Aula Prática 4

Paradigmas de Programação I / Programação Funcional

ESI/MCC 1º ano (2005/2006)

O objectivo desta aula é a introdução à escrita de definições de funções recursivas.

1 Tipos

Existe em Haskell a possibilidade de definir abreviaturas para tipos.
Considere-se por exemplo que queremos definir funções de manipulação de uma lista telefónica. Para isso resolvemos que a informação de cada entrada na lista telefónica conterá o nome, nº de telefone e endereço de e-mail. Podemos então fazer as seguintes definições:

```
type Entrada = (String, String, String)
type LTelef = [Entrada]
```

A função que calcula os endereços de email conhecidos pode ser definida como

```
emails :: LTelef -> [String]
emails t = [em | (nome, num, em) <- t]
```

Note que, uma zez que o tipo `String` é por sua vez uma abreviatura de `[Char]`, o tipo da função `emails` acima é equivalente a

```
emails ::  ([([Char],[Char],[Char])] -> [[Char]]
```

Tarefa 1 Construa um módulo com as definições apresentadas acima e verifique (usando o comando `:t`) o tipo da função `emails`.

Tarefa 2 Defina uma função que, dada uma lista telefónica, produza a lista dos endereços de email das entradas cujos números de telefone são da rede fixa (prefixo '2'). Não se esqueça de explicitar o tipo desta função.

2 Listas

As listas podem ser definidas de uma forma recursiva como:

1. `[]` (a lista vazia) é uma lista;
2. Se `x :: a` (i.e., `x` é do tipo `a`) e `t :: [a]` (i.e., `t` é uma lista com elementos do tipo `a`) então `(x:t) :: [a]` (i.e., `x:t` é uma lista com elementos do tipo `a`).

Esta definição conduz a uma estratégia para definir funções sobre listas. Por exemplo, a função que calcula a soma dos elementos de uma lista pode ser definida como:

```
soma [] = 0
soma (h:t) = h + (soma t)
```

Tarefa 3 Use a estratégia sugerida acima para definir as seguintes funções.

1. A função que calcula o comprimento de uma lista.
2. A função que, dada uma lista e um elemento, o coloca no fim da lista.
3. A função que, dadas duas listas as *concatena*, i.e., calcula uma lista com os elementos da primeira lista seguidos dos da segunda lista (Sugestão: analise o que acontece para ambos os casos da primeira lista).

Para definirmos uma função que calcula a média dos elementos de uma lista de números podemos calcular a soma dos seus elementos e o comprimento da lista retornando depois o quociente entre estes dois valores.

```
media1 l = let s = sum l
               c = length l
           in s / (fromIntegral c)
```

Esta solução corresponde a percorrer a lista 2 vezes.

Tarefa 4

1. Defina uma função que, dada uma lista, calcula um par contendo o comprimento da lista e a soma dos seus elementos, percorrendo a lista uma única vez.
2. Usando a função anterior defina uma função que calcula a média dos elementos de uma lista.
3. Use o comando `:set +s` para comparar as *performances* das duas soluções (Use listas com muitos elementos para fazer os seus testes, e.g., `[1..10000000]`).

Há no entanto funções em que é mais difícil evitar estas múltiplas travessias da lista.

Tarefa 5

1. Defina uma função que, dada uma lista, a divida em duas (retornando um par de listas) com o mesmo número de elementos (isto, é claro, se a lista original tiver um número par de elementos; no outro caso uma das listas terá mais um elemento).
2. Defina uma função que, dada uma lista e um valor, retorne um par de listas em que a primeira contem todos os elementos da lista inferiores a esse valor e a segunda lista contem todos os outros elementos.
3. Defina uma função que, dada uma lista de números, retorne a lista com os elementos que são superiores à média.