Aula Prática 5

Paradigmas da Programação I / Programação Funcional

ESI/MCC 1^o ano (2005/2006)

Objectivos da aula: introduzir as funções de ordem superior `map` e `filter`. Continuar a praticar a utilização de listas definidas em compreensão e a definição de funções recursivas, comparando as diferentes formas de definir funções.

1 Funções de ordem superior

Funções de *ordem superior* são funções que recebem funções como parametro e/ou retornam funções como resultado. Por exemplo, `map` e `filter` são funções de ordem superior.

- A função `map` pode ser definida do seguinte modo:

```
map :: (a -> b) -> [a] -> [b]
map f [] = []
map f (x:xs) = (f x) : (map f xs)
```

  Assim, `(map f l)` aplica a função `f` a todo o elemento da lista `l`. Observe a concordância de tipos entre os elementos da lista `l` e o dominio da função `f`. Repare ainda que o resultado de `(map f l)` é o mesmo de `[ f x | x <- l ]`.

- A função `filter` pode ser definida por:

```
filter :: (a -> Bool) -> [a] -> [a]
filter p [] = []
filter p (x:xs) = if (p x) then x:(filter p xs)
                           else filter p xs
```

  Ou seja, `(filter p l)` seleciona/filtra da lista `l` os elementos que satisfazem o predicado `p`. Observe a concordância de tipos entre os elementos da lista `l` e o dominio do predicado `p`. Repare ainda que o resultado de `(filter p l)` é o mesmo de `[ x | x <- l, p x ]`.

Tarefa 1 Para cada uma das expressões seguintes, determine o seu valor e reescreva-a usando listas por compreensão.

1. `map odd [1,2,3,4,5]`
2. `filter odd [1,2,3,4,5]`
3. `map (\x-> div x 3) [5,6,23,3]`

4. `filter (\y-> (mod y 3 == 0)) [5,6,23,3]`
5. `filter (7<) [1,3..15]`
6. `map (7:) [[2,3],[1,5,3]]`
7. `map (:[]) [1..5]`
8. `map succ (filter odd [1..20])`
9. `filter odd (map succ [1..20])`

Temos, portanto, várias formas de definir funções sobre listas. Considere o exemplo da função `dobra :: [Int] -> [Int]` que recebe uma lista de inteiros e produz a lista dos seus dobros. Podemos definir `dobra` de diferentes modos:

Versão recursiva

```
dobra1 :: [Int] -> [Int]
dobra1 [] = []
dobra1 (x:xs) = (2*x):(dobra1 xs)
```

Usando listas por compreensão

```
dobra2 :: [Int] -> [Int]
dobra2 l = [ 2*x | x <- l ]
```

Usando funções de ordem superior

```
dobra3 :: [Int] -> [Int]
dobra3 l = map (2*) l       -- ou, alternativamente:   dobra3 = map (2*)
```

Tarefa 2 Use funções de ordem superior e recursividade, para definir duas versões distintas de cada uma das seguintes funções:

1.
```
maximos :: [(Double,Double)] -> [Double]
maximos l = [ max x y | (x,y) <- l ]
```

2.
```
positivos :: [Integer] -> [Integer]
positivos l = [ x | x <- l, x > 0 ]
```

Tarefa 3 Pretende-se guardar a informação sobre os resultados dos jogos de uma jornada de um campeonato de futebol na seguinte estrutura de dados:

```
type Jornada = [Jogo]
type Jogo = ((Equipa,Golos),(Equipa,Golos))
type Equipa = String
type Golos = Int
```

Defina as seguintes funções:

1. `igualj: Jornada -> Bool`
   que verifica se nenhuma equipa joga com ela própria.
2. `semrepet: Jornada -> Bool`
   que verifica se nenhuma equipa joga mais do que um jogo.
3. `empates: Jornada -> [(Equipa,Equipa)]`
   que dá a listas dos pares de equipas que empataram na jornada.
4. `equipas: Jornada -> [Equipa]`
   que dá a lista das equipas que participam na jornada.
5. `semrep: Jornada -> [(Equipa,Int)]`
   que calcula os pontos que cada equipa obteve na jornada (venceu - 3 pontos; perdeu - 0 pontos; empatou - 1 ponto)

Tarefa ** Use listas por compreensão e recursividade, para definir duas versões distintas de cada uma das seguintes funções:

1.
```
indicativo  :: [Int] -> [[Int]] -> [[Int]]
indicativo ind telefs = filter (concorda ind) telefs
  where concorda :: [Int] -> [Int] -> Bool
        concorda [] _ = True
        concorda (x:xs) (y:ys) = (x==y) && (concorda xs ys)
        concorda (x:xs) [] = False
```
   que recebe uma lista de algarismos com um indicativo, uma lista de listas de algarismos representando números de telefone, e seleciona os números que começam com o indicativo dado. Por exemplo:
   `indicativo [2,5,3] [[2,5,3,1,1,6,7,8,7],[2,1,3,4,4,8,0,2,3],[2,5,3,1,1,9,9,0,5]]`
   devolve `[[2,5,3,1,1,6,7,8,7],[2,5,3,1,1,9,9,0,5]]`.

2.
```
abrev :: [String] -> [String]
abrev lnoms = map conv lnoms
   where conv :: String -> String
         conv nom = let ns = (words nom)
                    in if (length ns) > 1
                          then  (head (head ns)):(". " ++ (last ns))
                          else nom
```
   que converte uma lista de nomes numa lista de abreviaturas desses nomes, da seguinte forma: [“João Carlos Mendes”, “Ana Carla Oliveira”] em [“J. Mendes”, “A. Oliveira”].

Tarefa 4 Uma forma de representar polinómios de uma variável é usar listas de pares (coeficiente, expoente)

```
type Pol = [(Float,Int)]
```

Note que o polinómio pode não estar simplificado. Por exemplo,

```
[(3.4,3), (2.0,4), (1.5,3), (7.1,5)]  :: Pol
```

representa o polinómio $3.4\,x^3 + 2\,x^4 + 1.5\,x^3 + 7.1\,x^5$.

1. Defina uma função para ordenar um polinómio por ordem crescente de grau.
2. Defina uma função para simplificar um polinómio.
3. Defina uma função para somar dois polinómios nesta representação.
4. Defina a função de cálculo do valor de um polinómio num ponto.
5. Defina uma função que dado um polinómio, calcule o seu grau.
6. Defina uma função que calcule a derivada de um polinómio.
7. Defina uma função que calcule o produto de dois polinómios.
8. Será que podemos ter nesta representação de polinómios, monómios com expoente negativo ? As funções que definiu contemplam estes casos ?