Bacharelado em Sistemas de Informação

Programação Funcional

Lista de Exercicios - Linguagem Haskell

1. Forneça uma temperatura em graus *Fahrenheit* a partir de uma temperatura em graus Celsius.

O grau Fahrenheit (símbolo: °F) é uma escala de temperatura proposta por Daniel Gabriel Fahrenheit em 1724. Nesta escala o ponto de fusão da água é de 32 °F e o ponto de ebulição de 212 °F. Uma diferença de 1,8 grau Fahrenheit equivale à de 1 °C.

grau Fahrenheit	grau Celsius	°C = (°F - 32) / 1,8
grau Celsius	grau Fahrenheit	°F = °C × 1,8 + 32

2. Uma empresa decidiu dar a seus funcionários um abono de salario, baseando-se nos pontos obtidos durante o mês, de acordo com a tabela:

Pontos Obtidos	***Prêmio em R$***
1 a 10	*100,00*
11 a 20	*200,00*
21 a 30	*300,00*
31 a 40	*400,00*
A partir de 41	*500*

3. Considere que o preço de uma passagem de avião em um trecho pode variar dependendo da idade do passageiro. Pessoas com 60 anos ou mais pagam apenas 60% do preço total. Crianças até 10 anos pagam 50% e bebês (abaixo de 2 anos) pagam apenas 10%. Faça uma função que tenha como entrada o valor total da passagem e a idade do passageiro e produz o valor a ser pago.

4. Faça uma função que recebe um numero e retorna verdadeiro se o numero for par.

5. Faça uma função que recebe dois valores e retorna o menor.

6. Faça uma função que recebe três valores e retorna o menor.

7. Escreva uma função recursiva para calcular o fatorial de um numero natural.

8. Especifique as seguintes funções para a manipulação de listas:

   a) **nro-elementos**: recebe uma lista qualquer e retorna o número de elementos na lista.

b) **maior**: recebe uma lista de números e retorna o maior .

c) **conta-ocorrencias**: recebe um elemento e uma lista qualquer e retorna o número de ocorrências do elemento na lista.

d) **unica-ocorrencia**: recebe um elemento e uma lista e verifica se existe uma única ocorrência do elemento na lista .
ex.:
unica-ocorrencia 2 [1,2,3,2] = False
unica-ocorrencia 2 [3,1] = False
unica-ocorrencia 2 [2] = True

e) **maiores-que**: recebe um número e uma lista de números e retorna uma lista com os números que são maiores do que o valor informado.
ex.:
maiores-que 10 [4 6 30 3 15 3 10 7] ==> [30 15]

f) **concatena**: recebe duas listas quaisquer e retorna uma terceira lista com os elementos da primeira no início e os elementos da segunda no fim.
ex.:
concatena [] [] ==> []
concatena [1 2] [3 4] ==> [1 2 3 4]

g) **duplica**: recebe uma lista e retorna uma nova lista contendo a duplicação dos elementos da lista original.
ex: duplica [1, 2, 3] ==> [1,1,2,2,3,3]

-- Soluções dos exercícios

-- Exercício 1

```
-- De Celsius para Fahrenheit
converteCF::Float->Float
converteCF x = x * 1.8 + 32

-- De Fahrenheit para Celsius
converteFC::Float->Float
converteFC x = (x - 32)/ 1.8
```

-- Exercício 2

```
-- Calcular Abono de Salario
calculaAbono::Int->Int
calculaAbono x | x > 1 && x <=10  = 100
               | x > 10 && x <=20 = 200
               | x > 20 && x <=30 = 300
               | x > 30 && x <=40 = 400
               | x > 40           = 500
```

-- Exercício 3

```
-- Desconto Passagem
calculaDesconto::Float->Int->Float
calculaDesconto v i | i >= 60 = v * 0.6
                    | i > 2 && i<=10 = v * 0.5
                    | i < 2 = v * 0.1
                    | otherwise = v
```

-- Exercício 4

```
-- Verifica se numero é par
par::Int->Bool
par x = if mod x 2 == 0 then True else False
```

-- Exercício 5

```
-- Retorna o menor entre dois números
menor::Int->Int->Int
menor x y = if x < y then x else y
```

-- Exercício 6

```
-- Retorna o menor entre três números
menor3::Int->Int->Int->Int
menor3 x y z | x > y  =  if y > z then z else y
             | otherwise = if x > z then z else x
```

-- Exercício 7

```
-- Fatorial de um número natural
fatorial::Int->Int
fatorial 0 = 1
fatorial n = n * fatorial (n-1)
```

-- Exercício 8.a

```
- Conta o nro_elementos de uma lista
conta::[Int]->Int
conta [] = 0
conta (a:x) = 1 + conta x
```

-- Exercício 8.b

```
-- Retorna o maior elemento de uma lista de inteiros
maior::[Int]->Int
maior[a] = a
maior(a:x) = if a > maior x then a
                            else maior x
```

-- Exercício 8.c

```
-- Conta o número de ocorrências de um número na lista
contaOc::Int->[Int]->Int
contaOc a [] = 0
contaOc a (x:xs) | x == a = 1 + contaOc a xs
                 | otherwise = contaOc a xs
```

-- Exercício 8.d

```
-- Verifica se o elemento é único na lista
-- Versão 0, utilizando a função "contaOC" do exercicio 8.c:
unicaOcorrencia::Int -> [Int] -> Bool
unicaOcorrencia a lista =
     if (contaOc a lista == 1) then True else False

-- Versao 1, utilizando uma função "pertence":
unicaOc::Int -> [Int] -> Bool
unicaOc a [] = False
unicaOc a (x:xs) | x == a = if (pertence a xs) then False
                                                else True
                 | otherwise = unicaOc a xs

pertence :: Int -> [Int] -> Bool
pertence x [] = False
pertence x (a:t) | (x == a) = True
                 | otherwise = pertence x t

-- Versão 2, recursiva
unicaOcor::Int -> [Int] -> Bool
unicaOcor a [] = False
unicaOcor a (x:xs) | x == a = if (unicaOcor a xs) then False
                                                  else True
                   | otherwise = unicaOcor a xs
```

-- Exercício 8.e

```
-- Cria uma lista com os elementos maiores que x
maiores::Int->[Int]->[Int]
maiores x [] = []
maiores x (a:t) = if a > x then   a: maiores x t
                          else   maiores x t
```

-- Exercício 8.f

```
-- Concatena duas listas
concatena::[Int]->[Int]->[Int]
concatena [] l = l
concatena (a:x) l = a:concatena x l
```

-- Exercício 8.g

```
-- Duplica os elementos de uma lista de inteiros
duplica::[Int]->[Int]
duplica [] = []
duplica (a:x) = a: a: duplica x
```