1ª Lista de Exercícios de Paradigmas de Linguagens Computacionais
Professores: Fernando Castor, Paulo Borba e Márcio Cornélio
Monitores:
Ciência da Computação: Alberto Rodrigues Costa Junior (arcj), Jéssica de Carvalho Barbalho (jcb), Luana Martins dos Santos (lms7)
Engenharia da Computação: Bruno Gomes Correia Rodrigues (bgcr), Felipe de Souza Araújo (fsa2), Jefferson Luis Alves de Medeiros (jlam)
CIn-UFPE-2012.2
Disponível desde: 20/12/2012
Entrega:15/01/2013

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 11** das respostas estão corretas não foi entregue. A entrega da lista com **pelo menos 21** 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. Se você ainda não fez isso, vá até a página da monitoria da disciplina (http://sites.google.com/a/cin.ufpe.br/monitoria-plc/) e coloque os nomes dos membros da sua dupla abaixo do nome do monitor que tem menos duplas até o momento.

1) Dado um inteiro *N*, retorne uma lista de inteiros com os *N* primeiros números pares da sequência de Fibonacci. Assinatura: answer::Int->[Int]

Prelude> answer 2
[2,8]

Prelude> answer 3
[2,8,34]

2) Crie uma função de ordenação que receba uma função de comparação e uma lista, e ordena a lista de acordo com a função. Assinatura: sort::(a->a->Bool)->[a]->[a]

Prelude> sort (>) [1,4,2,3,8,15,0]
[15,8,4,3,2,1,0]

3) Crie um função que recebe uma lista de inteiros e retorna a lista ordenada em função da soma de seus digitos(crescente)
Exemplo:
Prelude> ordenar [5,12,70,8,25,3,150]

[12,3,5,150,70,25,8]

4) Crie uma função retangulos :: (Int,Int) -> (Int,Int) -> (Int,Int) -> (Int,Int) -> Bool , que recebe 4 pontos(representando 2 retângulos) no formato de tupla de inteiros de maneira que o primeiro e o segundo elementos da tupla representem, respectivamente, as coordenadas x e y em um plano cartesiano. O retorno da função é um booleano que deve indicar se há interseção entre os retângulos.
1º ponto representa o vértice superior direito do 1º retângulo
2º ponto representa o vértice inferior esquerdo do 1º retângulo
3º ponto representa o vértice superior direito do 2º retângulo
4º ponto representa o vértice inferior esquerdo do 2º retângulo
Exemplo:
Prelude> retangulos (1,2) (3,4) (2,1) (4,3)
True

5) Crie uma função agrupar :: Eq a => [a] -> [[a]] , que recebe uma lista e devolve uma lista de lista dos elementos iguais
Exemplo:
Prelude> agrupar "arara"
["aaa","rr"]
Prelude> agrupar [1,2,5,1,2,4,5,6,7,5,2,4]
[[1,1],[2,2,2],[5,5,5],[4,4],[6],[7]]
Prelude> agrupar [True,False,True,False,False,True]
[[True,True,True],[False,False,False]]

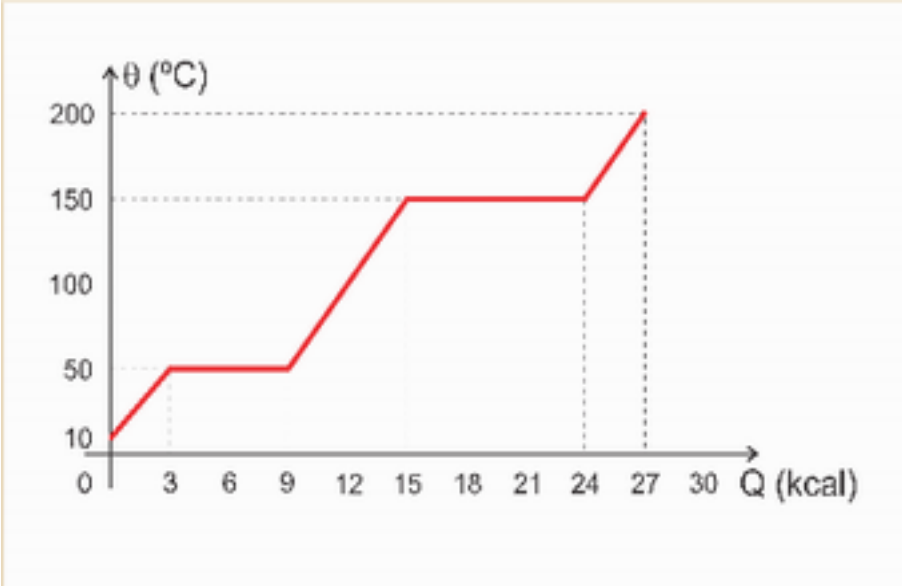
6) Crie uma função que receba como parâmetro uma função de comparação e uma lista, e retorna uma tupla onde o lado esquerdo da tupla contém os elementos da lista que não satisfazem a função e o lado direito os elementos que satisfazem.
Assinatura : divide::(a->Bool)->[a]->([a],[a])

Prelude> divide (<4) [1,2,3,4,5]
([4,5],[1,2,3])

7) Crie uma função que receba uma string como entrada e retorne uma substring que aparece mais vezes nessa string. Caso a quantidade de ocorrências da substring seja igual, o critério de desempate é a ordem lexicográfica.
Assinatura: ocorrencias::String->String
Obs.: Substring aqui são string's composta apenas de caracteres do alfabeto português.
Ex: String = “oi xau oba , , , , opa” , substrings = {“oi”,”xau”,”oba”,”opa”}

Prelude> ocorrencias "feliz natal, feliz ano novo"
"feliz"

8) Para ajudar no estudo de física, Pablo, um colega seu do CTG, pediu pra que você criasse uma programa que mapeia o comportamento de certa quantidade de uma substância (temperatura) em relação a quantidade de calor da mesma, como você é muito esperto, escolheu criar uma função em haskell (temperatura::Float->Float), devido à sua semelhança com a linguagem natural da matemática, que dado a quantidade de calor da substância desconhecida retorne sua temperatura. O gráfico da temperatura da substância em função de sua quantidade de calor é o seguinte:



Fonte: Link

Prelude> temperatura 0
10.0

Prelude> temperatura 1.5
30.0

Prelude> temperatura 12
100.0

Prelude> temperatura 18
150.0

Prelude> temperatura 6.75
50.0

As questões 9, 10 e 11 possuem relação entre si mas devem ser mandadas em arquivos separados, caso seja necessário use "sqrt", fazendo as devidas conversões de tipos é claro.

9) Crie uma função que quando chamada retorne uma lista com todos os primos em ordem crescente (primos::[Int]), perceba que a lista é infinita.

Prelude> take 100 primos *-- os 100 primeiros primos*
[2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,71,73,79,83,89,97,101,103,107,109,113,
127,131,137,139,149,151,157,163,167,173,179,181,191,193,197,199,211,
223,227,229,233,239,241,251,257,263,269,271,277,281,283,293,307,311,313,317,
331,337,347,349,353,359,367,373,379,383,389,397,401,409,419,421,431,433,439,
443,449,457,461,463,467,479,487,491,499,503,509,521,523,541]

Prelude> primos!!150
877

Prelude> sum (take 10 primos) *-- a soma dos 10 primeiros primos*
129

Prelude> primos *-- todos os primos*
[2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,71,73,79,83,89,97,101,103,
107,109,1..

10) Faça uma função, usando a função “primos” da questão anterior que, dado um numero, retorne se ele é ou não primo (isPrimos::Int->Bool).

Prelude> isPrimos 2
True
Prelude> isPrimos 104729
True
Prelude> isPrimos 104731
False
Prelude> isPrimos 6
False

11) Para saber se um número faz parte da lista dos primos (primos) basta saber se este numero é primo (isPrimo). Para saber se um número é primo (isPrimo) basta saber alguns dos números primos gerados (primos). Sabendo disso, modifique a função “primos” para que ela utilize a função “isPrimos”. Repare que agora uma função usa a outra.

12) Construa a função somaPG :: [(Int, Int, Int)] -> [Double]. Considere como entrada uma lista de tuplas. Cada elemento da lista terá este formato (*n*, *q, a1*), representando uma progressão geométrica. A saída dessa questão deverá ser uma lista com os valores da soma dos *n* primeiros termos de cada PG.

$$S_n = \frac{a_1(q^n - 1)}{q - 1}$$

OBS.: considere que *q* > 1 e *a1* > 0;

n = número de termos
q = razão
a1 = primeiro termo da PG

Prelude> somaPG [(10,2,1), (2,2,7), (5,11,3)]
[1023.0, 21.0, 48315.0]

13) Considere os seguintes tipos:
type Estilo = String
type Nome = String
type Musica = (Nome, Estilo)

Defina a função musicPlayer :: [Musica] -> [Estilo] -> [Musica] que recebe uma lista de *Musica* e uma lista de *Estilo* e retorna uma lista de *Musica* ordenada pelo *Estilo* desde este faça parte da lista de Estilos passada como parâmetro. Em caso de empate, as músicas podem aparecer em qualquer ordem.

Prelude> musicPlayer [("musica1", "Grindcore"), ("musica2", "Black Metal"), ("musica5", "Death Metal"),("musica", "Rap Metal"),("musica", "Grindcore"),("musica3", "Black Metal"), ("musica4", "Christian Metal")] ["Black Metal", "Grindcore"]

[("musica2","Black Metal"),("musica3","Black Metal"),("musica1","Grindcore"),("musica","Grindcore")]

14) Implemente a cifra de Vigenère. Para tal, defina a função cifra :: String -> String -> String que recebe uma string a ser "cifrada" e uma string representando a chave de encriptação. A fórmula para codificar é:

$$C_i \equiv P_i + K_i \pmod{26}$$

tal que *Ci* é um Char pertencente à String codificada, *Pi* é um Char da String original e *Ki* é um Char da chave utilizada na codificação.
O retorno de *cifra* é a string codificada, importe **Data.Char** se achar necessário.
Fonte: Wikipedia

Prelude> cifra "atacarbasesul" "limao"
"lbmcocjmssdcx"

15) Implemente a função tuplinhas :: String -> (Int, Int, Int) que recebe uma string como parâmetro e retorna uma tupla com os números que indicam quantos caracteres são letras minúsculas, letras maiúsculas e outros caracteres.

Prelude> tuplinhas "Agora é tarde"
(9, 1, 3)

16) Crie um tipo sinônimo **Vector**, que represente uma lista de **Double**.
Crie um tipo sinônimo **Matrix**, que represente uma lista de **Vector**.
Crie uma função **operateVectors**, que recebe como parâmetro uma "função de Double para Double para Double" e dois **Vector**'s e retorna um **Vector**. Os dois vetores passados para operateVectors devem necessariamente ter o mesmo tamanho. O vetor retornado deve representar um vetor formado pelos valores resultantes da aplicação da função passada como parâmetro a cada par de valores que aparecem na mesma posição nos dois vetores passados como parâmetro.
Ex.: **operateVectors (+) [1, 2] [3, 4]** deve resultar em **[4, 6]**.

Crie uma função **operateMatrices** que recebe como parâmetro uma "função de Double para Double para Double" e duas **Matrix**'s e retorna uma **Matrix**. As matrizes passadas para operateMatrices devem ter as mesmas dimensões. O funcionamento desta função deve ser semelhante ao da **operateVectors**.
Ex.: **operateMatrices (+) [[1, 2], [3, 4]] [[5, 6], [7, 8]]** deve resultar em **[[6, 8], [10, 12]]**.

17) Crie uma função **dotProduct** que recebe como parâmetro dois **Vector**'s e retorna um **Double**. O valor retornado deve representar o valor do produto interno entre os dois vetores passados como parâmetro.

18) Crie uma função **transpose** que recebe como parâmetro uma **Matrix** e retorna uma **Matrix**. A matriz retornada deve representar a matriz transposta da matriz passada como parâmetro.

19) Crie uma função multiplyMatrices que recebe como parâmetro duas **Matrix**'s e retorna uma **Matrix**. A matriz retornada deve representar a multiplicação entre as duas matrizes passadas como parâmetro.

20) Nem tudo são flores! Crie funções para realizar acessos às posições de uma Matrix através de índices. Você deve criar duas dessas funções, set e get, uma para "mudar" o valor em determinada posição e outro para obter o valor armazenado na posição.

21) Crie uma função inverse que recebe como parâmetro uma **Matrix** e retorna uma **Matrix**. A matriz retornada deve representar a matriz inversa da matriz passada como parâmetro.

22) Dada uma expressão aritmética, calcule o valor final da mesma sempre lendo da esquerda para a direita e realizando as operações. A expressão não deve ter parênteses e não precisa levar em conta precedência matemática (por exemplo, se uma soma aparece à esquerda de uma multiplicação, a soma é calculada primeiro). Se a expressão não puder ser calculada (0/0, ou a expressão contém caracteres inválidos) o resultado deve ser 0.

Prelude> calculaExpressao "3+4-9*3"
-6

Prelude> calculaExpressao "2/1*10-4"

16

Prelude> calculaExpressao "34)5"
0

23) Escreva uma função que, dada uma frase, duplique todas as vogais (a,e,i,o,u) da mesma, mantendo a vogal da forma que ela aparece (sendo maiúscula duplica com a mesma vogal maiúscula, o mesmo para as minúsculas).

Prelude> duplicaVogais "Haskell é uma beleza!"
"Haaskeel éé uumaa beeleezaa!"

Prelude> duplicaVogais "A entrega da lista foi adiada \o/"
"AA eentreegaa daa liistaa fooii aadiiaadaa \oo/"

24) Dada uma lista de pontos consecutivos representada por Ponto = Float, que representa a distância entre o ponto dado e o ponto anterior, calcule a distância total do percurso (a soma das distâncias entre os pontos). Lembre-se de considerar os casos de lista vazia e lista com apenas um ponto.

Prelude> calculaDistancia [0, 3.5, 6.7]
10.2

Prelude> calculaDistancia [0, 44, 9.1]
53.1

25) Uma lista de usuários é representada por um *id* e um *nome* (Usuario = (Int, String)). Escreva uma função que retorne todos os ids dos usuários que possuem o nome passado como parâmetro. Sabendo que o nome passado pode estar contido no nome do usuário.

Prelude> usuarioIds [(1, "John Green"), (2, "Doug Ross"), (3, "Rosa Lewis"), (4, “John Carter”), (5, “Kerry Weaver”)] "John"
[1, 4]

Prelude> usuariosIds [(1, "John Green"), (2, "Doug Ross"), (3, "Rosa Lewis"), (4, “John Carter”), (5, “Kerry Weaver”)] "Ros"
2, 3]