Princípios de Programação

Exercícios

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

2019/2020

Entrada e saída

1. Escreva uma função `writePrimes :: [Int] -> IO ()` que escreve no `stdout` números primos a partir de uma lista de números naturais. Os números constantes na lista representam as ordens dos números primos. Por exemplo:

   ```
   ghci> writePrimes [7,78,1453,0]
   7th prime is 19
   78th prime is 401
   1453th prime is 12149
   0th prime is 2
   ```

   Para gerar a lista de todos os números primos utilize a seguinte função.

   ```
   primes :: [Integer]
   primes = sieve [2..]
       where sieve (p:xs) =
               p : sieve [x | x <- xs, x `mod` p > 0]
   ```

2. Considere dada a seguinte função.

   ```
   palindrome:: String -> Bool
   palindrome xs = xs == reverse xs
   ```

   (a) Escreva um programa que lê uma linha do `stdin` e escreve no `stdout` "Sim" ou "Não" conforme a frase é ou não um palíndromo.

(b) Escreva um programa que lê continuamente uma linha e que escreve "Sim" ou "Não" conforme a frase é ou não um palíndromo. O programa termina com a introdução de uma linha vazia.

(c) Mesmo exercício mas o programa termina com o caracter de fim de ficheiro.[1] Utilize a função
`interact :: (String -> String)-> IO ()`.

3. Escreva um programa que classifique cada número contido numa lista com "Par" ou "Impar". A classificação deverá aparecer no `stdout`. Exemplo:

```
ghci> showParity [1..4]
Impar
Par
Impar
Par
```

(a) Comece por escrever uma função `printEven :: Int -> IO()` que escreva no `stdout` "Par" ou "Impar".

(b) Escreva a função `showParity :: [Int] -> IO()` recursivamente.

(c) Mesmo problema utilizando a função
`mapM_ :: (a -> IO b)-> [a] -> IO()`.

4. Escreva um programa que comece por ler um inteiro positivo, n, e que depois leia n números inteiros e apresente a sua soma. Exemplo:

```
$ ./somar
Quantos números? 5
1
3
5
7
9
A soma é 25
```

Depois de tentar uma versão recursiva, escreva uma variante que utilize a função

```
sequence :: [IO a] -> IO [a]
```

que executa uma lista de acções e retorna a lista com os valores resultantes.

[1] Em Unix: `Ctrl-D`. Pode também utilizar `Ctrl-C` para terminar o processo (Unix e Windows).

5. Implemente o seguinte jogo: o utilizador pensa num número entre 1 e um dado valor. Usando busca binária, e até acertar, o programa sugere um número e o utilizador responde se esse número é <, > ou = ao número em que pensou. No fim o programa indica o número de tentativas que foram necessárias para adivinhar. Implemente o jogo utilizando uma função `guess :: Int -> IO ()`. Exemplo:

```
ghci> guess 300
150? <
75? <
37? >
56? <
46? =
Sucesso apos 5 tentativas
```

6. Considere o jogo da forca: o programa pede ao primeiro jogador uma palavra. O segundo jogador tenta adivinhar a palavra, letra a cada vez. Exemplo:

```
$ ghc --make forca.hs
[1 of 1] Compiling Main         ( forca.hs, forca.o )
Linking forca ...
$ ./forca
Primeiro Jogador, pense numa palavra:
-----
Segundo Jogador, tente adivinhar:
Tentativa 1: a
-a-a-
Tentativa 2: p
-a-a-
Tentativa 3: t
-ata-
Tentativa 4: s
-ata-
Tentativa 5: n
nata-
Tentativa 6: l
natal
Acertou!
```

   Sugestão: afim de separar o mais possível computações puras da entrada/saída construa uma função

   ```
   jogar :: String -> String -> Char -> Maybe String
   ```

   que, dada a palavra segredo (`natal`, por exemplo), a palavra já descoberta (`-a-a-`, por exemplo) e um caracter (`t`, por exemplo),

devolve `Nothing` se o caracter completa a palavra já descoberta, ou `Just novaPalavra` com a nova palavra parcial (`-ata-`).

Para evitar mostrar caracteres no ecrã enquanto lê o segredo pode utilizar a seguinte função (sucesso não garantido em Windows).

```
import System.IO
pedeSegredo :: IO String
pedeSegredo = do
    hSetEcho stdin False
    palavra <- getLine
    hSetEcho stdin True
    putStrLn (replicate (length palavra) '-')
    return palavra
```

(a) Comece por escrever um programa no qual o segundo jogador dispõe de um número ilimitado de tentativas.

(b) Mesmo problema, mas agora o jogador dispõe de um máximo de seis tentativas (cabeça, tronco e quatro membros).