

Listas por compreensão

3.1 Defina uma função `divprop :: Integer → [Integer]` usando uma lista em compreensão para calcular a lista de *divisores próprios* de um inteiro positivo (i.e. inferiores ao número dado). Exemplo: `divprop 10 = [1, 2, 5]`.

3.2 Um inteiro positivo n diz-se *perfeito* se for igual à soma dos seus divisores (excluindo o próprio n). Defina uma função `perfeitos :: Integer → [Integer]` que calcula a lista de todos os números perfeitos até um limite dado como argumento. Exemplo: `perfeitos 500 = [6, 28, 496]`. *Sugestão*: utilize a solução do exercício 3.1.

3.3 Um trio (x, y, z) de inteiros positivos diz-se *pitagórico* se $x^2 + y^2 = z^2$. Defina a função `pitagoricos :: Integer → [(Integer, Integer, Integer)]` que calcule todos os trios pitagóricos cujas componentes não ultrapassem o argumento. Por exemplo: `pitagoricos 10 = [(3, 4, 5), (4, 3, 5), (6, 8, 10), (8, 6, 10)]`.

3.4 Defina uma função `primo :: Integer → Bool` que testa primalidade: n é primo se tem exactamente dois divisores, a saber, 1 e n . *Sugestão*: utilize a função do exercício 3.1 para obter a lista dos divisores próprios.

3.5 Usando uma função `binom` da folha 1 que calcula coeficientes binomiais, escreva uma definição da função `pascal :: Integer → [[Integer]]` que calcula o triângulo de Pascal até à linha n :

$$\begin{array}{c}
 \binom{0}{0} \\
 \binom{1}{0} \quad \binom{1}{1} \\
 \vdots \\
 \binom{n}{0} \quad \dots \quad \binom{n}{k} \quad \dots \quad \binom{n}{n}
 \end{array}$$

3.6 (T) A *cifra de César* é um dos métodos mais simples para codificar um texto: cada letra é substituída pela que dista k posições à frente no alfabeto; se ultrapassar a letra Z, volta à letra A. Por exemplo, para $k = 3$, a substituição efectuada é

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C

e o texto “ATAQUE DE MADRUGADA” é transformado em “DWDTXH GH PDGUXJDGD”.

Escreva uma função `cifrar :: Int → String → String` para cifrar uma cadeia de caracteres usando um deslocamento dado. Note que `cifrar (-n)` é a função inversa de `cifrar n`, pelo que a mesma função pode servir para codificar e decodificar.

Definições recursivas

3.7 Escreva novas definições recursivas de funções equivalentes às do prelúdio de Haskell. Por exemplo: defina uma função `myand` equivalente a `and`, `myor` equivalente a `or`, etc.

- (a) `and :: [Bool] → Bool` — testar se todos os valores são `True`;
- (b) `or :: [Bool] → Bool` — testar se algum valor é `True`;
- (c) `concat :: [[a]] → [a]` — concatenar uma lista de listas;
- (d) `replicate :: Int → a → [a]` — produzir uma lista com n elementos iguais;
- (e) `(!!) :: [a] → Int → a` — selecionar o n -ésimo elemento numa lista;
- (f) `elem :: Eq a ⇒ a → [a] → Bool` — testar se um valor ocorre numa lista.

3.8 Mostre que as funções do prelúdio-padrão `concat`, `replicate` e `(!!)` podem também ser definidas sem recursão usando listas em compreensão.

3.9 Defina uma função `forte :: String → Bool` para verificar se uma palavra-passe dada numa cadeia de caracteres é forte segundo os seguintes critérios: deve ter 8 caracteres ou mais e pelo menos uma letra maiúscula, uma letra minúscula e um algarismo.

Sugestão: use a função `or :: [Bool] → Bool` e listas em compreensão.

3.10 Neste exercício pretende-se implementar um teste de primalidade mais eficiente do que o do exercício 3.4.

- (a) Escreva uma função `mindiv :: Int → Int` cujo resultado é o menor divisor próprio do argumento (i.e. o menor divisor superior a 1). Note que se $n = p \times q$, então p e q são ambos divisores de n ; se $p \geq \sqrt{n}$, então $q \leq \sqrt{n}$ pelo que o menor divisor será sempre $\leq \sqrt{n}$. Assim *não necessitamos de tentar candidatos a divisores superiores a \sqrt{n}* .
- (b) Utilize `mindiv` para definir um teste de primalidade mais eficiente do que o exercício 3.4: n é primo se $n > 1$ e o seu menor divisor próprio for igual a n .

3.11 A função `nub :: Eq a ⇒ [a] → [a]` do módulo `Data.List` elimina ocorrências de elementos repetidos numa lista (“nub” em inglês significa *essência*). Por exemplo: `nub “banana” = “ban”`.

Escreva uma definição recursiva para esta função. Sugestão: use uma lista em compreensão com uma guarda para eliminar elementos numa lista.

3.12 Escreva uma definição da função `intersperse :: a → [a] → [a]` do módulo `Data.List` que intercala um valor entre os elementos numa lista. Exemplo: `intersperse 'a' “banana” = “b-a-n-a-n-a”`.