Listas por compreensão

2.1 Usando uma lista em compreensão, escreva uma expressão para calcular a soma $1^2 + 2^2 + \cdots + 100^2$ dos quadrados dos inteiros de 1 a 100.

2.2 A constante matemática π pode ser aproximada usando expansão em *séries* (i.e. somas infinitas), como por exemplo:

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \cdots + \frac{(-1)^n}{2n+1} + \cdots$$

(a) Escreva uma função $aprox :: Int \to Double$ para aproximar π somando em n parcelas da série acima (onde n é o argumento da função).

(b) A série anterior converge muito lentamente, pelo são necessário muitos termos para obter uma boa aproximação; escreva uma outra função $aprox'$ usando a seguinte expansão para π^2:

$$\frac{\pi^2}{12} = 1 - \frac{1}{4} + \frac{1}{9} - \cdots + \frac{(-1)^k}{(k+1)^2} + \cdots$$

Compare os resultados obtidos somado 10, 100 e 1000 termos com a aproximação *pi* pré-definida no prelúdio-padrão.

2.3 Escreva uma função $dotprod :: [Float] \to [Float] \to Float$ para calcular o *produto interno* de dois vectores (representados como listas):

$$dotprod\ [x_1, \ldots, x_n]\ [y_1, \ldots, y_n] = x_1 * y_1 + \cdots + x_n * y_n = \textstyle\sum_{i=1}^{n} x_i * y_i$$

Sugestão: utilize a função $zip :: [a] \to [b] \to [(a, b)]$ do prelúdio-padrão para "emparelhar" duas listas.

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

2.5 Um inteiro positivo n diz-se *perfeito* se for igual à soma dos seus divisores (excluindo o próprio n). Defina uma função $perfeitos :: Integer \to [Integer]$ 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 2.4.

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

2.7 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 2.4 para obter a lista dos divisores próprios.

2.8 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:

$$\begin{array}{ccccc} & & \binom{0}{0} & & \\ & \binom{1}{0} & & \binom{1}{1} & \\ & \iddots & \vdots & \ddots & \\ \binom{n}{0} & \cdots & \binom{n}{k} & \cdots & \binom{n}{n} \end{array}$$

2.9 A *cifra de César* é um dos métodos mais simples para codificar um texto: cada letra é substituida 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 descodificar.

Definições recursivas

2.10 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 duma lista;

(f) *elem* :: *Eq* a ⇒ a → [a] → *Bool* — testar se um valor ocorre numa lista.

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

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

Sugestão: use a função $or :: [Bool] \to Bool$ e listas em compreensão.

2.13 Neste exercício pretende-se implementar um teste de primalidade mais eficiente do que o do exercício 2.7.

(a) Escreva uma função $mindiv :: Int \to Int$ 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 à* $\sqrt{n}$.

(b) Utilize *mindiv* para definir um teste de primalidade mais eficiente do que o exercício 2.7: n é primo se $n > 1$ e o seu menor divisor próprio for igual a n.

2.14 A função $nub :: Eq\ a \Rightarrow [a] \to [a]$ 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 duma lista.

2.15 Escreva uma definição da função $intersperse :: a \to [a] \to [a]$ do módulo *Data.List* que intercala um valor entre os elementos duma lista. Exemplo: *intersperse* '-' "banana" = "b-a-n-a-n-a".

2.16 Escreva uma definição da função $algarismos :: Int \to [Int]$ que obtém os algarismos decimais de um inteiro positivo. Exemplo: *algarismos* 12345 = $[1, 2, 3, 4, 5]$.

Sugestão: Pode obter o algarismo das unidades usando o resto da divisão por 10 e prosseguir recursivamente com o quociente da divisão. Começe por definir uma função auxiliar que obtem os algarismos pela ordem inversa, i.e. *algarismosRev* 12345 = $[5, 4, 3, 2, 1]$.

2.17 Escreva uma definição da função $toBits :: Int \to [Int]$ que obtém a representação em binário de um inteiro não-negativo. Exemplo: *toBits* 29 = $[1, 1, 1, 0, 1]$. Note que os digitos binários do resultado estão pela ordem do mais significativo para o menos significativo.

Sugestão: O problema é semelhante ao exercício anterior, mas efetuando divisões por 2 em vez de 10.

2.18 Escreva uma definição função da função $fromBits :: [Int] \to Int$ que faz a transformação inversa da anterior, ou seja, converte digitos em binário para o inteiro não-negativo correspondente.

2.19 O algoritmo de Euclides para calcular o máximo divisor comum de dois inteiros a, b pode ser expresso de forma recursiva:

$$mdc(a,b) = \begin{cases} a\,, & \text{se } b = 0 \\ mdc(b, a \bmod b)\,, & \text{caso contrário} \end{cases}$$

Traduza esta definição recursiva para uma função $mdc :: Integer \to Integer \to Integer$.

2.20 Ordenação de listas pelo **método de inserção**.

(a) Escreva definição recursiva da função $insert :: Ord\ a \Rightarrow a \to [a] \to [a]$ da biblioteca *List* para inserir um elemento numa lista ordenada na posição correcta de forma a manter a ordenação. Exemplo: $insert\ 2\ [0,1,3,5] = [0,1,2,3,5]$.

(b) Usando a função *insert*, escreva uma definição também recursiva da função $isort :: Ord\ a \Rightarrow [a] \to [a]$ que implementa *ordenação pelo método de inserção*:

- a lista vazia já está ordenada;
- para ordenar uma lista não vazia, recursivamente ordenamos a cauda e inserimos a cabeça na posição correcta.

2.21 Ordenação de listas pelo **método de seleção**.

(a) Escreva definição recursiva da função $minimum :: Ord\ a \Rightarrow [a] \to a$ do prelúdio-padrão que calcula o menor valor duma lista não-vazia. Exemplo: $minimum\ [5,1,2,1,3] = 1$.

(b) Escreva uma definição recursiva da função $delete :: Eq\ a \Rightarrow a \to [a] \to [a]$ da biblioteca *List* que remove a primeira ocorrência dum valor numa lista. Exemplo: $delete\ 1\ [5,1,2,1,3] = [5,2,1,3]$.

(c) Usando as funções anteriores, escreva uma definição recursiva da função $ssort :: Ord\ a \Rightarrow [a] \to [a]$ que implementa *ordenação pelo método de seleção*:

- a lista vazia já está ordenada;
- para ordenar uma lista não vazia, colocamos à cabeça o menor elemento m e recursivamente ordenamos a cauda sem o elemento m.

2.22 Ordenação de listas pelo método **merge sort**.

(a) Escreva uma definição recursiva da função $merge :: Ord\ a \Rightarrow [a] \to [a] \to [a]$ para juntar duas listas ordenadas numa só mantendo a ordenação. Exemplo: $merge\ [3,5,7]\ [1,2,4,6] = [1,2,3,4,5,6,7]$.

(b) Usando a função *merge*, escreva uma definição recursiva da função $msort :: Ord\ a \Rightarrow [a] \to [a]$ que implementa o método *merge sort*:

- uma lista vazia ou com um só elemento já está ordenada;

- para ordenar uma lista com dois ou mais elementos, partimos em duas metades, recursivamente ordenamos as duas parte e juntamos os resultados usando *merge*.

Sugestão: começe por definir uma função $metades :: [a] \to ([a],[a])$ para partir uma lista em duas metades (ver a Folha 1).

2.23 Podemos fazer operações com polinómios representados pelas listas de coeficientes. Por exemplo, podemos representar dois polinómios $P = 1+3X-X^2$ e $Q = 3+2X^2$ pelas listas $[1,3,-1]$ e $[3,0,2]$, respectivamente.

(a) Escreva uma definição da função $addPoly :: [Int] \to [Int] \to [Int]$ que calcula a soma de dois polinómios. Exemplo: $addPoly\ [1,3,-1]\ [3,0,2] = [4,3,1]$ é a lista dos coeficientes de $P + Q$.

(b) Escreva uma definição da função $multPoly :: [Int] \to [Int] \to [Int]$ que calcula o produto de dois polinómios. Exemplo: $multPoly\ [1,3,-1]\ [3,0,2] = [3,9,-1,6,-2]$ é a lista de coeficientes de $P \times Q$.