Programação Funcional
Revisão 1 AP

Prof. Wladimir Araújo Tavares

1. Quais são os tipos das seguintes funções:

   (a)
   ```
   remove x [] = []
   remove x (y:ys) = if x == y then ys else y: (remove x ys)
   ```

   (b)
   ```
   partes [] = [ [] ]
   partes (x:xs) = [ x:y | y <- partes xs] ++ partes xs
   ```

   (c)
   ```
   rota n xs = drop n xs ++ take n xs
   ```

   (d)
   ```
   swap (x,y) = (y,x)
   ```

   (e)
   ```
   twice f x = f (f x)
   ```

2. Explique o que cada função da questão anterior faz?
3. Defina cada uma das seguintes funções usando apenas funções pré-definidas em Haskell:

   (a) `prodMn :: Int -> Int -> Int` que, dados dois valores inteiros m e n, retornar o produto de todos os valores inteiros entre m e n (inclusive).

   (b) `sumQuad :: Int -> Int` que, dado um valor inteiro positivo n, retorna a soma dos quadrados de todos os inteiros compreendidos entre 0 e n.

   (c) `interior :: [a] -> [a]`, que dada uma lista retorna uma lista obtida eliminando os extremos da lista. Exemplo: `interior [2,3,5,7] == [3,5]`

   (d) `segmento :: Int -> Int -> [a]` tal que (segmento m n xs) retorna uma lista dos elementos de xs compreendidos entre as posições m e n. Exemplo:

   ```
   segmento 3 4 [3,4,1,2,7,9,0] == [1,2]
   segmento 3 5 [3,4,1,2,7,9,0] == [1,2,7]
   segmento 5 3 [3,4,1,2,7,9,0] == []
   ```

4. Defina as seguintes funções usando compreensão de listas:

   (a) `subconjunto :: Eq a => [a] -> [a] -> Bool` tal que (subconjunto xs ys) verifica xs é um subconjunto de ys. Por exemplo,

   ```
   subconjunto [3,2,3] [2,5,3,5] == True
   subconjunto [3,2,3] [2,5,6,5] == False
   ```

   Dica: use a função `elem :: a -> [a] -> Bool` verifica se um elemento pertence a uma lista.

   (b) `union :: Eq a => [a] -> [a] -> [a]` tal que (union xs ys) é a união dos conjuntos xs e ys. Por exemplo,

   ```
   union [3,2,5] [5,7,3,4] == [3,2,5,7,4]
   ```

   Dica: use a função `notElem :: a -> [a] -> Bool` verifica se um elemento não pertence a uma lista.

   (c) `diferencia :: Eq a => [a] -> [a] -> [a]` tal que (diferencia xs ys) é a diferença entre os conjuntos xs e ys. Por exemplo,

   ```
   diferencia [3,2,5,6] [5,7,3,4] == [2,6]
   diferencia [3,2,5] [5,7,3,2] == []
   ```

5. A soma da série

$$\frac{\pi^2}{6} = \frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + \ldots \quad (1)$$

   Portanto, o valor de π pode ser aproximado mediante a raiz quadrada de 6 vezes a soma da série. Defina a função aproximaPi tal que (aproximaPi) é a aproximação obtida pela soma dos n primeiros termos da série. Por exemplo,

   ```
   aproximaPi 4 == 2.9226129861250305
   aproximaPi 1000 == 3.1406380562059946
   ```

6. Dado uma lista de números inteiros, definiremos o maior salto como o maior valor das diferenças (em valor absoluto) entre números consecutivos da lista. Por exemplo, dada uma lista [2,5,-3,7]

   - 3 (valor absoluto de 2 - 5)
   - 8 (valor absoluto de 5 - (-3))
   - 10 (valor absoluto de -3 - 7)

   Portanto o maior salto é 10. Não esta definido o maior salto para uma lista com menos de 2 elementos. Defina a função `maiorSalto :: [Integer] -> Integer` tal que maiorSalto xs é o maior salto da lista xs. Por exemplo,

   ```
   maiorSalto [1,5] == 4
   maiorSalto [10,-10,1,4,20,-2] == 22
   ```

7. Defina as seguintes funções a seguir usando foldr:

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

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

   (c) `todos :: (a -> Bool) -> [a] -> Bool` que dado um predicado `p :: a -> Bool` e uma lista de valores do tipo a, determina se todos os elementos da lista satisfazem o predicado p.

   (d) Usando essa função anterior, defina a função `positivos :: [Int] -> Bool`, que determina se todos os elementos da lista dada como argumento são inteiros positivos.

   (e) `algum :: (a -> Bool) -> [a] -> Bool` que, dado um predicado `p :: a -> Bool` e uma lista de valores do tipo a, determina se algum dos elementos da lista satisfaz o predicado p.

   (f) Usando essa função anterior, defina a função `membro :: a -> [a] -> Bool` que determina se o valor dado como primeiro argumento ocorre na lista dada como segundo argumento.

8. Uma uma lista xs é um prefixo de uma lista ys se `ys=xs++zs` para alguma lista zs. Por exemplo "ban"é um prefixo de "banana".

   Defina uma função recursiva `prefix :: Eq a => [a] -> [a] -> Bool` tal que prefix xs ys retorna True se xs é um prefixo de ys e retorna False em caso contrário.

9. Uma lista xs é uma subsequência de uma lista ys se `ys=as++xs++zs` para alguma lista zs e alguma lista as. Por exemplo "ana"é uma subsequência de "banana". Defina uma função recursiva `subsequence :: Eq a => [a] -> [a] -> Bool` tal que subsequence xs ys retorna True se xs é uma subsequência de ys e retorna False em caso contrário.
10. **Ordenação de listas pelo método merge sort.**

    (a) Usando as funções do prelúdio-padrão, escreva uma função metades que divide uma lista de comprimento par em duas com metade do comprimento.

    Exemplo: `metades [1, 2, 3, 4, 5, 6, 7, 8] = ([1, 2, 3, 4], [5, 6, 7, 8])`.

    Investigue o acontece se a lista tiver comprimento ímpar.

    (b) Escreva uma definição recursiva da função `merge :: Ord a => [a] -> [a] -> [a]` para juntar duas listas ordenadas numa só mantendo a ordenação. Exemplo: `merge [3, 5, 7] [1, 2, 4, 6] = [1, 2, 3, 4, 5, 6, 7]`.

    (c) Usando a função merge, escreva uma definição recursiva da função `mergesort :: Ord a => [a] -> [a]` que implementa o método merge sort :

    - uma lista vazia ou com um só elemento já está ordenada;
    - para ordenar uma lista com dois ou mais elementos, partimos em duas metades, recursivamente ordenamos as duas parte e juntamos os resultados usando merge.

11. Um inteiro positivo n diz-se perfeito se for igual à soma dos seus divisores (excluindo o próprio n). Defina uma função `perfeitos :: Int -> [Int]` que calcula a lista de todos os números perfeitos até um limite dado como argumento.

    Exemplo: `perfeitos 500 = [6, 28, 496]`.

12. Escreva uma função `permutations :: [a] -> [[a]]` para obter a lista com todas as permutações dos elementos uma lista. Assim, se xs tem comprimento n, então permutations xs tem comprimento n!.

    Exemplo: `permutations [1, 2, 3] = [[1, 2, 3], [2, 1, 3], [2, 3, 1], [1, 3, 2], [3, 1, 2], [3, 2, 1]]`

    Note que a ordem das permutações não é importante.

13. Escreva um programa que leia uma cadeia de caracteres da entrada padrão que representa uma expressão em notação polonesa reversa — com operadores `+`, `-` e `*` — e imprima o resultado da avaliação dessa expressão.

    Por exemplo, ao ler `"1 3 + 2 *`, o programa deve imprimir `8`.

    O programa não precisa considerar o caso de a entrada não estar correta, isto é, não for uma expressão correta em notação polonesa.