Primeiro Trabalho de Programação Funcional – Modalidade AARE
Data de entrega: 08/09/2020
Valor: 20 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.

1) Escreva a função `triangulo` que, dados três valores de ângulos (em graus), verifique se podem representar os ângulos internos um triângulo e retorne o tipo do triângulo resultante (como uma string). Os retornos podem ser:

- "equilatero": se os três ângulos são iguais
- "retangulo": se um dos ângulos é 90 graus
- "obtuso": se um dos ângulos é maior que 90 graus
- "simples": se é um triângulo e não se enquadra em nenhum dos casos anteriores (equilátero, retângulo ou obtuso)
- "nao_triangulo": se não puder ser classificado como triângulo

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 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 15 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 quadrado de todos os ímpares entre 4 e 14.

b) **gera2**: gerar a lista de duplas formadas tendo o primeiro elemento entre 1 e 4 e o segundo elemento no intervalo fechado entre o valor do primeiro elemento e o seu dobro.

c) **gera3**: a partir de uma lista l1 entre 10 e 15, 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 16, sendo o primeiro elemento ímpar (Ex: (1,2) e (3,4))

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 negativos e múltiplos de 2 (ao mesmo tempo) de uma lista de inteiros:

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

b) Escreva uma função (usando compreensão de listas) que extrai números que são negativos e múltiplos de 2 (ao mesmo tempo) de uma lista de inteiros e os retorna em uma nova lista:

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

6) Seja a função abaixo que recebe uma lista de pontos no plano cartesiano e calcula a distância de cada ponto à origem:

```
distancias :: [(Float,Float)] -> [Float]
distancias [] = []
distancias ((x,y):xys) = (sqrt (x^2 + y^2)) : (distancias xys)
```

Escreva uma outra versão da função `distancias` utilizando a construção de listas por compreensão.

7) 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]
```

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

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

9) 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.

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

10) 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 3, escreva “Fizz”.
- Se i é divisível por 5, escreva “Buzz”.
- Se i é divisível por ambos 3 e 5, escreva “FizzBuzz”.
- Caso contrário, diga “No”.

Exemplo:

```
> fizzbuzz 15
["No","No","Fizz","No","Buzz","Fizz","No","No","Fizz","Buzz","No","
Fizz","No","No","FizzBuzz"]
```

11) Escreva a função `conta_ocorrencias` que recebe dois elemento e uma lista qualquer e retorna um tupla com o número de ocorrências de cada elemento na lista. Obs: a lista deve ser percorrida uma única vez. O topo dos elementos deve ser o mesmo da lista de entrada. Exemplo:

```
> conta_ocorrencias 3 7 [-1,3,-4,3,4,3,-8,7,7,4]
(3,2)
```

12) 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]
False
> unica_ocorrencia 2 [3,1]
False
> unica_ocorrencia 2 [2]
True
```

13) 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]
```

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 `sem_repetidos` a seguir que recebe uma lista polimórfica e retorna uma lista sem elementos repetidos.

```
> sem_repetidos [3,3,2,9,1,7,2,5,9,7]
[3,2,9,1,7,5]

> sem_repetidos "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 x 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 4x4, onde: a 1ª rainha é posicionada na 1ª coluna e **3**ª linha, a 2ª rainha é posicionada na 2ª coluna e **1**ª linha, a 3ª rainha é posicionada na 3ª coluna e **4ª** linha e a 4ª rainha é posicionada na 4ª coluna e **2**ª 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.