1ª Lista de Exercícios de Paradigmas de Linguagens Computacionais
Professores: Fernando Castor, Paulo Borba e Márcio Cornélio
Monitores:

Ciência da computação: Agay Borges (abn), David Hulak (dbh), Glauco Santos (grps), Lívia Vilaça (lcjbv), Luís Gabriel Lima (lgnfl), Pedro Melo (pam2), Thales Alex (tata), Vanessa Gomes de Lima (vgl2), Walter Ferreira (wflf)

Engenharia da Computação: Dennis Silveira (dwas), Eduardo Rocha (ejfrf), Felipe de Souza (fsa2),Gabriel Avelar (gafm), Lucas Inojosa (licf), Rodrigo Folha (rbf2), Victor Hugo (vhca), Wellington Oliveira (woj)

CIn-UFPE – 2011.2
Disponível desde: 16/08/2011
Entrega: 30/08/2011

A lista deverá ser respondida **em dupla**. A falha em entregar a lista até a data estipulada implicará na perda de 0,25 ponto na **média** da disciplina para os membros da dupla. Considera-se que uma lista na qual **menos que 8** das respostas estão corretas não foi entregue. A entrega da lista com **pelo menos 15** das questões corretamente respondidas implica em um acréscimo de 0,125 ponto na média da disciplina para os membros da dupla. Se **qualquer situação de cópia de respostas** for identificada, os membros **de todas as duplas envolvidas** perderão 0,5 ponto na média da disciplina. O mesmo vale para respostas obtidas a partir da Internet. As respostas deverão ser entregues **exclusivamente em formato texto ASCII** (nada de .pdf, .doc, .docx ou .odt) e deverão ser enviadas para o monitor responsável por sua dupla, **sem** cópia para o professor. Devem ser organizadas em arquivos separados, um por questão, entregues em um único formato compactado, ou seja, um único arquivo zipado contendo as respostas para todas as questões.

1) Crie uma função que, dada uma lista de inteiros e um inteiro qualquer, retorne todos os inteiros que não são divisíveis por esse inteiro passado como parâmetro.

-Exemplos:

```
Prelude> primosEntreSi [5,8,10,32,56,73] 4
[5,10,73]
```

2) Crie uma função que, dados uma lista de String e um padrão (uma String), retorne todos os elementos da lista com suas letras trocadas (maiúsculas por minúsculas e vice-versa) e na ordem inversa, menos o padrão. Obs.: É proibido usar a função reverse.

-Exemplos:

```
Prelude> inverteRetira ["oi","pEdRo","cAMa"] "oi"
["CamA", "PeDrO"]
```

3) Defina uma função `splitChars :: String -> (Uppercase, Lowercase, Int, Remainder)` que lê uma String e retorna uma tupla com 4 elementos. Esses elementos são:
1 - Uppercase: as letras maiúsculas da String, ignorando-se repetição.
2 - Lowercase: as letras minúsculas da String, ignorando-se repetição.
3 - Int: a soma dos dígitos que aparecem na String.
4 - Remainder: os caracteres que não forem letras nem dígitos.
Represente Uppercase, Lowercase e Remainder através de tipos sinônimos de Haskell.

-Exemplos:

```
Prelude> :t splitChars
splitChars :: String -> (Uppercase, Lowercase, Int, Remainder)
Prelude> splitChars "Be4Ever-sK8!"
("BEK","vers",12,"-!")
```

4) Defina uma função `comparaConjuntos :: (Eq t) => [t] -> [t] -> String` que responda se o primeiro conjunto A contém o segundo conjunto B, se B contém A, se há interseção entre eles, se eles são disjuntos ou se eles são iguais. Caso A contenha B, a saída deve ser "A contem B"; caso B contenha A, a saída deve ser "B contem A"; caso haja interseção, mas nenhum conjunto contenha o outro, a saída deve ser "A interseciona B"; caso não haja nenhum elemento em comum, a saída deve "Conjuntos disjuntos"; caso os conjuntos sejam iguais, a saída deve ser "A igual a B".

Obs1: Nos conjuntos, a ordem e a quantidade de vezes que os elementos estão listados na coleção não é relevante.

Obs2: O "(Eq t) => " na assinatura da função comparaConjuntos indica que a função é polimórfica, conforme visto em aula, mas que é possível comparar elementos do tipo t para verificar se são iguais (operador "==") ou diferentes ("/="). Se for omitido, Haskell supõe que t pode ser qualquer tipo mas nem sempre é possível comparar valores de um mesmo tipo em Haskell.

-Exemplos:

```
Prelude> :t comparaConjuntos
comparaConjuntos :: (Eq t) => [t] -> [t] -> String
Prelude> comparaConjuntos [1,2,3,4,5] [3,4]
"A contem B"
Prelude> comparaConjuntos "banana" "nanaba"
"A igual a B"
Prelude> comparaConjuntos "caco" "macaco"
"B contem A"
Prelude> comparaConjuntos [True] [False]
"Conjuntos disjuntos"
Prelude> comparaConjuntos ['b','o','l','a'] ['b','u','l','e']
"A interseciona B"
```

5) Crie uma função `somaDigitos` que recebe um número inteiro não negativo e retorna a soma dos dígitos desse número.

-Exemplos:

```
Prelude> somaDigitos 42
6
Prelude> somaDigitos 1337
14
```

6) Um número quadrado perfeito é um número inteiro não negativo que possui raiz inteira. Crie uma função `listaQuadradosPerfeitos` que recebe um inteiro não negativo N e retorna uma lista de números inteiros com os números de 0 a N que são quadrados perfeitos.

-Exemplos:

```
Prelude> listaQuadradosPerfeitos 100
[0,1,4,9,16,25,36,49,64,81,100]
Prelude> listaQuadradosPerfeitos 35
[0,1,4,9,16,25]
```

7) Considere a sequência de notas musicais, da nota mais grave para a mais aguda: C, D, E, F, G. Um intervalo é o espaço que separa um som do outro. Ele pode ser:

1 – ascendente (a primeira nota é mais grave do que a segunda) , descendente (a primeira nota é mais aguda do que a segunda) ou uníssono (as duas notas são iguais);

2 – conjunto (formado por notas consecutivas) ou disjunto (formado por notas não-consecutivas). A nota conjunta é a nota mais próxima, tanto para cima quanto para baixo.

Um intervalo também possui um número de notas contidas no intervalo (de 1 a 5). Sua tarefa será criar uma função `classificarIntervalo :: Intervalo -> Classificacao`, que recebe um Intervalo (tipo sinônimo de uma 2-upla de Chars) e retorna uma Classificacao (tipo sinônimo de uma 3-upla contendo duas Strings e um Int). O primeiro item da classificação é referente à definição 1, o segundo item à definição 2 e o terceiro item dá como resposta o número de notas contido no intervalo.

-Exemplos:

```
Prelude> classificarIntervalo ('D', 'D')
("unissono","disjunto",1)
Prelude> classificarIntervalo ('E', 'G')
("ascendente","disjunto",3)
Prelude> classificarIntervalo ('G', 'F')
("descendente","conjunto",2)
```

8) Faça uma função `simulacaoLuta :: [Personagem] -> [Ataque] -> [Personagem]` que simula uma luta entre personagens. Cada Personagem é formado por (Nome, XP), em que XP é um tipo sinônimo de Int e Nome é um tipo sinônimo de String. As lutas são simuladas através de uma lista de Ataques. Um Ataque consiste de uma 3-upla, contendo o nome do personagem que ataca, o que recebe o dano e a experiência relacionada ao ataque, respectivamente. Ou seja, Ataque é uma 3-upla (Nome, Nome, XP).

Para cada ataque, deve-se somar a experiência ao valor XP de quem desfere o golpe e deve-se subtrair a experiência do valor XP de quem recebe o golpe. O retorno da função é a lista final de Personagens, ordenada de quem tem mais XP para quem menos XP. Caso haja empate de XP, o personagem vencedor é aquele que possui o maior valor lexicográfico, ou seja, “vem depois no dicionário”.

-Exemplos:

```
Prelude>simulacaoLuta [("Sephiroth", 250)::Personagem, ("Ganondorf", 30)::Personagem, ("Bowser", 40)::Personagem] [("Bowser", "Sephiroth", 10)::Ataque, ("Ganondorf", "Sephiroth", 20)::Ataque, ("Ganondorf", "Bowser", 100)::Ataque]

[("Sephiroth",220),("Ganondorf",150),("Bowser",-50)]

Prelude> simulacaoLuta [("Chuck Norris", 10000)::Personagem, ("Gold Roger", 100000)::Personagem, ("Lindomar", 5000)::Personagem, ("Kirby", 150)::Personagem]
[("Lindomar", "Chuck Norris", 5000)::Ataque, ("Gold Roger", "Kirby", 15)::Ataque,
```

```
("Gold Roger", "Lindomar", 20)::Ataque, ("Gold Roger", "Chuck Norris",
20)::Ataque, ("Chuck Norris", "Gold Roger", 5000)::Ataque]

[("Gold Roger",95055),("Lindomar",9980),("Chuck Norris",9980),("Kirby",135)]
```

9) Natália é professora do jardim de infância e intentou um novo jogo. Este jogo não apenas desenvolve a parte física das crianças, mas também as ensina a contar.

O jogo é o seguinte. As crianças ficam em pé lado-a-lado formando um círculo. Vamos considerar que as crianças são numeradas de **1** a *n* em sentido horário e a criança número **1** está segurando a bola. Inicialmente a primeira criança joga a bola para a próxima criança do círculo no sentido horário, ou seja, a criança número **2**. Então a criança número **2** lança a bola para o próximo mais um, ou seja, para criança número **4**, então a quarta criança lança a bola para a criança que está duas crianças após ela mesma, isto é, a criança número **7**. Depois a bola é lançada para a criança que está quatro crianças após a última, e assim por diante. Vale salientar que quando a bola é lançada pode passar para o início do círculo. Por exemplo, se *n* = **5**, após a terceira criança lançar, a criança número **2** terá a bola novamente. No total *n*-**1** lançamentos são feitos, e o jogo termina.

O problema é que não são todas as crianças que pegam a bola durante o jogo. Se uma criança não pegar a bola, ela fica muito chateada e chora até que Natália lhe dê um doce. Por isso, Natália pede para você ajudá-la a identificar o número das crianças que irão pegar a bola após cada lançamento.

Sua tarefa será criar uma função em Haskell `ballGame :: Int -> [Int]` que recebe como entrada o número de crianças que irão jogar uma partida do jogo e retorna uma lista com *n*-**1** números que são os números das crianças que pegarão a bola após cada lançamento.

Fonte: codeforces.com/problemset/problem/46/A

-Exemplos:

```
Prelude> ballGame 10
[2,4,7,1,6,2,9,7,6]
Prelude> ballgame 3
[2,1]
```

10) A notação pós-fixada, ou *Reverse Polish notation* é uma notação matemática onde cada operador de uma expressão se encontra após todos seus operandos, em contraste com a notação pré-fixada, ou *Polish notation*, onde os operadores aparecem antes dos operandos. Uma característica importante dessa notação é que ela é livre de parênteses enquanto as aridades dos operadores forem fixas.

Essa notação foi bastante usada por calculadoras financeiras da HP e é de fundamental importância para computação por tornar possível a utilização da pilha para avaliar expressões.

Sua tarefa será escrever uma função `onp :: String -> String` que transforma expressões algébricas com parênteses em expressões na forma RPN. Todos os operadores envolvidos são binários: +, -, *, /, ^ (ordem de prioridade do menor para o maior). Os operandos serão exclusivamente letras do alfabeto de a à z. Assuma que haverá apenas uma forma RPN (sem expressões como a*b*c).

Fonte: spoj.pl/problems/ONP/

-Exemplos:

```
Prelude> onp "a+(b*c)"
```

```
"abc*+"
Prelude> onp "(a+b)*(z+x)"
"ab+zx+*"
Prelude> onp "(a+t)*((b+(a+c))^(c+d))"
"at+bac++cd+^*"
```

11) Crie uma função `ordenarLista :: [String] -> [String]` que recebe uma lista de Strings e retorna uma lista lexicograficamente ordenada, que contém somente os caracteres que sejam letras maiúsculas de cada String.

-Exemplos:

```
Prelude> ordenarLista ["Ola, tuDo Bem?", "TuDO cerTo, e conTiGO?", "BeLEZa"]
["BDO","DGOOTTT","BELZ"]
Prelude> ordenarLista ["123456"]
[""]
Prelude> ordenarLista ["Ola, tuDo Bem?", "TuDO cerTo, e conTiGO?", "BeLEZa", "A1,
Z3 e f7"]
["BDO","DGOOTTT","BELZ","AZ"]
```

12) Os monitores da disciplina de PLC perceberam que alguns alunos tem tentado capturar as conversas nos emails entre os monitores e os professores das cadeiras do CIn. Os professores se reuniram e decidiram que, a partir dessa reunião, conversariam de uma forma mais segura, e um deles sugeriu que os emails, antes de serem enviados, deveriam ser criptografados com a cifra de césar. Implemente a função `cifraCesar :: [Char] -> Int -> [Char]` para ajudar os professores e monitores a manter suas conversas mais seguras.
Obs: o inteiro deve ser não-negativo.

-Exemplos:

```
Prelude> cifraCesar “abcdefghijklmnopqrstuvwxyz” 1
“bcdefghijklmnopqrstuvwxyza”
Prelude> cifrCesar “mensagem secreta” 13
“zrafntrz frpergn”
```

13) Um jogador da Mega Sena queria testar sua sorte, e decidiu implementar uma função em Haskell para chutar números aleatórios e ver se batiam com alguns preestabelecidos por ele. Para ajudá-lo, implemente a função `countEquals :: (Eq t) => [t] -> [t] -> Int` que dada duas listas, compara-as e conta quantos elementos elas tem em comum.

-Exemplos:

```
Prelude> countEquals [1..10] [2,4,6,8,10]
5
Prelude> countEquals [“olá”, “tudo”, “bem”, “como”, “vai”,”?”] [“Joseph”,
“Klimber”, “haha”]
0
```

14) Construa a função `produtoCartesianoEspecial :: [Int] -> [Int] -> ([(Int, Int)], Int)`, que recebe dois conjuntos e retorna uma tupla onde o primeiro elemento é o

produto cartesiano dos conjuntos e o segundo elemento é a soma dos números pares de todas as tuplas. Um conjunto será representado por uma lista de inteiros, e o produto cartesiano será uma lista de tuplas.

-Exemplos:

```
Prelude> produtoCartesianoEspecial [1, 2] [2, 3, 4]
([(1,2),(1,3),(1,4),(2,2),(2,3),(2,4)],18)
Prelude> produtoCartesianoEspecial [1, 9] [7, 3, 5]
([(1,7),(1,3),(1,5),(9,7),(9,3),(9,5)],0)
Prelude> produtoCartesianoEspecial [] [2, 4, 6]
([],0)
Prelude> produtoCartesianoEspecial [3, 4, 2] []
([],0)
```

15) Kiana pensa que dois inteiros são amigos se e somente se um deles divide o outro. Por exemplo, 12 e 4 são amigos, 6 e 6 são amigos também, mas 120 e 36 não são.

Um grupo de inteiros maiores que zero é chamado de amigo, se cada par de seus inteiros formam um par amigo.

Dado uma lista de inteiros faça uma função `friendship :: [Int] -> Bool` que retorne True se esses inteiros formam pares amigáveis e False caso contrário.

Fonte: codeforces.com/problemset/problem/100/B

-Exemplos:

```
Prelude> friendship [1, 4, 2, 12]
True
Prelude> friendship [1, 16, 3, 9]
False
```

16) Dado uma String criptografada, faça uma função `decode :: String -> String` que decodifica essa String. Para decodificar, a função `decode` deve remover os caracteres repetidos da String. O primeiro caractere encontrado é parte do código, entretanto os demais repetidos devem ser removidos.

-Exemplos:

```
Prelude> decoder ['c','o','o','d','e','d', 'o']
"code"
```

17) Implemente a função `somatorioHexadecimal`. Essa função recebe uma lista de Strings onde cada elemento representa um numero na base hexadecimal e retorna uma String contendo o resultado em hexadecimal do somatório da primeira lista.

-Exemplos:

```
Prelude> somatorioHexadecimal ["1","2","3","4","5"]
"F"
Prelude> somatorioHexadecimal["A","B","C","D","E","F"]
"4B"
```

18) Defina um tipo `Planeta` que deverá ter nome, massa e volume, e depois crie funções binárias para calcular o planeta maior em volume, o planeta de maior massa e uma função que, dados dois planetas e a distancia entre os centros de massa dos dois, retorne a força gravitacional entre os dois.

19) Defina a função `avaliarExpressoes` que recebe como parâmetro uma lista de tuplas representando expressões aritméticas e retorna uma lista de inteiros contendo os resultados das avaliações das expressões da lista. Cada expressão é representada por uma tupla contendo três elementos, na seguinte ordem: um inteiro representando o operando à esquerda da operação, um caracter representando a operação ('+', '-', '*','/'), e outro inteiro representando o operando à direita. Explique sua decisão de criar ou não funções auxiliares para implementar `avaliarExpressoes`.

20) Defina a função `somaTotal` que recebe uma lista de expressões, como na questão anterior, e retorna como resultado a soma das avaliações de todas as expressões da lista, mas evitando avaliar as divisões por zero. Explique sua decisão de criar ou não funções auxiliares para implementar `somaTotal`.