# Ficha 2

## Programaçao Funcional

2015/16

1. Indique como é que o interpretador de haskell avalia as expressões das alíneas que se seguem, apresentando a cadeia de redução de cada uma dessas expressões (i.e., os varios passos intermédios até se chegar ao valor final).

   (a) Considere a definicao da seguinte função

   ```
   funA :: [Float] -> Float
   funA [] = 0
   funA (y:ys) = y^2 + (funA ys)
   ```

   Diga, justificando, qual é o valor de `funA [2,3,5,1]`.

   (b) Considere a definicao da seguinte função

   ```
   funB :: [Int] -> [Int]
   funB [] = []
   funB (h:t) = if (mod h 2)==0 then h : (funB t)
                                else (funB t)
   ```

   Diga, justificando, qual é o valor de `funB [8,5,12]`

2. Defina recursivamente as seguintes funções sobre listas:

   (a) `dobros :: [Float] -> [Float]` que recebe uma lista e produz a lista em que cada elemento é o dobro do valor correspondente na lista de entrada.

   **dobros :: [Float] -> [Float]**
   **dobros [] = []**
   **dobros (h:ts) = (2 * h) : dobros ts**

   (b) `numOcorre :: Char -> String -> Int` que calcula o número de vezes que um caracter ocorre numa string.

   **numOcorre :: Char -> String -> Int**
   **numOcorre _ [] = 0**
   **numOcorre c (h:ts) = if (c == h) then 1 + numOcorre c ts else numOcorre c ts**

   (c) `positivos :: [Int] -> Bool` que testa se uma lista só tem elementos positivos.

   **positivos :: [Int] -> Bool**

   **positivos [] = True**

   **positivos (h:ts) = if (h > 0) then positivos ts else False**

(d) `soPos :: [Int] -> [Int]` que retiras todos os elementos negativos de uma lista de inteiros.

```
soPos :: [Int] -> [Int]
soPos [] = []
soPos (h:ts) = if (h >= 0) then h : soPos ts else soPos ts
```

(e) `somaNeg :: [Int] -> Int` que soma todos os números negativos da lista de entrada.

```
somaNeg :: [Int] -> Int
somaNeg [] = 0
somaNeg (h:ts) = if (h < 0) then h + somaNeg ts else somaNeg ts
```

(f) `tresUlt :: [a] -> [a]` devolve os últimos três elementos de uma lista. Se a lista de entrada tiver menos de três elementos, devolve a propria lista.

```
tresUlt :: [a] -> [a]
tresUlt [] = []
tresUlt [a] = [a]
tresUlt [a,b] = [a,b]
tresUlt [a,b,c] = [a,b,c]
tresUlt (h:ts) = tresUlt ts
```

(g) `primeiros :: [(a,b)] -> [a]` que recebe uma lista de pares e devolve a lista com as primeiras componentes desses pares.

```
primeiros :: [(a,b)] -> [a]
primeiros [] = []
primeiros ((x,y):ts) = x : primeiros ts
```

3. Utilizando as funcoes `ord :: Char->Int` e `chr :: Int->Char` do módulo `Data.Char`, defina as seguintes funçoes:

| | |
|---|---|
| (a) `isLower :: Char -> Bool` | (d) `toUpper :: Char -> Char` |
| (b) `isDigit :: Char -> Bool` | (e) `intToDigit :: Int -> Char` |
| (c) `isAlpha :: Char -> Bool` | (f) `digitToInt :: Char -> Int` |

Note que todas estas funçoes já estão também definidas no módulo `Data.Char` .

```
-- a) verifica se é um caracter em minusculas.
isLower' :: Char -> Bool
isLower' c = (c >= 'a') && (c <= 'z')

-- b) verifica se o caracter é um digito.
isDigit' :: Char -> Bool
isDigit' c = (c >= '0') && (c <= '9')

-- c) verifica se o caracter pertence ao alphabeto.
isAlpha' :: Char -> Bool
isAlpha' c = ((c >= 'a') && (c <= 'z')) || ((c >= 'A') && (c <= 'Z'))

-- d) se o caracter dado é minuscúla, devolve esse caracter em maiúscula.
toUpper' :: Char -> Char
toUpper' c = if ((c >= 'a') && (c <= 'z')) then chr ((ord c) - 32) else c

-- e) passa um inteiro (entre 0 e 9) para caracter.
intToDigit' :: Int -> Char
intToDigit' x | ((x >= 0) && (x <= 9)) = chr ((ord '0') + x)

-- f) recebe um caracter e devolve o inteiro correspondente ao seu valor inteiro.
```

```
digitToInt' :: Char -> Int
digitToInt' c | ((c >= '0') && (c <= '9')) = (ord c) - (ord '0')
```

4. Usando as funcões do modulo `Data.Char`

   (a) Defina a funçao `primMai`, e o seu tipo, que recebe uma string como argumento e testa se o seu primeiro caracter é uma letra maiúscula.

   ```
   primMai :: String -> Bool
   primMai st = isUpper (head st)
   ```

   (b) Defina a funcão `segMin`, e o seu tipo, que recebe uma string como argumento e testa se o seu segundo caracter é uma letra minúscula.

   ```
   segMin :: String -> Bool
   segMin st = isLower (head (tail st))
   ```

5. Recorrendo a funçoes do modulo `Data.Char`, defina recursivamente as seguintes funções sobre strings:

   (a) `soDigitos :: [Char] -> [Char]` que recebe uma lista de caracteres, e selecciona dessa lista os caracteres que são algarismos.

   ```
   soDigitos :: [Char] -> [Char]
   soDigitos [] = []
   soDigitos (h:ts) = if (isDigit h) then h : soDigitos ts else soDigitos ts
   ```

   (c) `minusculas :: [Char] -> Int` que recebe uma lista de caracteres, e conta quantos desses caracteres sao letras minúsculas.

   ```
   minusculas :: [Char] -> Int
   minusculas [] = 0
   minusculas (h:ts) = if (isLower h) then 1 + (minusculas ts) else minusculas ts
   ```

   (d) `nums :: String -> [Int]` que recebe uma string e devolve uma lista com os algarismos que occorem nessa string, pela mesma ordem.

   ```
   nums :: String -> [Int]
   nums [] = []
   nums (h:ts) = if (isDigit h) then (digitToInt h) : nums ts else nums ts
   ```