Programação Funcional
Lista de Exercícios 03

Prof. Wladimir Araújo Tavares

1. A função do prelúdio scanl é uma variante do foldl que produz a lista com os valores acumulados:

   `scanl f z [x1, x2, ...]  = [z, f z x1, f (f z x1) x2, ...]`

   Por exemplo:

   `scanl (+) 0 [1, 2, 3] = [0, 0 + 1, 0 + 1 + 2, 0 + 1 + 2 + 3] = [0, 1, 3, 6]`

   (a) Use a função `scanl` para definir a função `fatAcc n` que retorna uma lista de com $[1!, 2!, \ldots, n!]$

   `fatAcc 10 == [1,2,6,24,120,720,5040,40320,362880,3628800]`

   (b) Use a função fatAcc para definir a função `fatorial`.

2. Escreva uma função `dotprod :: [Float] -> [Float] -> Float` para calcular o produto interno de dois vetores (representados como listas):

   $dotprod[x_1, \ldots, x_n][y_1, \ldots, y_n] = x_1 * y_1 + \ldots + x_n * y_n = \sum_{i=1}^{n} x_i * y_i$

   Usando a função `zipWith` e `sum`.

3. Um trio (x, y, z) de inteiros positivos diz-se pitagórico se $x^2 + y^2 = z^2$. Defina a função `pitagoricos :: Int -> [(Int, Int, Int)]` que calcule todos os trios pitagóricos cujas componentes não ultrapassem o argumento.

   Por exemplo: `pitagoricos 10 = [(3, 4, 5), (4, 3, 5), (6, 8, 10), (8, 6, 10)]`.

   Use compreensão de listas `[(x,y,z) | x <- [],y<-[],z<-[],predicado]`

4. Considere a seguinte série (i.e. somas infinitas) que convergem para π:

$$\pi = \frac{4}{1} - \frac{4}{3} + \frac{4}{5} - \frac{4}{7} + \ldots \tag{1}$$

   (a) Construa uma lista infinita com os numerados das parcelas. $[4, -4, \ldots, 4(-1)^n]$

   (b) Construa uma lista infinita com os denominadores das parcelas.

   (c) Combine as duas listas usando a função `zipWith`.

   (d) Defina a função `calcPi1` n que calcula o valor de pi somando n parcelas da série. Calcule o valor o somatório para 10, 100 e 1000 parcelas.

   (e) Qual é o valor de pi considerando como critério de parada erro absoluto 0.01?

   (f) Qual é o valor de pi considerando como critério de parada erro relativo 0.01?

5. Considere a seguinte série (i.e. somas infinitas) que convergem para π:

$$\pi = 3 + \frac{4}{2*3*4} - \frac{4}{4*5*6} + \frac{4}{6*7*8} - \ldots \tag{2}$$

   (a) Defina a função `calcPi2` n que calcula o valor de pi somando n parcelas da série. Calcule o valor o somatório para 10, 100 e 1000 parcelas.

6. (a) Escreva uma função binom com dois argumentos que calcule o coeficiente

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} \tag{3}$$

   Sugestão: pode exprimir n! como product [1..n]

   (b) Para todas as listas de números xs e ys, temos que `product (xs++ys) = product xs ++ product ys`. Use esta propriedade para re-escrever a definição de forma mais eficiente, eliminando factores comuns entre o numerador e denominador.

   (c) Usando uma função binom que calcula o coeficiente binomal, escreva uma definição da função `pascal :: Int -> [[Int]]` que calcula as primeiras linhas triângulo de Pascal.

$$\binom{0}{0}$$

$$\binom{1}{0} \quad \binom{1}{1}$$

$$\binom{2}{0} \quad \binom{2}{1} \quad \binom{2}{2}$$

$$\binom{3}{0} \quad \binom{3}{1} \quad \binom{3}{2} \quad \binom{3}{3}$$

   `pascal 4 == [[1],[1,1],[1,2,1],[1,3,3,1],[1,4,6,4,1]]`

   (d) defina o triângulo de Pascal completo como uma lista infinita `pascal2 :: [[Int]]` das linhas do triângulo. Note que que `pascal2 !! n !! k = binom n k`, para quaisquer n e k tais que $n \geq 0$ e $0 \leq k \leq n$.

   Exemplos:

   `pascal2 !! 6 == [1,6,15,20,15,6,1]`

   `pascal2 !! 6 !! 3 == 20`

   (e) Escreva outra definição `pascal3 :: [[Int]]` que evite o cálculo de factoriais usando as seguintes propriedades de coeficientes binomiais:

$$\binom{n}{0} = \binom{n}{n} = 1 \quad \binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k} (0 < k < n) \tag{4}$$

   Dica: defina uma função `table i j` que consulta a lista infinita `pascal3`.

7. Considere um polinômio $P(X) = c_0 + c_1 z + \ldots + c_n z^n$ representado pela lista dos seus coeficientes $[c0, c1, \ldots; c_n]$. Podemos calcular o valor do polinômio num ponto de forma eficiente usando a forma de Horner :

$$P(z) = c0 + c_1 z + \ldots + c_n z_n = c_0 + z*(c_1 + z*(\ldots + z*(c_{n-1} + z*c_n)\ldots)) \tag{5}$$

   Note que usando a expressão não necessitamos de calcular potências: para calcular o valor dum polinômio de grau n usamos apenas n adições e n produtos.

   Complete a seguinte definição recursiva tal que horner cs z calcula o valor do polinômio com lista de coeficientes cs no ponto z usando a forma de Horner.

```
horner :: [Double] -> Double -> Double
horner [] z = 0
horner (c:cs) z =
```

   A forma de Horner também pode ser expressa como aplicação da função de ordem superior foldr . Complete a definição seguinte de forma a que a igualdade seja correta.

   `horner cs z = foldr ____________________________ ______ cs`

8. A lista infinita de números naturais [1,2,3,4,..] pode ser definida de várias maneiras em Haskell:

   `number1 = [1..]`

   `number2 = 1 : zipWith (+) number2 [1,1..]`

   `number3 = scanl (+) 1 [1,1..]`

   `number4 = 1 : [ x+y | (x,y) <- zip number4 [1,1..] ]`

   Os números triangulares são os números da seguinte forma $T_n = T_{n-1} + n$ e $T_0 = 0$. Defina uma lista infinita dos números triangulares `triangular :: [[Int]]`.

   take 10 triangular == [0,1,3,6,10,15,21,28,36,45]

9. A função length, que computa o número de elementos de uma lista, pode ser definida do seguinte modo:

```
length xs = length' 0 xs
        where length' n [] = n
length' n (x:xs) = length' (n+1) xs
```

   Essa função usa a função auxiliar length', que possui um parâmetro adicional para acumular o resultado. A função `length'` é definida usando recursão de cauda, uma vez que a chamada recursiva `length' (n+1) xs`, usada no lado direito da definição, não ocorre dentro de nenhum argumento de outra função. Use essa técnica de recursão de cauda para definir as seguintes funções:

   (a) `fac :: Int -> Int`, que computa o fatorial de um número natural

   (b) `reverse :: [a] -> [a]`, que inverte uma lista.

10. Considere a seguinte definição de função em Haskell:

    `until p f x = if p x then x else until p f (f x)`

    (a) Qual é o tipo da função until?

    (b) Qual é o resultado da avaliação da expressão until (i10) square 2?

    (c) Use a função until para definir uma função que, dado um string s, retorne o string obtido removendo-se todos os caracteres iguais a branco que ocorrem no início de s.

11. **INPUT:** Número n inteiro positivo
    OUTPUT: Lista de tuplas (f, p) que representam os fatores primos de n onde f denota o fator propriamente dito e p seu respectivo expoente. (Todo número x, tal que $x \in \mathbb{N}$, pode ser reescrito como o produto de potências de bases primas e expoentes naturais. Por exemplo, o número 3361743 pode ser reescrito na forma,

$$3361743 = 3^4 \times 7^3 \times 11^2 \tag{6}$$

    Os números 3, 7 e 11 são denominados fatores primos de 3361743 e 4, 3 e 2 seus respectivas expoentes.)

    PROT:

    `factors :: (Integral a) => a -> [(a,a)]`

    EX(S):

    `factors 3361743 => [(3,4),(7,3),(11,2)]`

12. Defina cada uma das funções a seguir, usando foldr e foldl:

    (a) `elem :: a -> [a] -> Bool`, que determina se um valor ´e uma elemento de uma lista.

    (b) `remdups :: Eq a => [a] -> [a]` que remove da lista elementos duplicados adjacentes.