maybe (Data,Float)
dmaxDebito ((_,Debito,d,m):t) = maxdeb (d,m) (dmaxDebito t)
dmaxDebito (h:t) = dmaxDebito t
dmaxDebito [] = Nothing
```

   Apresente a definição de `maxdeb` e indique claramente o seu tipo.

5. Uma possível generalização do tipo de dados apresentado na alínea 2, 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)
```

Declare `Exp` como instância da classe `Num`, completando a seguinte definição:

```
instance (Num a) => Num (Exp a) where
  .......
```

Note que, em rigor, deverá ainda definir o tipo `Exp a` como uma instância de `Show` e de `Eq`.