Aula Teórico-prática 10

Programação Funcional

LEI 1º ano

1. Relembre da ficha 9 os tipos `ExpInt` e `ExpN`, juntamente com as funções:

```
calcula :: ExpInt -> Int
expString :: ExpInt -> String
posfix :: ExpInt -> String
calcN :: ExpN -> Int
normaliza :: ExpInt -> ExpN
```

   Uma possível generalização será considerar expressões cujas constantes são de um qualquer tipo numérico (i.e., da classe `Num`). A definição desses tipos será agora

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

type ExpN a = [Parcela a]
type Parcela a = [a]
```

   (a) Adapte as definições que apresentou das funções referidas a estes novos tipos.

   (b) Sabendo o seguinte como *output* do `ghci`

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

   complete 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`.

2. O tipo `ExpN a`, pelo facto de não ter uma declaração `data`, não pode ser declarado como instância de nenhuma classe. Considere a seguinte declaração alternativa:

```
data EXPN a = Expn [Parcela a]
```

   (a) Defina `EXPN a` como instância da classe `Eq`, de modo a que a função `(==)` teste se duas expressões são equivalentes.

   (b) Defina `EXPN a` como instância da classe `Show`.