Aula Teórico-prática 9

Programação Funcional

LEI 1º ano

1. Considere o seguinte tipo para representar expressões.

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

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

   (b) A função `show`, que veremos em mais detalhe numa outra fase, pode ser usada para converter um inteiro para uma *string*. Por exemplo, `show 123 = "123"`. Usando a função `show`, defina uma função `expString :: ExpInt -> String` para converter expressões em strings. Por exemplo, `expString (Mais (Const 3) (Menos (Const 2)(Const 5)))` deverá dar como resultado a `"(3 + (2 - 5))"`.

   (c) Defina uma outra função de conversão para strings `posfix :: ExpInt -> String` de forma a que `posfix (Mais (Const 3) (Menos (Const 2)(Const 5)))` dê como resultado a `"3 2 5 - +"`.

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

2. Uma outra alternativa para representar expressões é como o somatório de parcelas em que cada parcela é o produto de constantes.

```
type ExpN = [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 uma função de conversão de expressões normalizadas para *string*s.

   Usando as funções anteriores podemos definir uma função de simplificação de expressões:

```
simplifica :: ExpInt -> String
simplifica e = expNString (normaliza e)
```