MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DO PIAUÍ – UFPI
CAMPUS SENADOR HELVÍDIO NUNES DE BARROS – PICOS
BACHARELADO EM SISTEMAS DE INFORMAÇÃO
PROGRAMAÇÃO FUNCIONAL
PROF. JULIANA OLIVEIRA DE CARVALHO

Relatório 1 - Programação Funcional

Aluno:
Leonardo Cristian Amorim Lopes

28 de Abril de 2021.

Relatório 1 - Programação Funcional

Relatório referente ao primeiro trabalho avaliativo da disciplina de Programação Funcional ministrada pela professora Juliana Oliveira de Carvalho.

Resumo

Este trabalho tem como objetivo a resolução de algumas questões de programação utilizando a linguagem de programação funcional Haskell. As questões abordam diferentes tipos de estruturas de códigos, como listas, geração de números aleatórios, manipulação de strings e funções recursivas. A fim de exercitar conhecimentos sobre o paradigma de programação funcional. Durante o relatório será apresentado 6 questões explicando o que deverá ser feito e, em seguida será mostrado sua solução com trechos de código seguido de sua explicação detalhada.

Questões abordadas

Esta seção aborda as questões e a lógica utilizada para resolvê-las.

Questão 2

Esta questão consiste em elaborar uma função para calcular o Mínimo Múltiplo Comum (MMC) entre três números. O cálculo do MMC corresponde ao menor número que é múltiplo de dois ou mais números ao mesmo tempo. Para obtermos esse valor, precisamos dividir os valores do cálculo pelos números primos e, em seguida multiplicá-los.

Em haskell, precisamos utilizar o Mínimo Divisor Comum MDC para calcular o MMC de 2 números e em seguida, utilizar o resultado do MDC para calcular o MMC. Para isso, criamos uma função MDC que realiza o cálculo com dois valores. Abaixo está o código e sua explicação.

```
mdc(x, y)
    | (x `mod` y ) == 0 = y
    | (y `mod` x) == 0 = x
    | (x > y) = mdc(y, (x `mod` y))
    | (y > x) = mdc(x, (y `mod` x))
```

Dois valores são passados por parâmetro, em seguida, retorna um valor dependendo de alguns casos. Caso o resto da divisão de X por Y seja 0, o valor do MDC entre X e Y é o valor de X. Do mesmo jeito para o resto da divisão de Y por X, caso o resultado seja 0, o valor do MDC é o X.

Nos outros casos, a função é chamada recursivamente. Porém, os parâmetros passados na chamada recursiva dependem da ordem dos valores, caso o X seja maior que o Y, o valor retornado é o resultado do cálculo do MDC passando Y como primeiro valor e o resto da divisão de X por Y como segundo parâmetro. O contrário acontece caso o Y seja maior que X. Feito isso, podemos criar a função MMC entre dois números. Seu código fica assim:

```
mmc(x, y)
    | x == 0 = 0
    | y == 0 = 0
    | x == y = x
    | otherwise = (x * y) `div` (mdc(x, y))
```

A função para calcular o MMC de dois números verifica se um deles é 0, se for, o resultado é 0. Se forem iguais, seu Mínimo Múltiplo Comum é qualquer um deles. Caso contrário, o resultado do mmc é a divisão entre a multiplicação dos dois valores pelo MDC dos mesmos. Após isso, podemos calcular o MMC de três números. Abaixo está o código para realizar este cálculo.

```
mmc_3_numeros(x, y, z) = mmc(x, mmc(y, z))
```

Para calcular o MMC entre 3 números, chamamos a função MMC explicada anteriormente passando X como primeiro parâmetro e o resultado do MMC entre Y e Z como segundo.

Questão 3

Esta questão consiste em calcular o número de ingressos que deveriam ser vendidos para que o produtor obtenha lucro. Esta questão utilizou duas funções para ser resolvida. Primeiro, foi criada uma função para realizar a venda de ingressos, em seguida, foi criada uma função para calcular o lucro no momento. A função calculaIngressos realiza a venda de ingressos até que o lucro seja maior ou igual ao custo da peça teatral.

```
vendeIngresso(valorIngresso, qtdPessoas) = (valorIngresso *
 qtdPessoas)

calculaIngressos(custo, valorIngresso, lucroMomento)
 | lucroMomento < custo = calculaIngressos(custo, valorIngresso
, lucroMomento+(vendeIngresso(valorIngresso, 300)))
 | otherwise = lucroMomento
```

Questão 5

A questão 5 consiste em realizar a leitura de uma lista de strings, em seguida realizar algumas operações de manipulação nessa lista. Obs: Uma string em haskell nada mais é do que uma lista de caracteres.

A alternativa “a” consiste em retornar o número de caracteres que cada string possui. Abaixo está o código seguido de sua explicação.

```
contaCaracteres([]) = 0
contaCaracteres(c:r) = contaCaracteres(r) + 1

contaCaracteres2([]) = []
contaCaracteres2((c:r1):r2) = contaCaracteres(c:r1):contaCaracteres2(r2)
```

Primeiramente , foi criada uma função chamada contaCaracteres, que retorna a quantidade de caracteres de apenas uma string. Após isso, foi criado a função contaCaracteres2. Esta função retorna uma lista, cada posição dessa lista será a quantidade de caracteres de cada string da lista passada como parâmetro. A função contaCaracteres é chamada passando a cabeça da lista inicial como parâmetro, ou seja, (c:r1), além de ser uma lista de caracteres, é também a cabeça da lista inicial (c:r1):r2.

Entendido isso, podemos assumir que r2 é também uma lista contendo cabeça e resto. O algoritmo cria uma lista contendo a contagem de caracteres da primeira string. Feito isso, a função é chamada recursivamente passando r2 como parâmetro que será dividido em cabeça e resto, ou seja, a lista vai ser percorrida sequencialmente e o resultado será uma lista contendo valores inteiros representando a quantidade de caracteres de todas as strings da lista inicial.

A alternativa B consiste em verificar o tipo de caractere de cada string da lista. Por exemplo, dígito, vogal, consoante, além disso, devemos identificar quando o caractere inicial da string é maiúsculo ou minúsculo. Abaixo está o algoritmo utilizado para resolver o problema e, em seguida, sua explicação.

```
tiposCaracteres([]) = []
tiposCaracteres((c:r1):r2)
 | (c == 'A' || c == 'E' || c == 'I' || c == 'O' || c == 'U')
 = "Vogal maiuscula":tiposCaracteres(r2)
 | (c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u')
 = "Vogal minuscula":tiposCaracteres(r2)
 | (c == '1' || c == '2' || c == '3' || c == '4' || c == '5'
|| c == '6' || c == '7' || c == '8' || c == '9' || c == '0')
= "Digito":tiposCaracteres(r2)
 | otherwise = "Consoante":tiposCaracteres(r2)
```

A função tiposCaracteres recebe uma lista de strings e retorna outra lista contendo o tipo de caractere de cada string da lista passada por parâmetro. O algoritmo verifica o caractere inicial de cada string da lista. primeiramente, a variável “c” representa o primeiro elemento da primeira string, com isso, podemos verificar se o mesmo é uma vogal maiúscula ou minúscula, caso seja uma vogal maiúscula, a string “Vogal maiuscula” é adicionada na lista resultante, e o resto da lista será a chamada recursiva da função passando r2 com parâmetro representando as outras

strings. Da mesma forma, para os outros casos, caso seja um dígito, será adicionado a string “digito” na lista resultante. O algoritmo percorre todas as posições da lista inicial, mas verivica apenas a primeira posição de cada string. Ao final, o resultado será uma lista de string onde cada string será o tipo de caractere inicial de cada posição da lista.

A alternativa C consiste em retornar a string que possui a maior quantidade de vogais. Para resolver este problema, foi necessário utilizar duas funções, uma para contar a quantidade de vogais de uma string e outra para percorrer toda a lista de strings. A função para realizar a contagem de vogais é descrita abaixo.

```
contaVogais([]) = 0
contaVogais(c:r)
 | (c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u')
 = contaVogais(r) + 1
 | (c == 'A' || c == 'E' || c == 'I' || c == 'O' || c == 'U')
 = contaVogais(r) + 1
 | otherwise = contaVogais(r)
```

A função contaVogais percorre uma lista recursivamente e verifica se cada posição corresponde a uma vogal, sendo ela maiúscula ou não. Após isso, foi criada uma função para percorrer toda a lista de strings, abaixo está seu código e a explicação.

```
maiorStringVogais([], maiorTam, maiorString) = maiorString
maiorStringVogais((c:r1):r2, maiorTam, maiorString)
 | contaVogais(c:r1) > maiorTam = maiorStringVogais(r2, contaVogais(c:r1), c:r1)
 | otherwise = maiorStringVogais(r2, maiorTam, maiorString)
```

A função maiorStringVogais recebe três parâmetros, uma lista, e duas variáveis, estas variáveis serão responsáveis por controlar qual string está sendo a maior naquele momento. A variável maiorTam inicia com 0, sendo este o maior tamanho encontrado até o momento. E a variável maiorString iniciará como vazia, ou seja, “” representa a maior string até o momento.

Na execução, o algoritmo chama a função descrita anteriormente para contar as vogais da primeira posição da lista e verifica se esta é maior que o conteúdo da variável maiorTam. Caso verdadeiro, é realizada uma chamada recursiva passando o restante da lista, a quantidade de vogais daquela string e a própria string como sendo a maior, e caso seja falso é passado os mesmos parâmetros iniciais, porém, r2 é passado como sendo o restante da lista. Com isso, ao final da execução o algoritmo retorna a string com maior número de vogais

Questão 6

A quesão 6 consiste em realizar algumas operações de manipulação de listas.
A alternativa “a” consiste em retornar uma lista contendo a união ordenada entre a diferença de duas listas.Primeiramente, foi criada a função difListas que é responsável por retornar uma lista contendo a diferença de duas outras listas passadas por parâmetro.

```
difListas([], []) = []
difListas(l1, []) = []
difListas([], l2) = []
difListas(c1:r1, c2:r2)
 | c1 /= c2 = c1:difListas(r1, c2:r2)
 | c1 < c2 = difListas(r1, c2:r2)
 | otherwise = difListas(r1, r2)
```

A função verifica se os elementos da cabeça das duas listas são diferentes e, se forem, coloca o valor da cabeça da primeira lista em uma outra lista resultante, e uma chamada recursiva é feita para preencher o resto. Na chamada recursiva, apenas a lista 1 é percorrida, pois cada elemento da lista 1 vai ser comparado com todos os elementos da lista 2.

Caso o elemento da primeira lista seja menor que o da segunda, significa que ele não pode estar mais dentro da lista 2, então a chamada é feita apenas percorrendo a lista 1. Caso contrário as duas listas serão percorridas e, ao final, o resultado será uma lista contendo a diferença entre a primeira e a segunda passada por parâmetro, ou seja, os valores da lista 1 que não estão na lista 2.

Após isso, foi criado uma função para retornar a união das duas listas.
Em haskell, podemos unir duas listas usando o operador “++”.