Programas interactivos

6.1 Escreva uma função `elefantes` :: $Int \rightarrow IO\ ()$ tal que, por exemplo, `elefantes` 5 imprime os seguintes versos:

> Se 2 elefantes incomodam muita gente,
> 3 elefantes incomodam muito mais!
> Se 3 elefantes incomodam muita gente,
> 4 elefantes incomodam muito mais!
> Se 4 elefantes incomodam muita gente,
> 5 elefantes incomodam muito mais!

Sugestões: defina uma função auxiliar recursiva para iterar de 2 até n (onde n é o argumento da função `elefantes`); pode usar `show` para converter um inteiro numa cadeia de caracteres.

6.2 Escreva um programa completo que reproduza a funcionalidade do utilitário `wc` de Unix: ler um ficheiro de texto da entrada-padrão e imprimir o número de linhas, número de palavras e de carateres. Exemplo:

```
$ echo "a maria tinha um cordeirinho" | wc
      1       5      29
```

Sugestão: pode usar as funções `words` e `lines` do prelúdio-padrão para partir o texto em palavras e linhas e a função `length` para contar comprimentos.

6.3 Escreva um programa completo que lê toda a entrada-padrão, parte em linhas e imprime cada linha invertida.

6.4 Escreva um programa completo que codifique a entrada-padrão usando a cifra de César com rotação de 13 posições.[1] Exemplo:

```
$ echo "a maria tinha um cordeirinho" | ./rot13
n znevn gvaun hz pbeqrvevaub
```

Note que a rotação de 13 posições é a sua própria função inversa, pelo que o mesmo programa serve para codificar e descodificar mensagens.

6.5 Considere o programa para o jogo *Life* apresentada na aula teórica 10.

(a) Modifique o programa para que no final das n gerações imprima o número de células ainda vivas do tabuleiro.

(b) Modifique o programa para durante a simulação verificar se a grelha fica vazia e, nesse caso, terminar imediatamente.

6.6 Pretende-se escrever um verificador de ortografia básico: um programa que lê uma lista de palavras válidas (o *dicionário*) e um texto e identifica palavras que não ocorrem no dicionário (possíveis erros de ortografia).

[1] Ver a aula teórica 5 e ainda o sítio `https://pt.wikipedia.org/wiki/ROT13`.

1. Leia o dicionário do ficheiro `/usr/share/dict/words` (que contém uma palavra por linha) e construa uma lista com as palavras.

```
readDict :: IO [String]
readDict = do txt <- readFile "/usr/share/dict/words"
              return (lines txt)
```

2. Leia o texto do utilizador e parta em linhas e palavras; imprima cada palavra separadamente e use uma cor diferente quando esta não ocorre na lista do dicionário.[2]

Note que este programa tem de pesquisar numa lista com muitas palavras (o dicionário); mais tarde veremos estruturas em árvore que permitem tornar uma pesquisa mais eficiente.

6.7 Escreva uma função interactiva `adivinha` :: *String* → *IO* () que implemente um jogo de advinha duma palavra secreta dada como argumento pelo utilizador; um outro jogador vai tentar adivinhá-la.

```
------
? a
-a-a-a
? e
Não ocorre
-a-a-a
? b
ba-a-a
? n
banana
Adivinhou em 4 tentativas
```

O programa deve mostrar a palavra, substituindo as letras desconhecidas por traços e pedir uma nova letra; todas as ocorrências dessa letra na palavra devem então ser reveladas. O jogo termina quando o jogador adivinha a palavra; o programa deve então imprimir o número de tentativas.

6.8 O jogo *Nim* desenrola-se com cinco filas de peças idênticas (representadas por estrelas), cujo estado inicial é o seguinte:

```
1: *****
2: ****
3: ***
4: **
5: *
```

Dois jogadores vão alternadamente retirar uma ou mais estrelas de uma das 5 filas; ganha o jogador que remover a última estrela ou grupo de estrelas.

Implemente um programa em Haskell que peça as jogadas de cada jogador e actualize o tabuleiro. Sugestão: represente o jogo como uma lista com o número de estrelas em cada fila; o estado inicial será então `[5,4,3,2,1]`.

[2] Veja o exemplo da aula 12 com códigos ANSI para mudar as cores no terminal e também `https://en.wikipedia.org/wiki/ANSI_escape_code`