Funções de ordem superior

5.1 Considere a seguinte definição de uma função que lista todos os divisores positivos de um inteiro:

```
divisores n = [d | d<-[1..n], n`mod`d == 0]
```

Re-escreva a definição usando a função `filter`.

5.2 Escreva uma definição da função `primo` :: *Integer* → *Bool* que testa se um inteiro n é primo usando a seguinte condição: n deve ser maior que 1 e nenhum dos números entre 2 e $\lfloor\sqrt{n}\rfloor$ deve ser divisor de n.

Sugestão: Utilize uma das funções de ordem superior `any` ou `all` para exprimir a condição "nenhum dos números...". Para calcular a parte inteira da raiz quadrada pode usar `floor (sqrt (fromIntegral n))`.

5.3 Escreva definições alternativas das seguintes funções do prelúdio-padrão usando as funções de ordem superior indicadas.

(a) $(+\!\!+) :: [a] \to [a] \to [a]$, usando `foldr`;

(b) `concat` $:: [[a]] \to [a]$, usando `foldr`;

(c) `reverse` $:: [a] \to [a]$, usando `foldr`;

(d) `reverse` $:: [a] \to [a]$, usando `foldl`;

(e) `elem` $:: Eq\ a \Rightarrow a \to [a] \to Bool$, usando `any`.

5.4 Usando `foldl`, defina uma função `fromBits` $:: [Int] \to Int$ que converte uma lista de algarismos binários no inteiro correspondente (ver Folha 4). Exemplo: `fromBits` $[1, 1, 0, 1] = 2^3 + 2^2 + 2^0 = 13$.

5.5 A função `zipWith` $:: (a \to b \to c) \to [a] \to [b] \to [c]$ do prelúdio-padrão é uma variante de `zip` cujo primeiro argumento é uma função usada para combinar cada par de elementos. Podemos definir *zipWith* usando uma lista em compreensão:

$$\texttt{zipWith}\ f\ xs\ ys = [f\ x\ y \mid (x, y) \leftarrow \texttt{zip}\ xs\ ys]$$

Escreva uma definição recursiva de `zipWith`.

5.6 Mostre que pode definir função `isort` $:: Ord\ a \Rightarrow [a] \to [a]$ para ordenar uma lista pelo método de inserção (ver a Folha 4) usando `foldr` e `insert`.

5.7 Escreva uma definição da função `palavras` :: *String* → [*String*] que decompõe uma cadeia de texto em palavras delimitadas por um ou mais espaços. Exemplos:

```
palavras "Abra- ca-   dabra!" == ["Abra-", "ca-", "dabra!"]
palavras " cadabra! "         == ["cadabra!"]
palavras "     "              == []
```

Sugestão: escreva uma definição recursiva; poderá usar as função `takeWhile` e `dropWhile` para obter cada palavra.

5.8 A função do prelúdio *scanl* é uma variante do *foldl* que produz a lista com os valores acumulados:

$$\texttt{scanl}\ f\ z\ [x_1,\, x_2,\, \ldots] = [z,\, f\ z\ x_1,\, f\ (f\ z\ x_1)\ x_2,\, \ldots]$$

Por exemplo:

$$\texttt{scanl}\ (+)\ 0\ [1,2,3] = [0,\, 0+1,\, 0+1+2,\, 0+1+2+3] = [0,1,3,6]$$

Em particular, para listas finitas xs temos que `last` (`scanl` $f\ z\ xs$) = `foldl` $f\ z\ xs$.

Escreva uma definição recursiva de *scanl*; deve usar outro nome para evitar colidir com a definição do prelúdio.

Listas infinitas

5.9 Considere duas séries (i.e. somas infinitas) que convergem para π:

$$\pi = \frac{4}{1} - \frac{4}{3} + \frac{4}{5} - \frac{4}{7} + \cdots$$
$$\pi = 3 + \frac{4}{2\times 3\times 4} - \frac{4}{4\times 5\times 6} + \frac{4}{6\times 7\times 8} - \cdots$$

Escreva duas funções `aproxPi1, aproxPi2 :: Int -> Double` que calculam um valor aproximado de π usando o número de parcelas dado como argumento; investigue qual das séries converge mais depresssa para π.

Sugestão: Construa listas infinitas para os numeradores e denominadores dos termos separadamente e combine-as usando `zip` ou `zipWith`.

5.10 Neste exercício pretende-se definir o triângulo de Pascal completo como uma lista infinita de listas `pascal` :: [[*Integer*]] com as linhas do triângulo.

(a) Escreva uma definição de `pascal` usando a função `binom` da Folha 1. Lembre que a linha n e coluna k do triângulo de Pascal é igual a `binom` $n\ k$, para quaisquer n e k tais que $n > 0$ e $0 \leq k \leq n$.

(b) Escreva outra definição que evite o cálculo de factoriais usando as seguintes propriedades de coeficientes binomiais:

$$\binom{n}{0} = \binom{n}{n} = 1 \qquad\qquad \binom{n+1}{k+1} = \binom{n}{k} + \binom{n}{k+1} \quad (\text{se } n > k)$$

5.11 A conjetura de Goldbach afirma que qualquer inteiro par maior que 2 se pode obter como a soma de dois primos. Por exemplo: 10 é um par maior de 2 e podemos escrever $10 = 3+7$ ou $10 = 5+5$ (a decomposição não tem de ser única). Esta conjetura não foi ainda provada[1] mas já foi testada experimentalmente para números muito altos.

Pretende-se que defina uma função `goldbach` :: *Integer* $\to$ (*Integer*, *Integer*) que, dado um inteiro par n maior que 2, encontra uma "testemunha" da conjetura de Goldbach, ou seja, um par (p, p') de números primos tal que $n = p + p'$.

Sugestão: Utilize a definição da lista infinita de primos apresentada na aula teórica.

5.12 Os *números de Hamming* têm 2, 3 e 5 como únicos fatores primos, ou seja, são números da forma $2^i \times 3^j \times 5^k$ onde i, j, k são inteiros não negativos. Podemos usar uma expressão em compreensão em Haskell para gerar alguns números de Hamming:

```
> [2^i*3^j*5^k | i<-[0..2], j<-[0..2], k<-[0..2]]
[1,5,25,3,15,75,9,45,225,2,10,50,6,30,150,18,90,450,4,20,100,
12,60,300,36,180,900]
```

No entanto, a seguinte tentativa **não** produz a lista de *todos* os números de Hamming (porquê?):

```
[2^i*3^j*5^k | i<-[0..], j<-[0..], k<-[0..]]
```

Pretende-se que escreva uma expressão correta para gerar a lista infinita de números de Hamming. Sugestão: escreva uma definição auxiliar para gerar todos os números da forma $2^i \times 3^j \times 5^k$ tais que $i + j + k = n$ para um n dado.

5.13 A *cifra Vigenère*[2] é uma variante da cifra de César apresentada na aulas que usa vários deslocamentos dados por uma palavra chave. Começamos por repetir a palavra-chave (por exemplo: `LUAR`) ao longo do texto da mensagem; cada letra de `A` a `Z` na chave corresponde a um deslocamento de 0 a 25 (por exemplo: `LUAR` corresponde aos deslocamentos 11, 20, 0 e 17).

mensagem	`ATAQUEDEMADRUGADA`
chave repetida	`LUARLUARLUARLUARL`
texto cifrado	`LNAHFYDVXUDIFAAUL`

(a) Implemente esta cifra como uma função *vigenere* :: *String* $\to$ *String* $\to$ *String* em que o primeiro argumento é a chave e o segundo é a mensagem a cifrar.
Sugestão: Utilize a função `cycle` para construir uma lista infinita com a chave repetida.

(b) Como poderá implementar uma função para descodicar mensagens?
Sugestão: Pense em definir uma função auxiliar para obter uma *chave inversa*, i.e. a chave correspondente aos deslocamentos inversos.

[1] `https://pt.wikipedia.org/wiki/Conjetura_de_Goldbach`
[2] `https://pt.wikipedia.org/wiki/Cifra_de_Vigen%C3%A8re`