1ª Lista de Exercícios – 2024.2

Definição de funções simples

1. Considere as seguintes definições de funções:

`inc` $x = x + 1$	`dobro` $x = x + x$
`quadrado` $x = x * x$	`media` $x\ y = (x + y)/2$

Efetuando reduções passo-a-passo, calcule os valores das expressões seguintes:

(a) `inc` (`quadrado` 5)

(b) `quadrado` (`inc` 5)

(c) `media` (`dobro` 3) (`inc` 5)

2. Num triângulo, verifica-se sempre a seguinte condição: a medida de um lado qualquer é menor que a soma dos outros dois. Complete a definição de uma função `triangulo` $a\ b\ c = \cdots$ que testa esta condição; o resultado deve ser um valor boleano.

3. Escreva uma definição em Haskell duma função para calcular a área A de um triângulo de lados a, b, c usando a fórmula de Heron:

$$A = \sqrt{s(s-a)(s-b)(s-c)},$$

onde s é metade do perímetro do triângulo.

4. Usando as funções do prelúdio-padrão (`head`, `tail`, `length`, `take`, `drop`, $++$, `reverse`, $!!$, `sum` e `product`), escreva uma função `metades` que divide uma lista de comprimento par em duas com metade do comprimento. Exemplo: `metades` $[1,2,3,4,5,6,7,8] = ([1,2,3,4],[5,6,7,8])$. O que acontece se a lista tiver comprimento ímpar ?

5. (a) Mostre que a função `last` (que seleciona o último elemento de uma lista) pode ser escrita como composição das funções do prelúdio-padrão apresentadas. Apresente *duas* definições distintas.

(b) Mostre que a função `init` (que remove o último elemento duma lista) pode ser definida analogamente de duas formas diferentes.

6. (a) Escreva uma função `binom` com dois argumentos que calcule o *coeficiente binomial*:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

Sugestão: $n!$ pode ser expresso por `product` $[1..n]$.

(b) Para quaisquer listas de números xs e ys, temos que `product` $(xs{+\!\!+}ys) =$ `product` $xs *$ `product` ys. Use esta propriedade para re-escrever a definição de forma mais eficiente, eliminando fatores comuns entre o numerador e denominador.

7. Considere as definições das funções `max` e `min` do prelúdio-padrão que calculam respectivamente o máximo e o mínimo de dois números:

$$\texttt{max}\ x\ y = \text{if } x \geq y \text{ then } x \text{ else } y$$
$$\texttt{min}\ x\ y = \text{if } x \leq y \text{ then } x \text{ else } y$$

(a) Usando condicionais aninhadas, escreva definições de duas funções `max3` e `min3` para calcular, respectivamente, o máximo e o mínimo de três números.

(b) Re-escreva as funções acima de forma a usar *apenas* a composição de `max` e `min` (i.e. sem condicionais ou guardas).

8. Implemente em Haskell as seguintes funções:

(a) `maxOccurs :: Integer → Integer → (Integer, Integer)` que retorna o máximo de dois inteiros e o número de vezes que ocorre.

(b) `orderTriple :: (Integer, Integer, Integer) → (Integer, Integer, Integer)` que ordena o triplo por ordem ascendente.

9. Escreva duas definições da função `classifica` :: *Int* →`String`, usando expressões condicionais e guardas respectivamente, que mapeia notas entre 0 e 20 para uma classificação qualitativa:

≤ 9	reprovado
10–12	suficiente
13–15	bom
16–18	muito bom
19–20	muito bom com distinção

10. Escreva uma definição da função lógica ou-exclusivo `xor` :: *Bool* → *Bool* → *Bool* usando múltiplas equações com padrões.

11. Pretende-se implementar uma função `safetail` :: [a] → [a] que extende a função `tail` do prelúdio resultando na lista vazia quando o argumento é a lista vazia (em vez de um erro). Escreva três definições diferentes usando condicionais, equações com guardas e padrões.

12. Escreva duas definições da função `curta` :: [a] → *Bool* para testar se uma lista tem zero, um ou dois elementos, usando:

(a) a função `length` do prelúdio-padrão;

(b) múltiplas equações e padrões.

13. Defina uma função `textual` :: *Int* → *String* para converter um número positivo inferior a um milhão para a designação textual em português. Alguns exemplos:

`textual` 21 = "vinte e um"
`textual` 1234 = "mil duzentos e trinta e quatro"
`textual` 123456 = "cento e vinte e três mil quatrocentos e cinquenta e seis"

Sugestão: Começe por definir funções auxiliares para converter para texto número inferiores a 100 e 1000.

Tipos e classes

14. Indique tipos admissíveis para os seguintes valores.

(a) $[\text{'a'}, \text{'b'}, \text{'c'}]$
(b) $(\text{'a'}, \text{'b'}, \text{'c'})$
(c) $[(\texttt{False}, \text{'0'}), (\texttt{True}, \text{'1'})]$
(d) $([\texttt{False}, \texttt{True}], [\text{'0'}, \text{'1'}])$
(e) $[\texttt{tail}, \texttt{init}, \texttt{reverse}]$
(f) $[\texttt{id}, \texttt{not}]$

15. Indique tipos possíveis para `f` e `g` tal que:

1. `f` $(2, 5)$ tenha tipo `Int`;
2. `f` $(\texttt{g}\ 5)$ tenha tipo `Int`;
3. $(\texttt{f}\ \texttt{g})\ 5$ tenha tipo `Int`;
4. `f g` $[1, 2, 3]$ tenha tipo $[\texttt{Int}]$;
5. `f` $(2, 5)$ tenha tipo $[\texttt{Int} \to \texttt{Int}]$.

16. Diga qual o tipo mais geral de `f` e `g` tal que `head (f g) 5` tenha o tipo $[\texttt{Int}]$.

17. Indique o tipo mais geral para as seguintes definições; tenha o cuidado de incluir restrições de classes no caso de operações com sobrecarga.

(a) $\texttt{segundo}\ xs = \texttt{head}\ (\texttt{tail}\ xs)$
(b) $\texttt{trocar}\ (x, y) = (y, x)$
(c) $\texttt{par}\ x\ y = (x, y)$
(d) $\texttt{dobro}\ x = 2 * x$
(e) $\texttt{metade}\ x = x/2$
(f) $\texttt{minuscula}\ x = x \geq \text{'a'}\ \&\&\ x \leq \text{'z'}$
(g) $\texttt{intervalo}\ x\ a\ b = x \geq a\ \&\&\ x \leq b$
(h) $\texttt{palindromo}\ xs = \texttt{reverse}\ xs == xs$
(i) $\texttt{twice}\ f\ x = f\ (f\ x)$

18. Indique exemplos de tipos concretos admissíveis e os tipos mais gerais para cada uma das definições dos exercícios 1 e 2. Tenha o cuidado de incluir apenas as restrições de classe estritamente necessárias.

19. Dê exemplo de funções cuja definição é compatível com os tipos seguintes:

1. $\texttt{Int} \to (\texttt{Int} \to \texttt{Int}) \to \texttt{Int}$
2. $\texttt{Char} \to \texttt{Bool} \to \texttt{Bool}$
3. $(\texttt{Char} \to \texttt{Char} \to \texttt{Int}) \to \texttt{Char} \to \texttt{Int}$
4. $\texttt{Eq a} \Rightarrow \texttt{a} \to [\texttt{a}] \to \texttt{Bool}$
5. $\texttt{Eq a} \Rightarrow \texttt{a} \to [\texttt{a}] \to [\texttt{a}]$
6. $\texttt{Ord a} \Rightarrow \texttt{a} \to \texttt{a} \to \texttt{a}$

20. Diga se a função $\texttt{f} :: (\texttt{a}, [\texttt{a}]) \to \texttt{Bool}$, pode ser aplicada aos argumentos $(2, [3])$, $(2, [])$ e $(2, [\texttt{True}])$. Nos casos afirmativos quais os tipos dos resultados?

21. Repita o exercício anterior para a função $\texttt{f} :: (\texttt{a}, [\texttt{a}]) \to \texttt{a}$.

Listas em compreensão

22. 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.

23. 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` :: $Int \to \mathrm{Double}$ para aproximar π somando n parcelas da série acima (onde n é o argumento da função).

(b) A série anterior converge muito lentamente, pois são necessários 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 somando 10, 100 e 1000 termos com a aproximação `pi` pré-definida no prelúdio-padrão.

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

25. Um inteiro positivo n diz-se *perfeito* se for igual à soma dos seus divisores próprios. Defina uma função `perfeitos` :: $Int \to [Int]$ que calcula a lista de todos os números perfeitos até um limite dado como argumento. Exemplo: `perfeitos` $500 = [6, 28, 496]$.

26. Defina uma função `primo` :: $Int \to Bool$ que testa primalidade: n é primo se tem exatamente dois divisores, a saber, seus *divisores triviais* 1 e n. *Sugestão*: utilize a função do exercício 24 para obter a lista dos divisores próprios.

27. Usando uma função `binom` que calcula o coeficiente binomal, escreva uma definição da função `pascal` :: $Int \to [[Int]]$ que calcula as primeiras linhas triângulo de Pascal. O *triângulo de Pascal* é constituido pelos valores $\binom{n}{k}$ das combinações de n em k em que n é a linha e k é a coluna.

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

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

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

29. Uma tripla (x, y, z) de inteiros positivos diz-se *pitagórico* se $x^2 + y^2 = z^2$. Defina a função `pitagoricos` :: $Int \to [(Int, Int, Int)]$ que calcule todos os trios pitagóricos cujas componentes não ultrapassem o argumento.
(Exemplo: `pitagoricos` $10 = [(3,4,5), (4,3,5), (6,8,10), (8,6,10)]$)

30. Defina uma função `forte` :: $String \to Bool$ para verificar se uma senha dada por uma cadeia de carateres é "forte", ou seja: tem 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.

Definições recursivas e processamento de listas

31. Defina uma função recursiva em Haskell que dado n calcula 2^n (sem recorrer ao operador de exponenciação).

32. Sem consultar as definições na especificação do prelúdio de Haskell, escreva definições recursivas das seguintes funções:

(a) `and` :: $[Bool] \to Bool$ — testar se todos os valores são `True`;

(b) `or` :: $[Bool] \to Bool$ — testar se algum valor é `True`;

(c) `concat` :: $[[a]] \to [a]$ — concatenar uma lista de listas;

(d) `replicate` :: $Int \to a \to [a]$ — produzir uma lista com n elementos iguais;

(e) (!!) :: $[a] \to Int \to a$ — selecionar o n-ésimo elemento duma lista;

(f) `elem` :: Eq $a \Rightarrow a \to [a] \to Bool$ — testar se um valor ocorre numa lista.

Nota: não deve usar os mesmos nomes, para não colidir com as definições no Prelude.

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

34. A raiz quadrada inteira de um número positivo n é o maior inteiro cujo quadrado é menor ou igual a n. Por exemplo, para 15 e 16, os resultados são, respectivamente 3 e 4.

(a) Defina uma função recursiva `leastSquare` n que calcula o menor inteiro k tal que $k * k \geq n$.

(b) Defina uma função não recursiva `isqrt` n que calcula a raiz quadrada inteira de n.

35. Defina em Haskell:

(a) a função `fatorial` (recursivamente).

(b) a função `rangeProduct` que calcula

$$a * (a + 1) * ... * (b - 1) * b$$

(c) a função `fatorial` usando a função `rangeProduct`.

36. Defina uma função em Haskell para calcular o máximo divisor comum de dois inteiros positivos segundo a seguinte definição:

$$\texttt{mdc}(a, b) = \begin{cases} a & b = 0 \\ \texttt{mdc}(b, a \ `mod`\ b) & \text{otherwise} \end{cases}$$

37. A função `nub` :: Eq $a \Rightarrow [a] \to [a]$ do módulo *Data.List* elimina ocorrências de elementos repetidos numa lista. 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.

38. 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".

39. Defina uma função `maxFun :: (Integer → Integer) → Integer → Integer`, que tendo como argumentos uma função `f :: Integer → Integer` e um inteiro `n`, retorne o valor máximo entre *f 0, f 1,..., f n*.

40. Defina uma função `anyZero :: (Integer → Integer) → Integer → Bool` que tendo como argumentos uma função `f :: Integer → Integer` e um inteiro `n` retorna *True* se algum valor de *f 0, f 1,..., f n* é igual a zero e *False* caso contrário.

41. Defina uma função `sumFun :: (Integer → Integer) → Integer → Integer`, que tendo como argumentos uma função `f :: Integer → Integer` e um inteiro `n` retorna *(f 0) + (f 1) + ... + (f n)*.

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

(a) Defina recursivamente a função `insert :: Ord a ⇒ a → [a] → [a]` da biblioteca *List*, para inserir um elemento numa lista ordenada na posição correta 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 ⇒ [a] → [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 o valor da cabeça na posição correta.

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

(a) Defina recursivamente a função `minimum :: Ord a ⇒ [a] → a` (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 ⇒ a → [a] → [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 ⇒ [a] → [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.

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

(a) Defina recursivamente a função `merge :: Ord a ⇒ [a] → [a] → [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 ⇒ [a] → [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 partes e juntamos os resultados usando `merge`.

Sugestão: começe por definir uma função `metades :: [a] → ([a], [a])` para partir uma lista em duas metades.

45. Escreva uma definição da função `bits` :: $Int \rightarrow [[Bool]]$ que obtém todas as sequências de boleanos do comprimento dado (a ordem das sequências não é importante). Exemplo: `bits` $2 =$ `[[False,False],[True,False],[False,True],[True,True]]`.

Sugestão: tente exprimir a função por recorrência sobre o comprimento.

46. Escreva uma função `permutations` :: $[a] \rightarrow [[a]]$ para obter a lista com todas as permutações dos elementos duma lista (a ordem das permutações não é importante). Assim, se xs tem comprimento n, então `permutations` xs tem comprimento $n!$. Exemplo: `permutations` $[1,2,3] = [[1,2,3],[2,1,3],[2,3,1],[1,3,2],[3,1,2],[3,2,1]]$.