Universidade Federal do Piauí – UFPI
Campus Senador Helvidio Nunes de Barros – CSHNB
Sistemas de informação – Programação Funcional – Prof. Juliana Oliveira

Relatório do segundo trabalho de programação funcional

Aluno: Marcos Vinicius Batista Sampaio

Resumo:

Este relatório apresenta o desenvolvimento dos códigos necessários para que as questões requeridas fossem respondidas. Para isso foram utilizados funções de alta ordem alem de listas comuns e comprehension em haskell.

Introdução:

Como relatado no título, o principal paradigma usado é o funcional que é visto na disciplina de programação funcional. Com questões sendo desenvolvidas na linguagem Haskell, também muito utilizada em universidades para introdução do paradigma, o relatório tem a seguir: Seções específicas onde discutirei mais sobre como as questões foram resolvidas, apresentando o funcionamento, Resultados que mostra as saídas das questões de acordo com o explicado e pedido no trabalho.

Seções Específicas:

Questão 1: A questão pedia para ler uma lista de strings e então devolver uma tupla com a string e a quantidade de vogais, uma lista com strings maiores que 5 e que inicia com vogais e uma lista substituindo vogais por @. Foram necessárias 4 funções que faziam o pedido, alem da main e uma auxiliar para pegar as lista de string, sendo:

A função `contarVogais` recebe uma String como argumento e retorna um Int que representa a contagem de vogais na string. A implementação é feita usando uma list comprehension que filtra todos os caracteres da string str que são vogais (letras "a", "e", "i", "o", "u" em minúsculas e maiúsculas) e, em seguida, retorna o comprimento dessa lista.

A função `listaContarVogais` recebe uma lista de String como argumento e retorna uma lista de tuplas (String, Int) que representa a contagem de vogais em cada uma das strings da lista de entrada. A implementação é feita usando uma list comprehension que itera sobre a lista de strings de entrada strList e, para cada string str, cria uma tupla (str, contarVogais str) onde o segundo elemento da tupla é a contagem de vogais da string str calculada pela função contarVogais. Ao final, a lista de tuplas resultante é retornada.

A função `listaLongaVogaisStrings` recebe uma lista de String como argumento e retorna uma lista de strings que têm comprimento maior que 5 e começam com uma vogal em minúsculas. A implementação é feita usando uma list comprehension que itera sobre a lista de strings de entrada strList e, para cada string str, verifica se seu comprimento é maior que 5 e se seu primeiro caractere (convertido para minúsculo) é uma vogal. Se ambas as condições forem satisfeitas, a string str é adicionada à lista de saída.

A função `refazerVogais` recebe uma lista de String como argumento e retorna uma nova lista de strings com as vogais substituídas pelo caractere '@'. A implementação é feita usando uma list comprehension que itera sobre a lista de strings de entrada strList e, para cada string str, aplica a função map para cada caractere da string str. A função map recebe uma função lambda que verifica se o caractere atual é uma vogal em minúsculo ou maiúsculo. Se for, a função lambda retorna o caractere '@', caso contrário, retorna o próprio caractere.

A função `getStrings` é uma função que usa recursão para ler uma lista de strings do usuário. A função lê uma string da entrada padrão usando a função getLine, e se a string for vazia (isto é, se o usuário inserir uma linha em branco), a função retorna uma lista vazia. Caso contrário, a função chama a si mesma recursivamente para ler o restante da lista e retorna uma lista que contém a string lida concatenada com a lista restante.

Questão 2: A questão pede para ler duas lista de string e então, retornar uma lista ordenada contendo strings das duas listas >= a 4. Devolver a quantidade de strings das duas listas que iniciam comvogais e uma lista contendo strings da duas lista que iniciam e terminam com vogal. Foi necessário 8 funções além da main e uma auxiliar para ler as listas.

A função `ehVogal` recebe um caractere e retorna um booleano indicando se ele é uma vogal ou não.

A função `contarVogais` recebe uma string e conta quantas vogais ela possui, usando a função `ehVogal` para verificar se cada caractere é uma vogal ou não.

As funções `filtrarTamanho`, `filtrarVogalInicio` e `filtrarVogalInicioEFim` recebem uma lista de strings e retornam uma lista filtrada com as strings que satisfazem determinados critérios: tamanho maior ou igual a 4, começar com vogal e começar e terminar com vogal, respectivamente. Todas elas usam a função filter combinada com funções lambda que verificam os critérios.

A função `contarVogalInicio` recebe duas listas de strings e retorna o número de strings nas duas listas que começam com vogal, usando a função `filtrarVogalInicio` para filtrar as strings.

A função `vogalInicioEFimLista` recebe duas listas de strings e retorna uma lista de strings que começam e terminam com vogal, usando a função `filtrarVogalInicioEFim` para filtrar as strings.

A função `mergeAndSort` recebe duas listas de strings, as concatena, filtra as strings com tamanho maior ou igual a 4 e retorna uma lista ordenada com as strings resultantes.

A função `getStrings` funciona igualmente a da explicada na questão1.

Questão 3: pedia para que leia duas listas ordenadas de inteiros e então, devolva duas listas, a primeira contendo os números das posições pares maiores do que 50 e a segunda os elementos ímpares menores que 200, devolva o produto dos elementos das duas listas dos múltiplos de 3 > 50 e dos múltiplos de 7 menos do que 200 e Devolva uma lista ordenada contendo elementos das duas listas que sejam maiores do que 50 e que sejam ímpares múltiplos de 3. Foram necessarias 5 funções além da main.

A função `quicksort` é uma implementação do algoritmo de ordenação quicksort para ordenar uma lista de inteiros.

A função `paresMaioresQue50` recebe uma lista de inteiros e retorna uma lista contendo apenas os elementos de posições pares maiores que 50. A função utiliza a função zip para criar uma lista de tuplas contendo os elementos da lista e suas respectivas posições. Em seguida, a lista de tuplas é filtrada usando a condição de que a posição seja par e o elemento seja maior que 50.

A função `imparesMenoresQue200` recebe uma lista de inteiros e retorna uma lista contendo apenas os elementos de posições ímpares menores que 200. A lógica é semelhante à função anterior, com a diferença de que a condição de filtro é a posição ser ímpar e o elemento ser menor que 200.

A função `produtoMultiplos` recebe duas listas de inteiros e retorna o produto dos elementos que são múltiplos de 3 e maiores que 50 na primeira lista, e múltiplos de 7 e menores que 200 na segunda lista. A função utiliza a função filter para selecionar os múltiplos de 3 e 7 em cada lista e, em seguida, aplica a função product para calcular o produto dos elementos.

A função `elementosMaiores50Multiplos3` recebe uma lista de inteiros e retorna uma lista contendo apenas os elementos maiores que 50, ímpares e múltiplos de 3. A função utiliza a função filter para selecionar os elementos que atendem a essas condições. A condição de ser ímpar é verificada com a função odd. A condição de ser múltiplo de 3 é verificada com a expressão mod x 3 == 0.

Questão 4: Pede para ler uma lista de times de futebol contendo nome do clube, estado e país a qual pertence e ano de fundação do clube e então faça uma função que ordene a lista pelo campo nome do clube usando o Quicksort, e depois possibilite ao usuário ver toda a lista e permita também o usuário buscar informações de um clube pelo nome do clube. Foram necessário 3 funções com o quicksort e uma função para entrada.

A função `quicksort` é uma implementação do algoritmo de ordenação rápida que funciona para qualquer tipo que seja uma instância da classe Ord.

A função `ordenarPorNome` é uma aplicação da função quicksort para uma lista de elementos do tipo Time, ordenando-os pelo nome. Isso é feito através da composição de quicksort com a função sortBy (comparing nome), que usa a função comparing para comparar os nomes dos times.

A função `buscarPorNome` recebe uma lista de Time e um nome a ser buscado e retorna uma lista contendo todos os elementos que possuem o nome buscado. Isso é feito usando a função filter para filtrar os elementos que têm o mesmo nome e retornando a lista resultante.

A função `getTimes` é utilizada para adicionar novos times à lista. Ela recebe um inteiro n que representa a quantidade de times a serem adicionados. Em cada chamada recursiva, ela solicita ao usuário as informações de um time (nome, estado, país e ano de fundação) e adiciona o time à lista, até que n times tenham sido adicionados.

Questão 5: Cadastro as listas lunos(Matrícula, Nome, Curso, Período), curso(Código, Nome, Quantidade de Períodos), Disciplinas(Código da Disciplina, Código do Curso, Nome Disciplina, Período) e Notas(Matricula, Código Disciplina, Nota1, Nota2). Dez funções foram necessarias de acordo com cadastro e buscas.

A função `cadastrarCurso` recebe uma lista de cursos e um curso para cadastrar. Se o curso já existe na lista, a lista original é retornada sem alterações. Caso contrário, o curso é adicionado à lista e a nova lista é retornada.

A função `cadastrarDisciplina` recebe uma lista de cursos, uma lista de disciplinas e uma disciplina para cadastrar. Para a disciplina ser cadastrada, é necessário que exista um curso na lista de cursos com o código do curso da disciplina e que não exista uma disciplina na lista de disciplinas com o mesmo código da disciplina a ser cadastrada. Se essas condições são satisfeitas, a disciplina é adicionada à lista de disciplinas e a nova lista é retornada. Caso contrário, a lista original de disciplinas é retornada.

A função `cadastrarAluno` recebe uma lista de cursos, uma lista de alunos e um aluno para cadastrar. Para o aluno ser cadastrado, é necessário que exista um curso na lista de cursos com o

código do curso do aluno e que não exista um aluno na lista de alunos com a mesma matrícula do aluno a ser cadastrado. Se essas condições são satisfeitas, o aluno é adicionado à lista de alunos e a nova lista é retornada. Caso contrário, a lista original de alunos é retornada.

A função `cadastrarNota` recebe uma lista de alunos, uma lista de disciplinas, uma lista de notas e uma nota para cadastrar. Para a nota ser cadastrada, é necessário que exista um aluno na lista de alunos com a matrícula do aluno da nota, uma disciplina na lista de disciplinas com o código da disciplina da nota e que não exista uma nota na lista de notas com a mesma matrícula do aluno e o mesmo código de disciplina da nota a ser cadastrada. Se essas condições são satisfeitas, a nota é adicionada à lista de notas e a nova lista é retornada. Caso contrário, a lista original de notas é retornada.

A função `buscarCursos` simplesmente imprime a lista de cursos recebida como parâmetro. Nesse caso, não há necessidade de filtragem ou ordenação, então a lista é simplesmente impressa como está.

As funções `buscarAlunosPorCurso`, `buscarDisciplinasPorCurso` e `buscarNotasPorAluno` recebem um código como parâmetro e usam a função filter para filtrar a lista de alunos, disciplinas ou notas respectivamente, mantendo apenas os que pertencem ao curso ou aluno correspondente ao código recebido. Se a lista resultante for vazia, é impressa uma mensagem indicando que não há dados cadastrados para aquele código.

As funções `buscarAlunosPorPeriodo` e `buscarDisciplinasPorPeriodo` funcionam de maneira semelhante, mas filtram a lista de alunos ou disciplinas pelo período em que estão matriculados. Novamente, se a lista resultante for vazia, é impressa uma mensagem indicando que não há dados cadastrados para aquele período.

No menu, quando escolhido a função para cadastrar nota. Se nota2Str estiver vazio, atribui -1 a nota2. Caso contrário, converte nota2Str em um valor do tipo Float e armazena em nota2.

Resultados:

Questão 1:

Entrada	Lista de tuplas (string, número de vogais):	Lista de strings com tamanho maior do que 5 e que iniciam com vogais:	Lista de strings com vogais substituídas por @:
Amanda cachorro abelha animal	[("amanda",3), ("cachorro",3), ("abelha",3), ("animal",3)]	["amanda","abelha","animal"]	["@m@nd@","c@ch@rr@","@b@lh@","@n@m@l"]
urubu aviao helicoptero universidade coperativa silhueta	[("urubu",3), ("aviao",4), ("helicoptero",5), ("universidade",6), ("coperativa",5), ("silhueta",4)]	["universidade"]	["@r@b@","@v@@@","h@l@c@pt@r@","@n@v@rs@d@d@","c@p@r@t@v@","s@lh@@t@"]
[("hotel",2), ("cadeira",4), ("astuto",3), ("jogos",2)]	[("hotel",2), ("cadeira",4), ("astuto",3), ("jogos",2)]	["astuto"]	["h@t@l","c@d@@r@","@st@t@","j@g@s"]

Questão 2:

Entrada 1	Entrada 2	lista ordenada contendo strings das duas listas onde as mesmas deve ter tamanho >= a 4:	Quantidade de strings das duas listas que iniciam com vogais:	Lista contendo strings das duas listas que iniciam e terminam com vogal:
aviao elefante coisa silhueta comida	helicoptero burro universidade	["aviao","burro","coisa","comida","elefante","helicoptero","silhueta","universidade"]	3	["aviao","elefante","universidade"]
corrida casa avenida posto bairro	cidade pais familia unidade centena	["avenida","bairro","casa","centena","cidade","corrida","familia","pais","posto","unidade"]	2	["avenida","unidade"]

Questão 3:

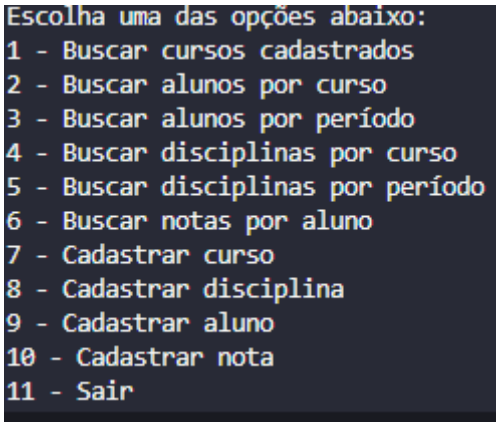
Entrada 1	Entrada 2	Elementos das posições pares maiores que 50:	Elementos das posições impares menores que 200:	Produto dos elementos das duas listas dos múltiplos de 3 > 50 e dos múltiplos de 7 menos do que 200:	lista ordenada contendo elementos das duas listas que sejam maiores do que 50 e que sejam ímpares múltiplos de 3
1 5 15 22 25 29	2	[]	[5,22,29]	1	[]
1 2 5 4 8 9 50 55 58 60 67 69 70 72 74	[2,5,9,55,60,69,72,6,10,29,48,95]	[58,67,70,74,92]	[2,5,9,55,60,69,72,6,10,29,48,95]	1	[69]

Questão 4:

Quantidade	Entrada	Lista de times	Busca	resultado
2	Flamengo – rio de janeiro – brasil – 1985 vasco – rio de janeiro – brasil – 1910	flamengo vasco	vasco	Time {nome = "vasco", estado = "rio de janeiro", pais = "brasil", anoFundacao = 1910}
4	Flamengo – rio de janeiro – brasil – 1985	flamengo palmeiras sao paulo	palmeiras	Time {nome = "palmeiras", estado = "sao

	vasco – rio de janeiro – brasil – 1910 são paulo – são paulo – brasil – 1910 palmeiras – são paulo – brasil - 1915	vasco		paulo", pais = "brasil", anoFundacao = 1915}

Questão 5:
Devido a complexidade de coletar os resultados da questao 5 será colocado prints:



Ação	resultado
7	Digite o código do curso: 1 Digite o nome do curso: si Digite a quantidade de períodos: 8 Curso cadastrado com sucesso!
1	Lista de cursos cadastrados: Curso {codigo = 1, nome = "si", quantidadePeriodos = 8}
8	8 Digite o código da disciplina: 1 Digite o código do curso: 1 Digite o nome da disciplina: si1 Digite o período: 1 Disciplina cadastrada com sucesso!
9	Digite a matrícula do aluno: 111 Digite o nome do aluno: marcos

Digite o código do curso do aluno:
1
Digite o periodo do aluno:
1
Aluno cadastrado com sucesso!
3 Digite o período:
1
Lista de alunos do período:
Aluno {matricula = 111, nomeAluno = "marcos", codigoCursoAluno = 1, periodoAluno = 1}
10 Digite a matrícula do aluno:
111
Digite o código da disciplina:
1
Digite a nota 1:
8
Digite a nota 2:

Nota cadastrada com sucesso!
6 Digite a matrícula do aluno:
111
Lista de notas do aluno:
Nota {matriculaAluno = 111, codigoDisciplinaNota = 1, nota1 = 8.0, nota2 = -1.0}

Conclusão:

O desenvolvimento dos códigos teve desafios enormes desde funções dando erro ou um nive l de complexidade maior com a manipulação de estruturas com listas, porém foi possível resolver todos os problemas conforme o trabalho especificou.