

Primeiro Trabalho de Programação Funcional – 2020/1

Data de entrega: 13/04/2021

Valor: 22 pontos

Obs1: Esse trabalho pode ser desenvolvido em **Dupla** ou **Individual**, mas a arguição na apresentação e a respectiva nota será individual.

Obs2: A nota será dada pela nota de desenvolvimento (de 0 a 20) multiplicada pela nota de arguição (de 0 a 1), onde o aluno deverá demonstrar completo entendimento do trabalho desenvolvido.

Obs3: O arquivo .hs deve ser enviado para o e-mail claudineyrt@gmail.com.

1) Escreva a função `analisa_raizes` que, dados os 3 coeficientes a , b e c de uma equação quadrática $ax^2 + bx + c = 0$, realiza a análise das raízes da equação. A equação é dita degenerada, se o coeficiente do termo quadrático a for igual a zero. Por outro lado, uma equação não degenerada possui o número de raízes reais de acordo com as regras: (1) a equação possui duas raízes reais, se $b^2 > 4*a*c$; (2) a equação possui uma raiz real, se $b^2 = 4*a*c$; (3) a equação não possui raízes reais, se $b^2 < 4*a*c$. A análise de saída deve ser uma das 4 opções a seguir: “1-possui duas raízes reais”, “2-possui uma raiz real”, “3-nenhuma raiz real” ou “4-equação degenerada”.

2) Escreva a função `equacao` que recebe três valores reais a , b , c . Se a for diferente de 0, a função retorna uma tupla com as duas raízes da equação de segundo grau $ax^2 + bx + c = 0$. Se a for igual a 0, a função retorna uma tupla, sendo o primeiro elemento a solução da equação de primeiro grau $bx + c = 0$ e o segundo elemento o próprio a .

3) Considere que o preço de uma passagem de ônibus intermunicipal pode variar dependendo da idade do passageiro. Crianças até 10 anos pagam 40% e bebês (abaixo de 2 anos) pagam apenas 15%. Pessoas com 70 anos ou mais pagam apenas 50% do preço total. Os demais passageiros, pagam a tarifa normal (100%). Faça uma função que tenha como entrada: o valor total da passagem, a data atual e a data de nascimento do passageiro. Como saída, a função retorna o valor a ser pago. (Obs.: na solução, deve ser definido o tipo data para representar a tupla (d,m,a)).

4) Construa funções que gerem as seguintes listas, utilizando-se lista por compressão. Todas as funções devem utilizar a lista de inteiros de 1 a 20 em pelo menos um dos geradores. Apresentar o código da função e o resultado da lista gerada.

a) **gera1**: gerar a lista de inteiros, contendo o cubo de todos os pares entre 3 e 11.

b) **gera2**: gerar a lista de duplas formadas tendo o primeiro elemento menor ou igual a 5 e o segundo elemento no intervalo fechado entre o valor do primeiro elemento e o seu triplo.

c) **gera3**: a partir de uma lista `l1=[15,16]`, gerar a lista com todos os elementos dentro do intervalo fechado definido entre 1 e cada elemento de `l1` (Obs.: pode ter elemento repetido na lista final).

d) **gera4**: gerar uma lista de duplas, onde cada dupla são 2 números consecutivos de 1 a 10, sendo o primeiro elemento par (Ex: (2,3) e (4,5))

e) **gera5**: a partir da lista de duplas geradas no item d, gerar a lista onde cada elemento corresponde à soma dos elementos da dupla.

5) a) Escreva uma função (usando compreensão de listas) que calcula a quantidade de números que são positivos e múltiplos de 3 (ao mesmo tempo) de uma lista de inteiros:

```
> contaNegM2 [1,-3,-4,3,4,-5,-8,-7,9]
2
```

b) Escreva uma função (usando compreensão de listas) que extrai números que são positivos e múltiplos de 3 (ao mesmo tempo) de uma lista de inteiros e os retorna em uma nova lista:

```
> listaNegM2 [1,-3,-4,3,4,-5,-8,-7,9]
[3,9]
```

6) Escreva a função `primos` a seguir que recebe dois valores inteiros `x,y` e retorna todos os números primos que se encontram entre `x` e `y`. Obs: construir uma segunda função `fatores` que retorna todos os divisores de um número inteiro e utilizá-la na elaboração da função `primos`.

```
> primos 10 50
[11,13,17,19,23,29,31,37,41,43,47]
```

7) Construa a função `mmc` a seguir que calcula o valor do mínimo múltiplo comum de três números inteiros.

```
> mmc 2 3 4
12
```

8) Escreva uma função que calcula a série a seguir, dados um número real x e o número de termos a serem calculados n . Obs: se preciso, use a função `fromIntegral` para converter n de Inteiro para Float.

$$Y = \frac{1}{x} + \frac{x}{2} + \frac{3}{x} + \frac{x}{4} + \dots$$

9) Escreva a função `fizzbuzz` a seguir que recebe um inteiro n e retorna uma lista de strings. Para cada inteiro i entre 1 e n , a lista será composta da seguinte forma.

- Se i é divisível por 2, escreva "Fizz".
- Se i é divisível por 3, escreva "Buzz".
- Se i é divisível por ambos 2 e 3, escreva "FizzBuzz".
- Caso contrário, diga "No".

Exemplo:

```
> fizzbuzz 10
["No", "Fizz", "Buzz", "Fizz", "No", "FizzBuzz", "No", "Fizz", "Buzz", "Fizz"]
```

10) Usando lista por compreensão, escreva a função `seleciona_multiplos` que recebe um lista de inteiros e um inteiro n e retorna uma nova lista com todos os números presentes na lista de entrada que são múltiplos de n . Ex:

```
> sel_multiplos 3 [2,6,4,3,1,5,9]      > sel_multiplos 2 [2,6,4,3,1,5,9]
[6,3,9]                                [2,6,4]
```

11) Escreva a função `unica_ocorrencia` a seguir que recebe um elemento e uma lista e verifica se existe uma única ocorrência do elemento na lista .

```
> unica_ocorrencia 2 [1,2,3,2]          > unica_ocorrencia 2 [3,1]
False                                   False
```

12) Crie uma função que `intercala` os elementos de duas listas de qualquer tamanho numa nova lista. Obs: as listas de entrada devem ser do mesmo tipo mas podem ter tamanhos diferentes. Caso sejam diferentes, os elementos excedentes da lista maior devem complementar a lista de saída

```
> intercala [1,2,3,4] [100,200]
```

```
[1,100,2,200,3,4]
> intercala [1,2] [100,200,300]
[1,100,2,200,300]
```

13) Construa a função `zipar` que recebe duas listas de mesmo tipo e elabora uma terceira lista de sublistas de tamanho 2 formadas por um elemento da primeira lista e outro da segunda lista. Assim, a primeira sublista da lista de saída é formada com o primeiro elemento de cada lista, a segunda sublista é formada com o segundo elemento de cada uma, e assim por diante. Se listas de entrada tiverem tamanhos diferentes, a lista de saída é truncada para o tamanho da menor lista de entrada. Obs: não é permitido usar funções de manipulação de listas da biblioteca PRELUDE padrão do Haskell. Ex:

```
> zipar [1,2,3,4] [5,6,7,8]          > zipar [1,2,3] [4,5]
[[1,5],[2,6],[3,7],[4,8]]          [[1,4],[2,5]]
```

14) Defina novos tipos para representar os dados contidos numa agenda pessoal. Para cada contato, armazene as informações: nome, endereço, telefone, e-mail. Em seguida, crie uma função para recuperar o nome de um contato, a partir do email. Caso o número não seja encontrado, retornar a mensagem “Email desconhecido”.

15) Seja o tipo `Pessoa` e a lista de pessoas a seguir.

O tipo `pessoa` é uma tupla que inclui nome, altura, idade e estado civil ('c' ou 's').

```
type Pessoa = (String, Float, Int, Char)
pessoas :: [Pessoa]
pessoas = [ ("Rosa", 27, 1.66, 'F'),
            ("João", 1.85, 26, 'C'),
            ("Maria", 1.55, 62, 'S'),
            ("Jose", 1.78, 42, 'C'),
            ("Paulo", 1.93, 25, 'S'),
            ("Clara", 1.70, 33, 'C'),
            ("Bob", 1.45, 21, 'C'),
            ("Rosana", 1.58, 39, 'S'),
            ("Daniel", 1.74, 72, 'S'),
            ("Jocileide", 1.69, 18, 'S') ]
```

Escreva funções que, dada a lista `pessoas`, retornem:

- A altura média entre todas as pessoas.
- A idade da pessoa mais nova.
- O nome e o estado civil da pessoa mais velha.
- Todos os dados de cada pessoa com 50 anos ou mais.
- O número de pessoas casadas com idade superior a i (ex: $i = 35$).

16) Escreva a função `insere_ord` a seguir, que recebe uma lista polimórfica ordenada de elementos (critério de ordenação crescente) e um novo elemento `x` (do mesmo tipo da lista) e retorna a nova lista com o novo elemento inserido

```
> insere_ord 5 [1,4,7,11]
[1,4,5,7,11]
> insere_ord 'g' "abcjkl"
"abcgjkl"
```

17) Escreva a função `reverte` a seguir que recebe uma lista polimórfica e retorna uma lista com seus elementos ao contrário.

```
> reverte [1,2,3,4]
[4,3,2,1]
> reverte "abcd"
"dcba"
```

18) Escreva a função `elimina_repet` a seguir que recebe uma lista polimórfica e retorna uma lista sem elementos repetidos.

```
> elimina_repet [3,3,2,9,1,7,2,5,9,7]
[3,2,9,1,7,5]
> elimina_repet "mississippi"
"misp"
```

19) Escreva a função `notasTroco` a seguir usando compreensão de listas que calcula todas as combinações de notas para devolver o troco durante um pagamento, a partir de uma lista com os valores das notas disponíveis (definido no arquivo `.hs`) e o valor do troco `x` (argumento da função). Ex:

Considere `disponiveis = [1,2,5,10,20,50,100]`

```
> notasTroco 4
[[1,1,1,1],[1,1,2],[1,2,1],[2,1,1],[2,2]]
> notasTroco 7
[[1,1,1,1,1,1,1],[1,1,1,1,1,2],[1,1,1,1,2,1],[1,1,1,2,1,1],
[1,1,1,2,2],[1,1,2,1,1,1],[1,1,2,1,2],[1,1,2,2,1],[1,1,5],
[1,2,1,1,1,1],[1,2,1,1,2],[1,2,1,2,1],[1,2,2,1,1],[1,2,2,2],
[1,5,1],[2,1,1,1,1,1],[2,1,1,1,2],[2,1,1,2,1],[2,1,2,1,1],
[2,1,2,2],[2,2,1,1,1],[2,2,1,2],[2,2,2,1],[2,5],[5,1,1],[5,2]]
```

20) Desenvolver a função `nRainhas` que resolve o Problema das N rainhas, para um valor n dado como entrada. Esse problema consiste em posicionar N rainhas num tabuleiro $N \times N$ de forma que nenhuma rainha possa capturar outra rainha em um único movimento. O resultado deve conter uma lista de todas as soluções possíveis para o valor n dado como entrada, em que cada solução é uma lista que apresenta a posição da linha de cada rainha em ordem de coluna (colunas de 1 a N). Por exemplo, a lista `[3,1,4,2]` é uma possível solução para o problema de 4 rainhas em um tabuleiro 4×4 , onde: a 1ª rainha é posicionada na 1ª coluna e 3ª linha, a 2ª rainha é posicionada na 2ª coluna e 1ª linha, a 3ª rainha é posicionada na 3ª coluna e 4ª linha e a 4ª rainha é posicionada na 4ª coluna e 2ª linha. Note que essa não é a única solução para a instância de 4 rainhas e a lista `[3,1,4,2]` é uma sub-lista da lista de saída.