Princípios de Programação

Exercícios

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

2019/2020

Mais sobre entrada e saída

Tópicos endereçados neste capítulo: Tratamento de ficheiros, manipulação da linha de comandos, números aleatórios.

1. Escreva uma função

   ```
   toFile :: Show a => FilePath -> [a] -> IO ()
   ```

   que escreva uma lista de elementos (de tipo pertencente à classe **Show**) num ficheiro, colocando um elemento por linha.

2. Escreva uma função

   ```
   fromFile :: Read a => FilePath -> IO [a]
   ```

   que leia uma lista de elementos (de tipo pertencente à classe **Read**) de um ficheiro. Cada elemento ocupa uma linha distinta no ficheiro.

3. Escreva uma função

   ```
   sumInts :: FilePath -> IO Int
   ```

   que calcule a soma dos inteiros contidos num dado ficheiro. Cada inteiro ocupa uma linha distinta no ficheiro.

4. Escreva uma função

   ```
   mergeFiles :: (Read a, Ord a) => FilePath -> FilePath
       -> IO [a]
   ```

que, dados dois ficheiros *ordenados*, produza a lista ordenada de todos os elementos constantes nos ficheiros. Cada elemento ocupa uma linha distinta no ficheiro. Considere dada uma função `merge :: Ord a =>[a] -> [a] -> [a]` que, dadas duas listas ordenadas, produz uma nova lista, também ordenada, com os elementos das duas listas.

5. Considere a seguinte assinatura.

```
filterFiles :: (String -> Bool) -> FilePath ->
   FilePath -> IO()
```

   (a) Escreva uma função que leia o conteúdo de um ficheiro, filtre as suas linhas de encontro a um predicado dado, e finalmente escreva o resultado num segundo ficheiro, linha a linha.

   (b) Escreva uma função

```
filterPrefix :: String -> FilePath -> FilePath ->
    IO ()
```

   que escreva num ficheiro as entradas que comecem com um dado prefixo. Considere dada uma função `isPrefix :: Eq a =>[a] -> [a] -> Bool`.

   (c) Mesmo exercício mas agora os três parâmetros são lidos da linha de comandos.

6. Escreva um programa `wc` que leia o nome de um ficheiro da linha de comandos e que escreva no `stdout` o número de linhas, de palavras e de carateres constantes no ficheiro. Por exemplo:

```
$ ghc --make wc.hs
[1 of 1] Compiling Main             ( wc.hs, wc.o )
Linking wc ...
$ ./wc wc.hs
        15        102        538        wc.hs
```

   Nota: `wc` é um comando Unix; faça `man wc` para aprender um pouco mais sobre o funcionamento e as opções do programa.

7. Escreva um programa que leia um menu de um ficheiro e que imprima no `stdout` as entradas do menu numeradas. Exemplo:

```
$ cat menu.txt
Bacalhau à Gomes Sá
Ensopado de borrego
$ ./menu
1 - Bacalhau à Gomes Sá
2 - Ensopado de borrego
```

(a) Comece por escrever uma função `linhasComNumeros :: [String] -> [String]` que coloque um número à esquerda de cada linha. Utilize a função `zipWith :: (a -> b -> c)-> [a] -> [b] -> [c]`.

(b) Escreva o programa utilizando a função `sequence :: [IO a] -> IO [a]`.

(c) Resolva agora o problema utilizando a função `mapM :: (a -> IO b)-> [a] -> IO [b]`.

8. Escreva uma função `rand :: Int -> Int` que devolva um número pseudo-aleatório entre `0` e `n-1`.

   Sugestão: utilize as funções **`randomR`** e **`mkStdGen`**. Espera obter um número diferente de cada vez que chamar a função com o mesmo `n`? Porquê?

9. Considere que cada carta de um baralho é representada por um número inteiro entre 1 e 52:

```
type Carta = Int
type Mao = [Carta]
```

   (a) Utilizando a função **`mkStdGen`**, escreva uma função que devolva uma mão de cartas de jogar, composta por um dado número de cartas.

   ```
   mao :: Int -> Mao
   ```

   (b) Mesmo exercício mas agora utilizando a função **`getStdGen`**.

   ```
   mao' :: Int -> IO Mao
   ```

10. O método de permutação por ordenação (*permute by sorting*) atribui a cada elemento `a[i]` de uma lista `a` uma prioridade `p[i]` aleatória. Depois ordena os elementos de `a` de acordo com as prioridades `p`. Por exemplo, se a lista inicial for `[1,2,3,4]` e escolhermos prioridades aleatórias `[36,3,62,19]` então produzimos uma lista `[2,4,1,3]`.

    Escreva uma função obtenha uma permutação de uma lista: `permutar :: Ord a =>[a] -> [a]` utilizando a técnica de permutação por ordenação. Sugestão: para uma lista de n elementos escolha prioridades entre 1 e n^3 de modo a minimizar a probabilidade de haver elementos repetidos em `p`. Analise a complexidade assintótica, $\mathcal{O}$, da sua função.

11. Um outro método para gerar uma permutação aleatória de uma lista dá pelo nome de *randomization in place*. Neste método, cada elemento `a[i]` de uma dada lista `a` é trocado pelo elemento na posição `j`, onde `j` é um

número aleatório entre `i` e **`length`** `a - 1`. Deste modo, cada elemento `a[i]` não é alterado depois da iteração `i`. Analise a complexidade assintótica, $\mathcal{O}$, da sua função.

12. Considere os seguintes tipos de dados:

```
type Numero = Int
data Naipe = Copas | Ouros | Paus | Espadas deriving
    (Eq, Ord, Show, Enum)
type Carta = (Numero, Naipe)
type Baralho = [Carta]
```

    (a) Escreva uma expressão `baralho52 :: Baralho` que represente um baralho convencional, com 52 cartas. Sugestão: utilize uma lista em compreensão.

    (b) Aplique a técnica de um dos exercícios anteriores para baralhar um `Baralho`.