Relatório Técnico Referente Primeira Avaliação
da Disciplina de Programação Funcional

Paulo Henrique de Carvalho Silva
Elias de Sousa Neto

Resumo do Projeto

1. Introdução

Este relatório tem o intuito de demonstrar o desenvolvimento como programador no estudo da linguagem Haskell, bem como o paradigma funcional. Para isto, foi respondido uma série de atividades com o intuito de aprimorar o conhecimento acerca do paradigma funcional fazendo uso da linguagem de programação Haskell.

As atividades respondidas neste relatório testam o aprendizado acerca das funcionalidade mais básicas da linguagem, funcionalidades como operadores, parâmetros, variáveis, funções com e sem recursividade e listas.

2. Seções Específicas

O experimento deste trabalho trata-se de uma série de atividades que devem ser respondidas utilizando a linguagem haskell. Mais adiante pode-se encontrar a questão bem como a resposta, isto é, o código que corresponde a sua questão.

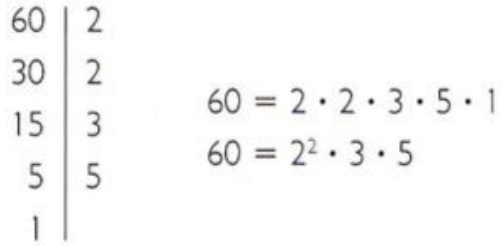
2.1Questão 1

Enunciado da questão:

"Faça um programa em Haskell que leia três números inteiros e então faça a fatoração entre eles e mostre os valores que aparecem na fatoração e quantas vezes os mesmos aparecem."

A fatoração entre dois ou mais números é um processo utilizado na matemática que consiste em representar um número ou uma expressão como produto de fatores.

ex:



A lógica utilizada para responder a questão é bem semelhante, onde a diferença é que a fatoração foi feita com 3 elementos ao invés de 1. Na imagem abaixo encontra-se a

função principal deste problema, onde temos uma função “verifica” que como o nome diz verifica se o primeiro parâmetro é divisível pelo segundo parâmetro, esta função é importante para reduzir código nas comparações da função abaixo, a função de “fatoração” é a função principal. Nesta função a função de parada é quando todos os números chegaram ao último nível da divisão, isto é, quando todos forem iguais a 1, caso isso não ocorra é feito uma verificação se os numeros sao divisiveis pelo “n” que é o divisor da função, caso ninguém seja divisível “n” é incrementado.

```
verifica(a,n) = mod a n == 0    Top-level binding with no type signature: verifica :: Integral a => (a, a)

--funcao que devolve a lista com os numeros da fatoracao
fatoracao :: (Num a2, Show a1, Show a2, Integral a1, Ord a2) => (a1, a1, a1, a1, a2) -> IO ()
fatoracao(a,b,c,n,cont)
    | a==1 && b==1 && c==1 = mostra(n, cont)
    | verifica(a,n) && verifica(b,n) && verifica(c,n) = fatoracao((div a n),(div b n),(div c n),n,cont+1)
    | verifica(a,n) && verifica(b,n) = fatoracao((div a n),(div b n),c,n,cont+1)    Redundant bracket Four
    | verifica(a,n) && verifica(c,n) = fatoracao((div a n),b,(div c n),n,cont+1)    Redundant bracket Four
    | verifica(b,n) && verifica(c,n) = fatoracao(a,(div b n),(div c n),n,cont+1)    Redundant bracket Four
    | verifica(a,n) = fatoracao((div a n),b,c,n,cont+1)    Redundant bracket Found: ((div a n), b, c, n, c
    | verifica(b,n) = fatoracao(a,(div b n),c,n,cont+1)    Redundant bracket Found: (a, (div b n), c, n, c
    | verifica(c,n) = fatoracao(a,b,(div c n),n,cont+1)    Redundant bracket Found: (a, b, (div c n), n, c
    | not(verifica(a,n)) && not(verifica(b,n)) && not(verifica(c,n)) = do
                                                                    fatoracao(a,b,c,n+1,0)
                                                                    mostra(n, cont)
```

2.2 Questão 2

Enunciado da questão:
“Faça uma função em Haskell que imprima quantos ingressos de uma peça teatral devem ser vendidos para que o produtor da peça obtenha lucro, e imprima também o número de seções da peça que devem ser realizadas. Considere que o custo da peça teatral é de 50.000 reais e que para cada sessão apresentada o produtor tem que pagar 10.000 reais. Além disso, que o ingresso é de 50 reais e que em cada sala cabem 300 pessoas.”

Aqui primeiro todos os dados necessários foram pegos por entrada de teclado, já que pela observação o cálculo deve ser feito de forma que se os valores forem alterados as funções não precisam ser refeitas.

```
lc(valor) = valor*(-1)    Top-level binding with no type signature: lc :: Num a => a -> a
principal :: IO ()
principal = do
    putStr "Digite o valor da peca: "    Top-level binding with no type signature: principal :: IO
    a <- getLine
    putStr "Digite o valor por seccao: "
    b <- getLine
    putStr "Digite o valor do ingresso: "
    c <- getLine
    putStr "Digite a capacidade do local: "
    d <- getLine
    putStr "\n"

    putStr "O lucro vem: \n"
    quantidade(read a::Int, read b::Float, 0, read c::Float, read d::Int, 0, lc(read a::Float))
    putStr "\n\n"        * Defaulting the following constraints to type `Integer' (Show f0) arisin
```

Após a leitura dos dados, então os valores são passados para a função principal, onde a lógica é o lucro inicial com o valor da peça negativa, isto é, o débito e então os ingressos são vendidos até que a seção esteja cheia, então é iniciado uma nova seção (lembrando que toda vez que uma nova seção começa é acrescentado ao débito o valor negativo do adicional).

```
quantidade(valor_peca, adicional, ingresso, valor, capacidade, qsecoes, lucro)    Top-level binding with n
    | lucro > 0 = putStr("ingressos: ") >> print(ingresso) >> putStr("sessões: ") >> print(qsecoes)    Red
    | lucro <= 0 && mod ingresso capacidade == 0 = quantidade(valor_peca, adicional, ingresso+1,valor,
      capacidade, qsecoes+1,lucro+valor-adicional)
    | otherwise = quantidade(valor_peca, adicional, ingresso+1, valor, capacidade, qsecoes,(lucro+valor))
```

2.3 Questão 3

Enunciado da questão:

"Ana, Beatriz e Carolina sempre saem juntas para tomar café numa padaria onde as mesas são circulares e têm três cadeiras numeradas 0, 1 e 2, no sentido anti-horário. Elas gostam de decidir quem vai sentar em qual cadeira com uma brincadeira gerando números aleatórios nos seus celulares. Primeiro Ana sorteia um número inteiro A e, começando da cadeira 1, seguindo no sentido anti-horário, conta A cadeira e senta na cadeira em que a contagem termina. Depois Beatriz sorteia um número B e faz a mesma coisa: começando da cadeira 1, no sentido anti-horário, conta B cadeiras. Se a cadeira final estiver livre, Beatriz senta nela. Caso seja a cadeira onde Ana está sentada, então Beatriz senta na próxima cadeira no sentido anti-horário. Claro, ao final, Carolina senta na cadeira que está livre."

A lógica utilizada nesta questão foi dividida em 3 funções, onde cada função trata de onde cada garota vai sentar, começando pela Ana, então Bea e por fim a Carolina.

```
fimL :: Integral a => a -> Bool
fimL cont = mod cont 3 == 0
proximo :: [a] -> a
proximo(h:r) = h
anaAssento :: Integral d => ([a], [a], d, d) -> a
anaAssento(h:t, laux, nAna, cont)
    | cont == nAna = h
    | fimL cont = anaAssento(laux, laux, nAna, cont+1)
    | otherwise = anaAssento(t, laux, nAna, cont+1)
```

```
beaAssento :: (Eq c, Integral e) => ([c], [c], c, e, e) -> c
beaAssento(h:t, laux, ana, nBea, cont)
    | cont == nBea && h /= ana = h
    | cont == nBea && fimL cont  = proximo laux
    | cont == nBea = proximo t
    | fimL cont = beaAssento(laux, laux, ana, nBea, cont+1)
    | otherwise = beaAssento(t, laux, ana, nBea, cont+1)
```

```haskell
carolAssento :: (Eq c, Num d) => ([c], c, c, d) -> c
carolAssento(h:t, ana, bea, cont)
    | h /= ana && h /= bea = h
    | otherwise = carolAssento(t, ana, bea, cont+1)
```

2.4 Questão 4

Enunciado da questão:

“Faça um programa em Haskell que leia uma lista de strings e então faça: (1,5 pontos)
(a) uma função para contar o número de caracteres que cada string possui sem repetir;
(b) uma função que devolve uma lista contendo os tipos de caracteres que iniciam as strings da lista, por exemplo: vogal, dígito ou outro tipo de carácter e se o mesmo é ou não maiúsculo, quando possível.
(c) uma função que devolva a string que possui o maior número de vogais.”

Nesta atividade, temos uma peculiaridade onde a entrada deve ser uma lista de strings então para facilitar na leitura ao invés de uma função que lê item a item da lista de palavras, foi feito uma função que transforma uma frase, isto é, as palavras separadas por espaço em uma lista de palavras.

Ex:

Entrada: “Paulo Teste 123 @menu Haskell2"

Saida: [“Paulo”,”Teste”,”123”,”@menu”,”Haskell2”]

```haskell
proxima :: [Char] -> [Char]
proxima([]) = []    Top-level binding with no type signature: p
proxima(h:t)
    | h == ' ' = t
    | otherwise = proxima(t)    Redundant bracket Found: (t) Wh

separa :: [Char] -> [Char]
separa([]) = []        Top-level binding with no type signature
separa(h:t)
    | h == ' ' = []
    | otherwise = h : separa(t)    Redundant bracket Found: (t)

minhaLista :: [Char] -> [[Char]]
minhaLista([]) = []    Top-level binding with no type signature
minhaLista(lista) = separa(lista) : minhaLista(proxima(lista))
```

Quanto ao resto da execução do problema, para cada item (a, b, c) foi feito uma função para atender a sua demanda.

a)

```haskell
proxima :: [Char] -> [Char]
proxima([]) = []    Top-level binding with no type signature: p
proxima(h:t)
    | h == ' ' = t
    | otherwise = proxima(t)    Redundant bracket Found: (t) Wh

separa :: [Char] -> [Char]
separa([]) = []        Top-level binding with no type signature
separa(h:t)
    | h == ' ' = []
    | otherwise = h : separa(t)    Redundant bracket Found: (t)

minhaLista :: [Char] -> [[Char]]
minhaLista([]) = []    Top-level binding with no type signature
minhaLista(lista) = separa(lista) : minhaLista(proxima(lista))
```

b)

```
verifica :: Eq a => (a, [a]) -> Bool
verifica(letra, []) = False    Top-level binding with no type signature: ve
verifica(letra, h:t)    This binding for `letra' shadows the existing bindi
    | letra == h = True
    | otherwise = verifica(letra, t)

letra :: [a] -> a
letra(h:t) = h    Top-level binding with no type signature: letra :: [a] ->

tipo :: [[Char]] -> [String]
tipo([]) = []    Top-level binding with no type signature: tipo :: [[Char]]
tipo(h:t)
    | verifica(letra(h),vogaisMini) = "Vogal Minuscula" : tipo(t)    Redund
    | verifica(letra(h),vogaisMai) = "Vogal Maiuscula" : tipo(t)    Redunda
    | verifica(letra(h),consoanteMini) = "Consoante Minuscula" : tipo(t)
    | verifica(letra(h),consoanteMai) = "Consoante Maiuscula" : tipo(t)
    | verifica(letra(h),digito) = "Digito" : tipo(t)    Redundant bracket F
    | otherwise = "Caractere Especial" : tipo(t)    Redundant bracket Found
```

c)

```
contaVogal :: Num a => [Char] -> a
contaVogal([]) = 0    Top-level binding with no type signature: contaVo
contaVogal(h:t)
    | repete(h, vogaisMai) || repete(h, vogaisMini) = 1 + contaVogal t
    | otherwise = 0 + contaVogal t

maisVogal :: ([[Char]], [Char]) -> [Char]
maisVogal([], maior) = maior    Top-level binding with no type signatur
maisVogal(h:t, maior)
    | contaVogal(h) >= contaVogal(maior) = maisVogal(t,h)    * Defaulti
    | otherwise = maisVogal(t,maior)
```

2.5 Questão 5

Enunciado da questão:

"Faça um programa em Haskell que leia duas listas de inteiros ordenadas, A e B, e então faça:

(a) uma função que devolve uma lista contendo a união ordenada entre (A – B) e (B – A).

(b) uma função que devolva uma lista contendo a soma entre os quadrados dos elementos das duas listas que forem maiores do que a soma entre o cubo dos dois primeiros elementos da lista."

De forma semelhante à questão anterior, para cada item (a, b) foram feitas funções que satisfaçam seus requisitos. Porém desta vez não foi necessário tratar a entrada já que a entrada já é bem definida como duas listas de inteiros ordenadas.

a) aqui passou primeiro A, B para a função (A - B) e depois B, A (B - A)

```
existeEm :: Eq b => ([b], b) -> Bool
existeEm([], item) = False    Top-level bi
existeEm(h:t, item)
    | item == h  = True
    | otherwise = existeEm(t, item)

uniao :: Eq a => ([a], [a]) -> [a]
uniao([], lista) = []    Top-level binding
uniao(h:t, lista)
    | existeEm(lista,h) = uniao(t, lista)
    | otherwise = h : uniao(t, lista)
```

b)

```
somaQ :: (Num c, Ord c) => ([c], [c], c) -> [c]
somaQ([],c:r,quad) = []    Top-level binding with no
somaQ(h:t,[],quad) = []    Defined but not used: `h'
somaQ(h:t,c:r,quad)
    | (h^2+c^2) > quad = (h^2+c^2) : somaQ(t,r,quad)
    | otherwise = somaQ(t,r,quad)
```

3. Resultados da Execução do Programa

A fim de complementar a seção anterior, esta seção apresenta os resultados obtidos da execução dos códigos apresentados anteriormente.

3.1 Código 1

Dados de entrada:
10, 20 30 - ordenado
Saída:

```
Numeros: 10, 20, 30
Divisores - Quantas vezes repete
5 - 1
3 - 1
2 - 2
```

48, 24, 12 - ordenado do maior para o menor
Saída:

```
Numeros: 10, 20, 30
Divisores - Quantas vezes repete
5 - 1
3 - 1
2 - 2
```

32, 16, 64 - completamente desordenado
Saída:

```
Numeros: 32, 16, 64
Divisores - Quantas vezes repete
2 - 6
```

3.2 Código 2

Valores originais da peça
Entrada:
Valor da peça: 50.000
Adicional: 10.000
Valor do ingresso: 50
Capacidade: 300
Saída:

```
O lucro vem:
ingressos: 3201
sessões: 11
```

Entrada com o dobro dos valores originais da peça
Valor da peça: 100.000
Adicional: 20.000
Valor do ingresso: 100
Capacidade: 600
Saída:

```
O lucro vem:
ingressos: 1601
sessões: 3
```

3.3 Código 3

Entrada: bom aqui não temos entrada, a entrada sempre é aleatória então apenas será mostrado a saída:

```
Numero sorteado por Ana: 89
Ana sentou na cadeira: 0
Numero sorteado por Beatriz: 65
Beatriz sentou na cadeira: 2
Carolina sentou na cadeira: 1
```

```
Numero sorteado por Ana: 89
Ana sentou na cadeira: 0
Numero sorteado por Beatriz: 65
Beatriz sentou na cadeira: 2
Carolina sentou na cadeira: 1
```

```
Numero sorteado por Ana: 58
Ana sentou na cadeira: 1
Numero sorteado por Beatriz: 36
Beatriz sentou na cadeira: 2
Carolina sentou na cadeira: 0
```

3.4 Código 4

Entrada: “Teste 0123 Python ppyytthhoonn @UFPI” - teste para verificar a questão “a”

Saída:

```
Digite sua lista de string (palavras separadas por espaco): Teste 0123 Python ppyytthhoonn @UFPI

a)Uma função para contar o número de caracteres que cada string possui sem repetir:
Item: Teste Tamanho: 4
Item: 0123 Tamanho: 4
Item: Python Tamanho: 6
Item: ppyytthhoonn Tamanho: 6
Item: @UFPI Tamanho: 5



b)Uma função que devolve uma lista contendo os tipos de caracteres que iniciam as strings da lista:
["Consoante Maiuscula","Digito","Consoante Maiuscula","Consoante Minuscula","Caractere Especial"]

c)Uma função que devolva a string que possui o maior número de vogais:
"@UFPI"
```

Entrada: “BCD FGH JKL” - teste para verificar a questão “c”
Saída:

```
Digite sua lista de string (palavras separadas por espaco): BCD FGH JKL

a)Uma função para contar o número de caracteres que cada string possui sem repetir:
Item: BCD Tamanho: 3
Item: FGH Tamanho: 3
Item: JKL Tamanho: 3



b)Uma função que devolve uma lista contendo os tipos de caracteres que iniciam as strings da lista:
["Consoante Maiuscula","Consoante Maiuscula","Consoante Maiuscula"]

c)Uma função que devolva a string que possui o maior número de vogais:
""
```

Entrada: “1 A B a b #” - teste para verificar a questão “b”
Saída:

```
Digite sua lista de string (palavras separadas por espaco): 1 A B a b #

a)Uma função para contar o número de caracteres que cada string possui sem repetir:
Item: 1 Tamanho: 1
Item: A Tamanho: 1
Item: B Tamanho: 1
Item: a Tamanho: 1
Item: b Tamanho: 1
Item: # Tamanho: 1



b)Uma função que devolve uma lista contendo os tipos de caracteres que iniciam as strings da lista:
["Digito","Vogal Maiuscula","Consoante Maiuscula","Vogal Minuscula","Consoante Minuscula","Caractere Especial"]

c)Uma função que devolva a string que possui o maior número de vogais:
"A"
```

3.5 Código 5

Com Listas de mesmo tamanho
Entrada: [1,2,3,4,5] - [6,7,8,9,10]
Saída:

```
ghci> principal([1,2,3,4,5],[6,7,8,9,10])
a) uma função que devolva uma lista contendo a união ordenada entre (A  B) e (B  A).
A: "[1,2,3,4,5]"
B: "[6,7,8,9,10]"

b)Uma função que devolva uma lista contendo a soma entre os quadrados dos elementos das duas listas
que forem maiores do que a soma entre o cubo dos dois primeiros elementos da lista.
"[]"
```

Primeira lista maior que a Segunda
Entrada: [1,2,3,4,5,6,7,8,9] - [1,3,5,7]
Saída:

```
ghci> principal([1,2,3,4,5,6,7,8,9],[1,3,5,7])
a) uma função que devolva uma lista contendo a união ordenada entre (A  B) e (B  A).
A: "[2,4,6,8,9]"
B: "[]"

b)Uma função que devolva uma lista contendo a soma entre os quadrados dos elementos das duas listas
que forem maiores do que a soma entre o cubo dos dois primeiros elementos da lista.
"[13,34,65]"
```

Segunda lista maior que a Primeira
Entrada: [4,6,8] - [1,2,3,4,5,6,7,8,9]
Saída:

```
a) uma função que devolva uma lista contendo a união ordenada entre (A  B) e (B  A).
A: "[]"
B: "[1,2,3,5,7,9]"

b)Uma função que devolva uma lista contendo a soma entre os quadrados dos elementos das duas listas
que forem maiores do que a soma entre o cubo dos dois primeiros elementos da lista.
"[73]"
```

Lista vazia
Entrada: [] - [1,2,3,4,5,6,7,8,9]
Saída:

```
ghci> principal([],[1,2,3,4,5,6,7,8,9])
a) uma função que devolva uma lista contendo a união ordenada entre (A  B) e (B  A).
A: "[]"
B: "[1,2,3,4,5,6,7,8,9]"

b)Uma função que devolva uma lista contendo a soma entre os quadrados dos elementos das duas listas
que forem maiores do que a soma entre o cubo dos dois primeiros elementos da lista.
"[]"
```

Lista vazia
Entrada: [1,2,3,4,5,6,7,8,9] - []
Saída:

```
ghci> principal([1,2,3,4,5,6,7,8,9],[])
a) uma função que devolva uma lista contendo a união ordenada entre (A  B) e (B  A).
A: "[1,2,3,4,5,6,7,8,9]"
B: "[]"

b)Uma função que devolva uma lista contendo a soma entre os quadrados dos elementos das duas listas
que forem maiores do que a soma entre o cubo dos dois primeiros elementos da lista.
"[]"
```

4. Conclusão

Como apresentado anteriormente, o experimento tratou-se de uma série de questões para serem resolvidas utilizando a linguagem Haskell, bem como os conceitos de programação funcional.

No fim, este trabalho teve o principal intuito de me aprimorar como aluno bem como avaliar meu conhecimento acerca das funcionalidades apresentadas, após muito trabalho para a finalização deste relatório posso afirmar que com o tempo e esforço que investi para responder as atividades propostas sinto que meu nível de conhecimento em haskell, ao menos nas funcionalidades mais básicas, se aprimoraram a ponto de conceitos que antes desconhecia ou não dominava bem agora estão bem mais claros.

5. Apêndice

Todos os códigos e demais arquivos relacionados a este relatório podem-se encontrar em anexo, isto é, foi enviado em conjunto com o relatório em forma compactada.