

Aula Teórico-prática e Prática 9

Programação Funcional

LCC 1º ano

1. 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) 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 :: Exp -> String` para converter expressões em strings. Por exemplo quando aplicada à expressão

```
Mais (Const 3) (Menos (Const 2)(Const 5))
```

deverá dar como resultado `"(3 + (2 - 5))"`.

- (c) Defina uma outra função de conversão para strings `posfix::Exp -> String` de forma a que quando aplicada à expressão acima dê como resultado a `"3 2 5 - +"`.
2. Uma outra alternativa para representar expressões é como o somatório de factores em que cada factor 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 *strings*.

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

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