Princípios de Programação

Exercícios

Universidade de Lisboa
Faculdade de Ciências
Departamento de Informática
Licenciatura em Engenharia Informática

2019/2020

Funções de ordem superior

1. Descreva o comportamento e um tipo para cada uma das seguintes secções.

   (a) `(*2)`

   (b) `(>0)`

   (c) `(1/)`

   (d) `(/2)`

   (e) `(+1)`

   (f) `(++"\n")`

2. Qual a diferença entre as seguintes funções? Discuta o tipo de cada uma.

   (a) `add1 (x, y)= x + y`

   (b) `add2 x y = x + y`

   (c) `add3 x = (x+)`

   Escreva a função `successor:: Int -> Int` recorrendo a cada uma delas.

3. Determine um tipo e o valor para cada expressão.

   (a) `map (+1)[1..3]`

   (b) `map (>0)[3,-5,-2,0]`

   (c) `map (++"s")["A", "arte", "do", "aluno"]`

(d) `map ("s"++)["o", "aluno","bem-comportado"]`

(e) `let f x = x * x in map (map f)[[1,2],[3,4,5]]`

(f) `filter (>5)[1..6]`

(g) `filter even [1..10]`

(h) `filter (>0)(map (^2)[-3..3])`

(i) `map (^2)(filter (>0)[-3..3])`

4. Defina a função `zipWith' :: (a->b->c)-> [a] -> [b] -> [c]` semelhante à função `zip` mas que aplica uma dada função a cada par de valores.

   (a) Utilize uma função recursiva.

   (b) Tente agora uma solução com listas em compreensão.

   (c) Defina a função `zip` utilizando a função `zipWith` .

5. A função `takeWhile` é semelhante à função `take` com a diferença que espera, como primeiro argumento, um predicado em vez de um inteiro. O valor de `takeWhile p xs` é o mais longo segmento inicial de `xs` cujos elementos verificam `p`. Por exemplo:

   ```
   ghci> takeWhile even [4,2,6,1,8,6,2]
   [4,2,6]
   ```

   Defina por recursão a função `takeWhile`.

6. Defina a função `dropWhile :: (a -> Bool)-> [a] -> [a]`, que elimina os primeiros elementos da lista enquanto o predicado se verificar.

7. `dropUntil :: (a -> Bool)-> [a] -> [a]` que elimina os primeiros elementos da lista até que um deles satisfaça a condição dada. Exemplo:

   ```
   ghci> dropUntil (>0) [-4, 0, -8, 3, -2, -5, 3]
   [3, -2, -5, 3]
   ```

8. Defina uma função `total :: (Int -> Int)-> Int -> Int`, de modo a que `total f` é a função que, no ponto `n`, retorna $\sum_{i=0}^{n} f(i)$.

   (a) Utilize uma função recursiva.

   (b) Utilize uma lista em compreensão.

   (c) Utilize a função `map`.

9. Defina a função `aplica :: [a -> a] -> [a] -> [a]` que dada uma lista de funções e uma lista de elementos, devolve a lista resultante de aplicar sucessivamente as funções da lista de funções aos valores da lista argumento. Exemplo, onde `5` resulta de multiplicar `1` por `2` e em seguida somar-lhe `3`:

```
ghci> aplica [(*2), (+3)] [1,3,0,4]
[5,9,3,11]
```

(a) Utilize uma função recursiva

(b) Utilize **`foldr`**

(c) Utilize **`foldl`**

10. Determine um tipo para cada uma das seguintes expressões lambda.

(a) `\x -> x + 1`

(b) `(\x -> x + 1)6`

(c) `\x -> x > 0`

(d) `\x y -> x + y`

(e) `(\x y -> x + y)7`

(f) `(\x y -> x + y)7 3`

(g) `\x -> (\y -> x + y)`

(h) `\f x -> f (f x)`

(i) `(\f x -> f (f x))(\y -> y + 1)`

11. Escreva a função `mult x y z = x * y * z` utilizando uma expressão lambda.

12. Escreva as secções `(++)`, `(++[1,2])`, `([1,2]++)` como expressões lambda. Quais os seus tipos?

13. Utilizando uma expressão lambda, escreva uma função `isNonBlank` com a assinatura **`Char -> Bool`** que devolve **`True`** apenas quando aplicada a caracteres não brancos, isto é, para caracteres que não pertencem à lista `[' ','\t','\n']`.

14. Dada uma função `f` do tipo `a -> b - > c` e dois parâmetros de tipos `a` e `b`, escreva uma expressão lambda
`(a -> b -> c)-> b -> a -> c` que se comporta como `f`, mas que aceita os seus argumentos por ordem inversa. Como aplicação, escreva uma função $x//y = (x/y)^{-1}$.

15. Defina as funções **`curry'`** `:: ((a, b)-> c)-> a -> b ->c` e **`uncurry'`** `:: (a -> b -> c)-> (a, b)-> c` que transforma uma função não *curried* numa função *curried* e vice-versa. Por exemplo:

```
ghci> let multPar (x, y) = x * y
ghci> multPar (3, 4)
12
ghci> let mult = curry' multPar
ghci> mult 3 4
```

```
12
ghci> let soma x y = x + y
ghci> soma 3 4
7
ghci> let somaPar = uncurry' soma
ghci> somaPar (3,4)
7
```

16. Determine um tipo e o valor para cada expressão.

    (a) `foldr (\y z -> y * 3 + z)0 [1..4]`

    (b) `foldr (\x y -> if x > 0 then x + y else y)0 [4,-3,2,-1]`

    (c) `foldr (\x y -> x ^ 2 + y)0 [2..5]`

    (d) `foldr (*)1 [-3..(-1)]`

    (e) `foldr (\x s -> if x == 'z' then x:s else s)[] "Oz␣alunoz␣dze␣PzPz"`

17. Escreva as funções **`sum`** e **`length`** usando o **`foldl`** e **`foldr`** respectivamente.

18. Apresente definições para **`map`** e **`filter`** recorrendo à função **`foldr`**.

19. Escreva um conversor binário para decimal utilizando uma das variantes da função `fold`. O número binário é apresentado por uma lista de inteiros. Por exemplo:

    ```
    ghci> binary2decimal [1,1,0,1]
    13
    ```

20. Escreva a função `indexOf` que recebe uma lista e um possível elemento dessa lista, e devolve o primeiro índice onde esse elemento se encontra, ou -1 caso esse elemento não se encontre na lista.

21. Um polinómio pode ser representado por uma lista de coeficientes. Por exemplo, a lista `[5,2,0,1,2]` representa o polinómio $5x^4 + 2x^3 + 0x^2 + 1x^1 + 2x^0 = 5x^4 + 2x^3 + x + 2$. Defina uma função `poly :: Int -> [Int] -> Int` que, dado um valor para x e um polinómio, calcule o valor do polinómio nesse ponto. Utilize a função **`foldl`** ou **`foldr`**.

22. Defina a função `selectApply :: (a -> b)-> (a -> Bool)-> [a] -> [b]` que devolve uma lista contendo os resultados de aplicar a função dada no primeiro argumento aos elementos da lista dada no terceiro argumento, que satisfaçam a condição dada no segundo argumento. Exemplo:

```
ghci> selectApply (*3) (>0) [-4..4]
[3,6,9,12]
```

23. Defina a função `histograma :: Eq a =>[a] -> [(a, Int)]` que receba uma lista e devolva uma lista de tuplos com cada elemento distinto da lista original e o número de ocorrências. Exemplo:

```
ghci> histograma [1, 1, 2, 3, 20, 3, 20, 3]
[(1,2),(2,1),(3,3),(20,2)]
```

24. Qual o tipo mais geral de **`map map`**? e o de **`map . map`**? Utilizando este último, escreva uma função `gz :: [[Int]] -> [[Bool]]` que transforme uma matriz (lista de listas) de inteiros numa matriz de valores lógicos, onde cada entrada indica se o valor inicial era ou não maior do que zero. Por exemplo:

```
ghci> gz [[1,2,3],[2,-1,3,7]]
[[True,True,True],[True,False,True,True]]
```

25. Utilizando o operador de composição, defina a função `iter` que, dado uma função f e um número natural n, devolve a função f^n, i.e., a função f aplicada a si mesma n vezes. Faça uma resolução recursiva e uma outra usando o **`foldr`**.

26. A função **`filter'`** pode ser definida em termos de **`.`**, **`concat`**, e **`map`**, da seguinte forma:

```
filter' :: (a -> Bool) -> [a] -> [a]
filter' p = concat . map box
          where
          box x = ...
```

    Dê uma definição para a função `box`.

27. Defina a função `sumlen` que recebe uma lista de inteiros e devolve um par cuja primeira componente é a soma da lista, e a segunda componente é o comprimento da lista. Use a função **`foldr`**.