

UNIVERSIDADE FEDERAL DE ALFENAS

Curso de Ciência da Computação - Instituto de Ciências Exatas

Disciplina: DCE673 - Programação Funcional - Lista de Exercícios 01

Agradecimentos ao professor Vinícius Ferreira da Silva pelos exercícios [1..10] gentilmente cedidos



1. Declare, em *Haskell*, as funções abaixo, contemplando, também, os protótipos (cabeçalhos):

(a)

$$f_1 : R \rightarrow R; f_1(x) = \begin{cases} \frac{x+4}{x+2}, & \text{se } x \geq 0 \\ \frac{2}{x}, & \text{se } x < 0 \end{cases}$$

(b)

$$f_2 : R^2 \rightarrow R; f_2(x, y) = \begin{cases} x + y, & \text{se } x \geq y \\ x - y, & \text{se } x < y \end{cases}$$

(c)

$$f_3 : R^3 \rightarrow R; f_3(x, y, z) = \begin{cases} x + y + z, & \text{se } (x + y) > z \\ x - y - z, & \text{se } (x + y) < z \\ 0, & \text{se } (x + y) = z \end{cases}$$

2. Localize, explique e corrija o erro na função que deve calcular o fatorial de um número, como se segue:

```
fat :: Int -> Int
fat x = x * fat(x-1)
```

3. Considere a função em *Haskell* soma :: Int -> Int -> Int que retorna a soma entre os dois parâmetros. Assim, faça uma função em *Haskell* que resulte a multiplicação de dois parâmetros fazendo uso da função soma.
4. Escreva, em *Haskell*, a função invertInt :: Int -> Int que inverta os dígitos de um número inteiro.

```
Main> invertInt 123 = 321
```

5. Escreva, em *Haskell*, a definição de uma função fourPower que retorne o seu argumento elevado à quarta potência. Use a função square dada em sala de aula na definição de fourPower.

```
fourPower :: Int -> Int
```

6. Considere a sequência: $\sqrt{6}; \sqrt{6 + \sqrt{6}}; \sqrt{6 + \sqrt{6 + \sqrt{6}}}; \dots$; com tendência ao $+\infty$. Faça, em *Haskell*, uma função para calcular o i -ésimo termo desta sequência, considerando $i_0 = \sqrt{6}$.

7. Escreva, em *Haskell*, uma função que informa de quantas maneiras é possível escolher n objetos em uma coleção original de m objetos, para $m \geq n$.
8. Considere a função escrita na linguagem C que calcula o máximo denominador comum entre dois números:

```
int mdc(int m, int n) {
  while ((m \% n) != 0) {
    int aux = m;
    m = n;
    n = aux \% n ;
  }
  return n ;
}
```

Escreva uma função, em *Haskell*, que calcule o MDC de maneira recursiva.

9. Escreva, em *Haskell*, uma função que retorna quantos múltiplos de um determinado inteiro tem em um intervalo fornecido. Por exemplo, o número 4 tem 2 múltiplos no intervalo de 1 a 10.

```
howManyMultiples 4 1 10 = 2
```

10. Escreva, em *Haskell*, uma função que retorna o último dígito de um número inteiro.

```
lastDigit 1234 = 4
```

11. Escreva, em *Haskell*, uma função que retorna o dígito de um número inteiro de acordo com a posição informada.

```
anyDigit 0 7689 = 7
anyDigit 2 7689 = 8
anyDigit 9 7689 = -1
```

12. Um programador especificou a função *allDifferent* para identificar se três números inteiros são todos diferentes entre si, da seguinte forma:

```
allDifferent::Int->Int->Int->Bool
allDifferent m n p = (m/=n) && (n/=p)
```

- (a) O que está errado nessa definição?
- (b) Especifique corretamente uma função *allDifferent* para o propósito necessário.

13. Escreva uma função *howManyEqual* que retorne quantos dos três números inteiros fornecidos como argumentos são iguais. A resposta poderá ser 3 (todos iguais), 2 (dois iguais e o terceiro diferente) ou 0 (todos diferentes).

```
howManyEqual :: Int -> Int -> Int -> Int
```

14. Para o exemplo da função *sales* :: Int -> Int dada em sala de aula faça o que se pede:

- (a) Implemente a função *howManyLess* que calcule quantos dias as vendas foram inferiores a um dado valor, dentro de um intervalo de dias dentro do período total. O primeiro parâmetro de *howManyLess* indica o valor mínimo de vendas, o segundo parâmetro indica o dia do início do intervalo e o terceiro parâmetro é o dia do fim do intervalo desejado dentro do período total de dias da função;

```
{-parameters: value; interval beginning; interval ending; return value-}  
howManyLess :: Int -> Int -> Int -> Int
```

- (b) Implemente a função *noZeroInPeriod* :: Int -> Bool que retorna True somente se não há nenhum dia no período em que o número de vendas da função *sales* foi zero.
- (c) Implemente a função *zerosInPeriod* :: [Int] que retorne a lista de todos os dias em que as vendas foram de zero unidades;
- (d) Utilizando listas de inteiros, retorne os dias em que as vendas foram abaixo de um determinado valor passado como parâmetro;

15. A sequência de Fibonacci é definida e conhecida na literatura. Os dois primeiros números são 0 e 1, e os seguintes são calculados como a soma dos dois anteriores na sequência. Defina a função *antFib* que, dado um valor *x*, calcule a posição de *x* na sequência de Fibonacci. Caso *x* não esteja na sequência, retorne (-1).

```
{-exemplo-}  
Main> antFib 13 = 7
```

16. Escreva uma definição equivalente à exibida abaixo, mas usando apenas uma única cláusula em casamento de padrão:

```
funny x y z
  | x > z = True
  | y >= x = False
  | otherwise = True
```

17. Implemente uma função que converte uma letra minúsculas como entrada para seu equivalente em maiúsculo. Caso a entrada não seja uma letra minúscula, retorne o próprio caractere de entrada. Como dica, veja a função predefinida `isLower::Char->Bool`. Para verificar outras funções pré-definidas para o tipo `Char`, consulte a biblioteca padrão no endereço <http://zvon.org/other/haskell/Outputglobal/index.html>
18. Defina uma função `charToNum::Char->Int` que converte um dígito numérico do tipo `Char` (como `'3'`) para o valor que ele representa em `Int`, (3). Se o caractere de entrada não representa um dígito numérico, a função deve retornar -1. Como dica, veja as funções *isDigit*, *chr* e *ord* do módulo *Data.Char*.
19. Implemente a função `duplicate::String ->Int->String` que recebe uma string *s* e um número inteiro *n*. A função deve retornar a concatenação de *n* cópias de *s*. Se *n* for zero, retorna `.` Como dica, usar o operador de concatenação pré-definido `(++)::String->String->String`.
20. Implemente a função `pushRight::String->Int->String` que recebe uma string *s* e um número inteiro *n* e retorna uma nova string *t* com *k* caracteres `'>'` inseridos no início de *s*. O valor de *k* deve ser tal que o comprimento de *t* seja igual a *n*. Obs: se *n* é menor que o comprimento de *s*, a função retorna a própria string *s*.

```
{-exemplo-}
Main> pushRight "abc" 5 = ">>abc"
```

21. Defina um operador binário de nome `&-`, com a semântica: $x \&- y = x - 2*y$.
 - Qual é o resultado da avaliação da expressão `10 &- 3 &- 2`, se o operador for definido como: (a) `infixl 6 &-`; (b) `infixr 6 &-`; e (c) `infix 6 &-` ? Explique esses resultados.
 - Qual é o resultado da avaliação da expressão `10 &- 3 * 2`, caso o operador seja definido como: (a) `infix 6 &-`; e (b) `infix 8 &-` ? Explique esses resultados.
22. Faça em Haskell uma solução para inverter os elementos de uma lista de Inteiros.

```
{-exemplo-}
Main> inverte [1,2,3,4,5,6,150] = [150,6,5,4,3,2,1]
```

23. Faça em *Haskell* uma solução para, dada uma lista de inteiros, retornar uma dupla de listas de inteiros onde a primeira conterá os elementos ímpares e a segunda os elementos pares passados como parâmetro.

```
{-exemplo-}  
Main> separa [1,4,3,4,6,7,9,10] = ([1,3,7,9],[4,4,6,10])
```

24. Faça em *Haskell* uma solução para, dada uma lista de inteiros, retornar a string contendo as letras do alfabeto cuja posição é dada pelos elementos da lista.

```
{-exemplo-}  
Main> converte [1,2,6,1,9] = "ABFAI"  
Main> converte [ ] = "".
```

25. Sabendo que $[1..7]$ é equivalente à lista $[1,2,3,4,5,6,7]$, complete as correspondências abaixo:

- (a) $['a'..'g'] =$
- (b) $[0.1 ..0.9] =$
- (c) $[0.1,0.3 .. 0.9] =$
- (d) $[0.1,0.3 ..1.8] =$
- (e) $[0.4,0.2 ..0.8] =$
- (f) $[1,4..15]$

26. Faça em *Haskell* uma solução para o seguinte problema: Dada uma lista de caracteres `[Char]`, e um caractere a , retornar quantos caracteres da lista são iguais a a .

```
{-exemplo-}  
Main> conta "ABCAABCDDA" "B" = 2
```

27. Para uma lista de elementos inteiros ordenada qualquer, faça uma função que retorne uma lista de inteiros ordenada sem elementos repetidos.

```
{-exemplo-}  
Main> purifica [1,1,4,5,5,5,6,7,8,8] = [1,4,5,6,7,8]
```

28. Faça uma solução em *Haskell* que, dada uma lista de inteiros, ela retorne uma lista com uma repetição de cada elemento de acordo com seu valor.

```
{-exemplo-}
Main> proliferaInt [3,0,2,4,0,1] = [3,3,3,2,2,4,4,4,4,1]
```

29. Faça uma solução em *Haskell* que, dada uma lista de caracteres maiúsculos, ela retorne uma lista com uma repetição de cada elemento de acordo com o valor de sua ordem no alfabeto.

```
{-exemplo-}
Main> proliferaChar [C,B,D] = "CCCBDDDD"
```

30. Defina uma função que retorne uma tupla-3(tripla) contendo o caractere fornecido como entrada, o mesmo caractere em letras minúsculo ou maiúsculas, e o seu número da tabela ASCII.

```
{-exemplo-}
Main> converte 'b' = (b, B, 98)
```

31. Seja o cadastro de pessoas dado pela função a seguir:

```

pessoa rg
  | rg == 1 = ("João Silva", 12, 'm')
  | rg == 2 = ("Jonas Souza", 51, 'm')
  ...
  | rg == 321 = ("Jocileide Strauss", 21, 'f')
  | otherwise = ("Não há ninguém mais", 9999, 'x')
```

Construa funções que retornem os seguintes dados:

- (a) O nome da pessoa de menor idade até um determinado registro.
 - (b) A idade média de todas as pessoas até um dado registro.
 - (c) O número de pessoas do sexo masculino.
 - (d) O número do registro da pessoa de maior idade.
32. Construa uma função em *Haskell* que recebe 4 inteiros e devolve uma tupla-4 com os quatros valores originais, só que ordenados.

```
{-exemplo-}
Main> ordena 3 5 1 (-3) = (-3, 1, 3, 5)
```

33. Dadas duas datas (d_1, m_1, a_1) e (d_2, m_2, a_2) , tal que $data_1 \leq data_2$, construa uma função que retorne quantos dias existem entre estas duas datas, onde d_i define o dia do mês m_j no ano a_k .

34. Crie uma função que receba os coeficientes de uma equação do segundo grau $ax^2 + bx + c = 0$ na forma (a,b,c) e retorne as raízes imaginárias, indicando um erro.

```
{-exemplo-}  
Main> equacao (1,(-5), 6) = (2, 3)
```

35. Construa uma função que, dado três valores, verifique se os mesmos podem ser os lados de um triângulo. Se for possível formar o triângulo, retorne uma tupla-2 com o tipo do triângulo formado (com relação às arestas) e o perímetro do mesmo.

```
{-exemplo-}  
Main> triangulo (7,7,11) = ("Isóceles", 25)
```

36. Apresentada a base de dados de 10 professores:

```
base :: Int -> (Int, String, String, Char)  
base x  
  | x == 0 = (1793, "Pedro Paulo", "MESTRE", 'M')  
  | x == 1 = (1797, "Joana Silva Alencar", "MESTRE", 'M')  
  | x == 2 = (1534, "João de Medeiros", "DOUTOR", 'F')  
  | x == 3 = (1267, "Cláudio César de Sá", "DOUTOR", 'M')  
  | x == 4 = (1737, "Paula de Medeiros", "MESTRE", 'F')  
  | x == 5 = (1888, "Rita de Matos", "MESTRE", 'F')  
  ...  
  | x == 9 = (1698, "Tereza Cristina Andrade", "MESTRE", 'F')  
  | x == 10 = (0, "", "", '0')
```

Construa funções que retornem:

- (a) O número de doutores na base.
 - (b) O número de mulheres.
 - (c) O número de mestres do sexo masculino.
 - (d) O nome do professor mais antigo (número de menor matrícula)
37. Para a solução dos próximos itens, sempre use uma função implementada no item anterior para auxiliar nas novas implementações, se possível. Também, use sempre o *underscore* quando um parâmetro não for necessário. Considere, para os itens que se seguem, as funções e tipos:

```
acervo :: [(Isbn, Titulo, Reserva, Volumes)]  
emprestimo :: [(Matricula, Isbn)]
```

```

type Acervo = [(Isbn, Titulo, Reserva, Volumes)]
type Emprestimo = [(Matricula, Isbn)]
type Isbn = Int          -- Isbn de um livro
type Volumes = Int       -- quantidade no acervo
type Titulo = String     -- título do livro
type Matricula = String  -- matrícula do discente
type Reserva = Bool      -- deve permanecer na biblioteca?

```

- (a) O campo *Reserva*, de *acervo*, é *True* quando o livro é reservado ao uso exclusivo na biblioteca. Então, um livro só pode ser emprestado caso esse campo seja *False*. Sabendo disso, faça uma solução em *Haskell*, chamada por *func_1*, que receba um *Isbn* e o *acervo* e, em seguida, informe se o livro para tal *Isbn* pode ser emprestado (*True*) ou não (*False*).

```
func_1::Isbn-> Acervo -> Bool
```

- (b) Faça uma solução em *Haskell* chamada *func_2* que receba um *Isbn* e informe quantos volumes daquele livro estão emprestados.

```
func_2::Isbn -> Emprestimo -> Int
```

- (c) Faça uma solução em *Haskell*, chamada *func_3* que receba um *Isbn* e retorne quantos livros há no *acervo*.

```
func_3::Isbn -> Acervo -> Int
```

- (d) Faça uma solução em *Haskell*, chamada *func_4* que receba um *Isbn* e retorne a quantidade de livros disponível para empréstimo. Para tanto, deve-se observar os três pontos: a) quantos livros você tem no *acervo*; b) quantos estão emprestados; c) se *Reserva*, retorne zero.

```
func_4 :: Isbn -> Int
```

- (e) Faça uma solução em *Haskell*, chamada *func_5* que receba uma matrícula e um *Isbn* e, em seguida, retorne a lista de empréstimo atualizada. Caso o livro não possa ser emprestado (reservado ou sem disponibilidade), deve retornar a lista de empréstimo sem alterações.

```
func_5::Matricula -> Isbn -> Emprestimo
```

- Exercícios 30 ao 36: SÁ, CLÁUDIO CÉSAR DE; SILVA, MÁRCIO FERREIRA DA; Haskell - Uma Abordagem Prática. 1a Ed .Editora Novatec, 2006.

Bom Trabalho!

eliseu César miguel

Texto elaborado em L^AT_EX. Seja Livre! Seja Legal!