DECSI - UFOP
Grupo de Estudos de Programação Funcional 2º semestre de 2013
Professor: Rodrigo Geraldo Ribeiro
e-mail: rodrigogribeiro@decsi.ufop.br

Lista de Exercícios 2 **Tema: Funções de Ordem Superior**

Em análise numérica, a *regra de Horner* é um algoritmo eficiente para a avaliação / representação de polinômios.

Dado o polinômio:

$$p(x) = \sum_{k=0}^{n} a_k x^k = a_n x^n + a_{n-1} x^{n-1} + ... + a_1 x^1 + a_0$$

onde $a_0, ..., a_n$ são números reais, podemos ordená-lo usando uma ordem decrescente de expoentes. Uma vez que os expoentes são apresentados em ordem decrescente, podemos representar um polinômio utilizando apenas uma lista que contêm seus coeficientes. Desta maneira, o seguinte tipo Haskell pode ser utilizado para representar um polinômio:

```
type Polynomial = [Int]
```

isto é, um polinômio é representado como uma lista de seus coeficientes em ordem decrescente de expoentes. Se um valor a_i está na i-ésima posição da lista, então o coeficiente a_i corresponde ao termo x^i. Isto é, a i-ésima posição da lista representa o termo correspondente ao expoente i. Considere o seguinte polinômio:

$$p(x) = 4x^3 + 2x^2 - 5$$

Este seria representado como:

```
px = [4, 2, 0, -5]
```

Cabe ressaltar que caso o polinômio não possua um termo correspondente a um determinado expoente, este é representado com o coeficiente 0. No polinômio $4x^3+2x^2-5$, não existe termo para o expoente 1, e, portanto, este é representado utilizando o valor 0 como seu coeficiente.

Baseado no que foi apresentado, desenvolva o que se pede a seguir. Para cada item, é apresentado um exemplo envolvendo o polinômio $4x^3 + 2x^2 - 5$, isto é, `[4,2,0,-5]`.

1. Desenvolva a função:

   ```
   grade :: Polynomial -> Int
   ```

   que retorna o maior expoente do polinômio fornecido como parâmetro. Exemplo:

   ```
   grade [4, 2, 0, -5] = 3
   ```

2. Para somar dois polinômios basta somar os coeficientes de termos de mesmo expoente. O objetivo deste exercício é implementar a função:

```
(.+.) :: Polynomial -> Polynomial -> Polynomial
```

que recebe dois polinômios como parâmetros e retorna como resultado a soma destes. Porém, nem sempre os dois polinômios a serem somados possuem o mesmo grau (maior expoente). Como exemplo, considere

$$\begin{aligned} p_1(x) &= 4x^3 + 2x^2 - 5 \\ p_2(x) &= 6x^2 + 1 \end{aligned}$$

Portanto, temos que: $p_1(x) + p_2(x) = 4x^3 + 8x^2 - 4$. Um inconveniente deste fato, é que a representação de polinômios não está normalizada, isto é, ambos os polinômios a serem somados não possuem o mesmo grau. No exemplo anterior, temos que:

```
p1x = [4, 2, 0, -5]
p2x = [6, 0, 1]
```

A versão normalizada destes polinômios (isto é, ambos possuindo o mesmo grau) seria:

```
p1x = [4, 2, 0, -5]
p2x = [0, 6, 0, 1]
```

Observe que a única alteração realizada foi acrescentar no segundo polinômio o coeficiente 0, para que ambos possuíssem termos de expoente 3.

(a) Desenvolva a função:

```
normalize :: Polynomial -> Polynomial -> (Polynomial, Polynomial)
```

que recebe como parâmetro dois polinômios possivelmente não normalizados e retorna um par contendo os dois polinômios fornecidos como parâmetros normalizados. Exemplo:

```
normalize [4, 2, 0, -5] [6, 0, 1] = ([4,2,0,-5], [0,6,0,1])
```

(b) Utilizando a função `normalize`, desenvolvida no item anterior, implemente a função

```
(.+.) :: Polynomial -> Polynomial -> Polynomial
```

que soma dois polinômios fornecidos como argumento. Exemplo:

```
[4,2,0,-5] .+. [0,6,0,1] = [4, 8, 0, -4]
```

3. Seja $p(x)$ o seguinte polinômio

$$p(x) = \sum_{k=0}^{n} a_k x^k = a_n x^n + a_{n-1} x^{n-1} + ... + a_1 x^1 + a_0$$

A derivada de $p(x)$, $\frac{d}{dx}(p(x))$, é definida como:

$$\frac{d}{dx}(p(x)) = \sum_{k=1}^{n} k \times (a_k x^{k-1}) = n \times a_n x^{n-1} + (n-1) \times a_{n-1} x^{(n-1)-1} + ... + 1 \times a_1 x^0$$

Como exemplo, considere o polinômio $4x^3 + 2x^2 - 5$. A derivada deste é igual a:

$$\frac{d}{dx}(4x^3 + 2x^2 - 5) = 3 \times 4x^2 + 2 \times 2x^1 = 12x^2 + 4x$$

Com base no apresentado, desenvolva a função:

```
derivative :: Polynomial -> Polynomial
```

que dado um polinômio como entrada, retorna a derivada deste. Exemplo:

```
derivative [4, 2, 0, -5] = [12, 4, 0]
```

4. A representação de Horner pode ser utilizada para obter um algoritmo eficiente para calcular o valor de um polinômio para um determinado valor x_0. Seja $p(x)$ o polinômio:

$$p(x) = \sum_{k=0}^{n} a_k x^k = a_n x^n + a_{n-1} x^{n-1} + ... + a_1 x^1 + a_0$$

e x_0 um inteiro qualquer. Então $p(x_0)$ é igual a:

$$p(x_0) = a_0 + x \times (a_1 + x \times (a_2 + ... + x \times (a_{n-1} + x \times a_n \quad \underbrace{)...))}_{\text{n parêntesis}}$$

Com base no apresentado, desenvolva a função:

```
eval :: Int -> Polynomial -> Int
```

que recebe um valor inteiro e um polinômio e retorna como resultado o valor deste polinômio para o inteiro fornecido como parâmetro. Exemplo:

```
eval 2 [4 , 2, 0, -5] = 35
```