Universidade Federal do Ceará – Campus Russas
Linguagens de Programação
Prof. Ms. Rafael Ivo

Lista de Exercícios 7 – Paradigma Funcional

1) Reimplemente as funções e operadores Haskell abaixo usando operações mais simples.

Função	**Domínio e Intervalo**	**Explicação**
`++`	`[a] -> [a] -> [a]`	Concatena duas listas
`!!`	`[a] -> Int -> a`	`x !! n` retorna o n-ésimo elemento da lista x
`length`	`[a] -> Int`	Número de elementos em uma lista
`head`	`[a] -> a`	Primeiro elemento em uma lista
`tail`	`[a] -> [a]`	Todos os elementos da lista, exceto o primeiro
`last`	`[a] -> a`	Último elemento em uma lista
`init`	`[a] -> [a]`	Todos os elementos da lista, exceto o último
`take`	`Int -> [a] -> [a]`	Retorna a lista dos n primeiros elementos de uma lista
`drop`	`Int -> [a] -> [a]`	Remove os n primeiros elementos de uma lista
`reverse`	`[a] -> [a]`	Inverte a ordem dos elementos de uma lista
`elem`	`a -> [a] -> Bool`	Verifica se um elemento está na lista
`zip`	`[a] -> [b] -> [(a,b)]`	Faz uma lista de pares com elementos correspondentes das listas
`sum`	`[Int] -> Int`	Soma os elementos de uma lista
`product`	`[Int] -> Int`	Multiplica os elementos de uma lista

2) Escreva um programa em Haskell que:
a) recebe uma lista de inteiros e retorne a lista com o dobro do valor de cada elemento.
b) recebe um elemento e uma lista e retorne o número de ocorrências do elemento na lista.
c) recebe uma string e retorna uma string com somente as letras minúsculas presentes na string.

3) Na sequência de Fibonacci, cada elemento é igual a soma dos dois anteriores, exceto os dois primeiros que são dois números 1. Escreva um programa em Haskell que:
a) retorne o n-ésimo elemento da sequência de Fibonacci.
b) retorna a lista dos n primeiros elementos da sequência de Fibonacci.

4) Números primos são uma classe de números dos quais possuem apenas 2 divisores: o número 1 e o próprio número. Escreva um programa em Haskell que:
a) calcule a quantidade de números que podem dividir um determinado número n sem deixar resto
b) verifique se um determinado número n é primo
c) escreva uma lista dos números primos menores do que um determinado número n
d) escreva uma lista dos n primeiros números primos

5) Escreva funções em Haskell que dado um valor de x:
a) receba os números a e b e calcule uma equação do primeiro grau: $ax + b$
b) receba os números a, b e c e calcule uma equação do segundo grau: $ax^2 + bx + c$
c) receba os números a e b , c e d e calcule uma equação do terceiro grau: $ax^3 + bx^2 + cx + d$
d) receba uma lista $[a_0, a_1, \ldots, a_n]$ e calcule um polinômio de grau n: $a_n x^n + \ldots + a_1 x + a_0$

6) O algoritmo de ordenação quicksort é um dos algoritmos de ordenação mais eficientes. Em uma de suas implementações mais básicas, ele toma o primeiro elemento da lista como pivot, divide a lista original em duas partes: uma lista dos elementos menores do que o pivot e uma lista dos elementos maiores do que o pivot. Após chamar recursivamente o algoritmo para cada uma dessas metades, ele reagrupa a lista ao unir a lista dos menores, agora ordenada, com o pivot e, por fim, a lista dos maiores, já ordenada também. Abaixo, mostra uma possível execução de quicksort sobre uma lista de elementos:

```
QS [5,3,1,8,7,4,6,2,9]
QS [3,1,4,2] ++ [5] ++ QS [8,7,6,9]
QS [1,2] ++ [3] ++ QS [4] ++ [5] ++ QS [7,6] ++ [8] ++ QS [9]
QS [] ++ [1] ++ QS [2] ++ [3] ++ [4] ++ [5] ++ QS [6] ++ [7] ++ QS [] ++ [8] ++ [9]
[] ++ [1] ++ [2] ++ [3,4,5] ++ [6] ++ [7] ++ [] ++ [8,9]
[1,2,3,4,5,6,7,8,9]
```

Implemente o algoritmo quicksort em Haskell.

7) O cadastro de pessoas em uma academia guarda informações dos seus clientes como: número de matrícula (**`Int`**), nome (**`String`**), idade (**`Int`**), peso (**`Float`**), altura (**`Float`**) e sexo (**`Char`**). Faça o que se pede em Haskell:
a) Defina um tipo para guardar os dados de uma pessoa
b) Defina funções para retornar cada um dos atributos de uma pessoa
c) Defina uma função para retornar o índice de massa corpóreo de uma pessoa. (`IMC = peso/altura`^{2})
d) Defina uma função que recebe uma lista de pessoas e o número de matrícula e retorna o registro de uma pessoa se existir. Se não existir, deve emitir uma mensagem de erro.
e) Defina uma função que receba o número de matrícula de duas pessoas e retorne o número da que possui o IMC mais alto
f) Defina uma função que receba uma lista de pessoas e retorne o nome da pessoa com maior IMC.

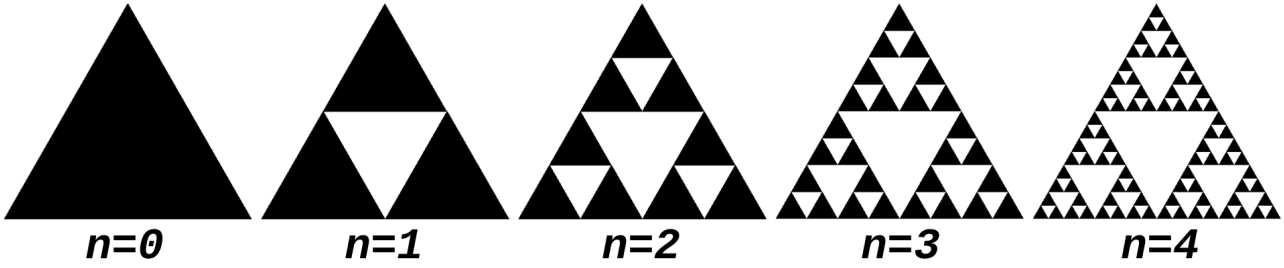
8) Algumas funções para tratamento de strings:
a) Escreva uma função que torne letras minúsculas em maiúsculas e deixe os demais caracteres como são.
b) Escreva uma função que converta todas as letras minúsculas em letras maiúsculas.
c) Escreva uma função que verifique se um caractere é uma letra.
d) Escreva uma função que remova todos os caracteres que não são letras.

9) Haskell permite o uso de funções como argumentos em outras funções de forma trivial. Implemente as funções que se pede:
a) Função **`mapear::(a->b)->[a]->[b]`**. Essa função aplica uma determinada função **`f`** em todos os elementos de uma lista retornando uma outra lista com os resultados da aplicação de **`f`**.
b) Função **`filtrar::(a->Bool)->[a]->[a]`**. Essa função recebe uma função booleana **`p`** e uma lista de elementos e remove todos os elementos que retornam falso quando aplicadas a **`p`**.
c) Função **`pegarEnquanto::(a->Bool)->[a]->[a]`**. Essa função recebe uma função booleana **`p`** e uma lista de elementos e retorna uma lista com os primeiros elementos que retornam verdadeiro até o primeiro que é falso.
d) Função **`removerEnquanto::(a->Bool)->[a]->[a]`**. Essa função recebe uma função booleana **`p`** e uma lista de elementos e remove os primeiros elementos que retornam verdadeiro até o primeiro que é falso.

10) Escreva uma função que pegue o nome completo de uma pessoa e retorne uma lista com os nomes e cada um dos sobrenomes separados.

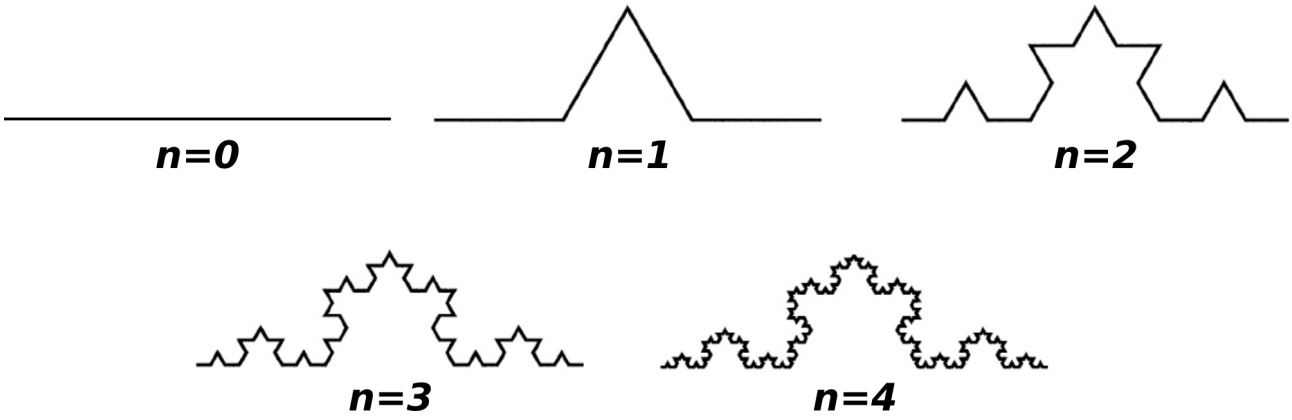
11) Escreva uma função que receba duas strings e verifique se a primeira está contida na segunda

12) O Triângulo de Sierpinski é uma das formas elementares da geometria fractal. Trata-se de um conjunto autossimilar de um triângulo. Ao unir os pontos médios dos lados de um triângulo, dividimos o triângulo original em 4 triângulos semelhantes, com metade da base e metade da altura do triângulo original. O triângulo do meio não pertence ao fractal. Os demais são outros triângulos de Sierpinski por recursão. Ex: abaixo estão os triângulos de Sierpinski com zero a quatro níveis de recursão.



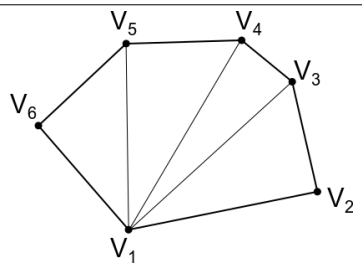
A área de um triângulo pode ser calculada por **(base × altura)/2** . Escreva um programa em Haskell para calcular a área de um triângulo de Sierpinski com base **b**, altura **a** e **n** níveis de recursão.

13) Um fractal é um objeto geométrico que pode ser dividido em partes, cada uma das quais semelhante ao objeto original. A curva de Koch é uma curva geométrica e um dos primeiros fractais a serem descritos. Podemos imaginar a sua construção a partir de um segmento de reta que será submetido a alterações recorrentes (iterações), como a seguir se descreve:
1. Divide-se o segmento de recta em três segmentos de igual comprimento.
2. Desenha-se um triângulo equilátero em que o segmento central, referido no primeiro passo, servirá de base.
3. Apaga-se o segmento que serviu de base ao triângulo do segundo passo.



Crie uma função recursiva em Haskell (**snowKoch n l**) que calcule o comprimento da curva no **n**-ésimo nível recursivo e possuindo largura igual a **l**.

14) A área de um polígono convexo pode ser calculada ao particioná-lo em diversos triângulos, com diagonais saindo de um dos vértices do polígono. Suponha que um determinado código em Haskell já possua uma função de calcular a área do triângulo (**areaT**) dados os três vértices. Crie uma função para calcular a área de um polígono convexo (**areaP**).

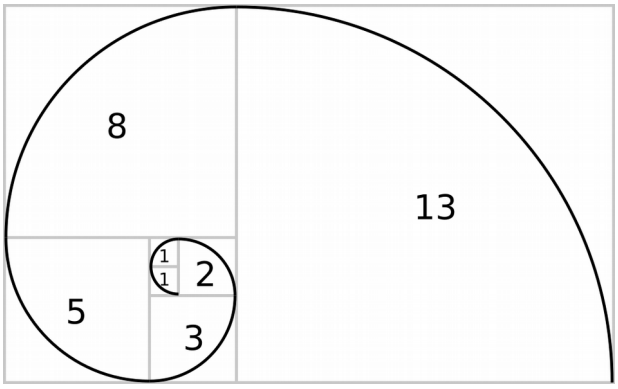



```haskell
type Ponto = (Float,Float)

--Calcula a Área do Triâgulo
areaT::Ponto->Ponto->Ponto->Float
areaT a b c = ...

--Calcula a Área do Polígono Convexo
areaP::[Ponto]->Float
...
```

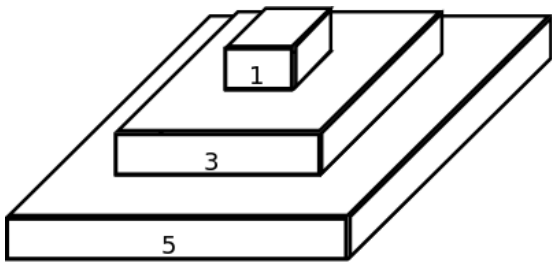
15) Cilindros podem ser empilhados na forma de uma “pirâmide” como mostrado ao lado. Escreva uma função **recursiva** em Haskell que receba o número de cilindros da base e retorne a quantidade total de cilindros da pilha.

16) A sequência de Fibonacci está intrinsecamente ligada à natureza. Ela pode ser percebida na espiral do nautilus, um cefalópode marinho. Quadrados com lados iguais às medidas da sequêcia de Fibonacci (**1,1,2,3,5,8,13,...**) são posicionados como mostrados na figura abaixo. Dentro de cada quadrado existe ¼ de circunferência de raio igual ao lado do quadrado onde ela se encontra (A função **arco** dada abaixo calcula esse comprimento dado o raio **r**). Crie um código em Haskell que calcule o comprimento da espiral até o n-ésimo número de Fibonacci.

```
arco r = (pi * r)/2 where pi = 3.141592
```



17) As pirâmides do Egito são pirâmides de base quadrada e foram construídas usando blocos de pedra praticamente de mesmo tamanho que eram postas exatamente uma sobre a outra. Após construir uma camada de blocos, a camada superior não pode ter blocos na borda da camada inferior. O exemplo abaixo mostra uma pirâmide de 3 camadas, onde os números indicam a quantidade de blocos vistos da lateral. Crie uma função recursiva em Haskell **qtdBlocos n** que retorne a quantidade de blocos de uma pirâmide com **n** camadas.



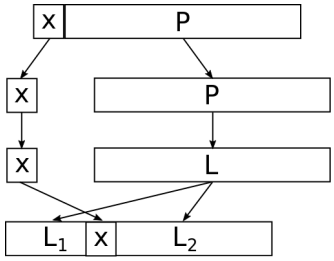
18) O máximo divisor comum (**MDC**) de dois números inteiros positivos **x** e **y** possui as seguintes propriedades:

- O **MDC** de **x** e **y** é igual ao **MDC** de **y** e **x-y**, se **x > y**.
- O **MDC** de **x** e **y** é igual ao **MDC** de **y** e **x**.
- O **MDC** de dois números iguais é o próprio número.

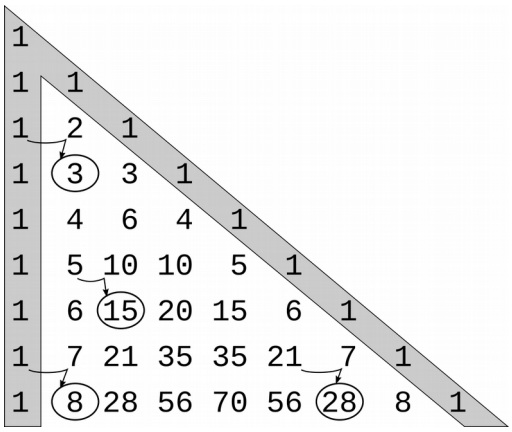
Ex: **MDC(10,6) = MDC(4,6) = MDC(4,2) = MDC(2,2) = 2**.
Escreva uma função em Haskell para calcular o **MDC** de dois números inteiros positivos.

19) Uma progressão aritmética (P.A.) é uma sequência numérica em que cada termo, a partir do segundo, é igual a soma do termo anterior com uma constante **`r`**, denominada razão da P.A. Por exemplo, a sequência **`1,4,7,10,13`** é uma P.A. de razão **`3`**. Escreva uma função em Haskell **`paList`** que retorne uma lista representando uma P.A. dado o elemento inicial **`a`**, a razão **`r`** e a quantidade de elementos **`n`**. Por exemplo: **`paList 2 6 5 = [2,8,14,20,26]`**.

20) InsertionSort é um algoritmo de ordenação que, dada uma lista com n números, ele vai inserindo os elementos um a um em uma lista, inicialmente vazia, colocando os elementos na ordem correta. A figura ao lado mostra um esboço da ideia: o elemento x é reservado, o restante da lista é ordenado recursivamente, e x é inserido na posição apropriada nesta nova lista ordenada. Implemente esse algoritmo em Haskell.



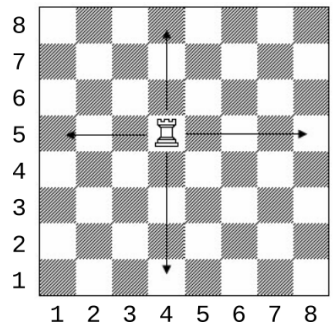
21) O triângulo de Pascal é um triângulo numérico infinito formado por números binomiais que recebeu este nome devido aos estudos que o filósofo e matemático Blaise Pascal. O triângulo é infinito e simétrico, e seus lados esquerdo e direito sempre devem possuir o número 1. Uma propriedade interessante do triângulo de Pascal é que a soma de dois números vizinhos de uma mesma linha do triângulo é igual ao número que está logo abaixo do segundo número como mostrado na figura ao lado. Escreva uma função em Haskell **`pascalElem n m`** que retorne o **`m`**-ésimo elemento da **`n`**-ésima linha do triângulo de Pascal. Obs: As linhas iniciam-se de zero e a posição do elemento na linha também inicia-se de zero. Por exemplo, no caso dos números circulados:



`pascalElem 3 1 = 3`
`pascalElem 6 2 = 15`
`pascalElem 8 1 = 8`
`pascalElem 8 6 = 28`

22) A função **`nDig::Int->Int`** retorna a quantidade de dígitos que um determinado número possui. Por exemplo: **`nDig 452 = 3`**. Implemente em Haskell essa função de forma recursiva.

23) No xadrez, a torre se move em linha reta horizontalmente e verticalmente pelo número de casas não ocupadas, até atingir o final do tabuleiro ou ser bloqueado por outra peça, como mostrado na figura ao lado. Seja uma função **`torre::(Int,Int)->[(Int,Int)]`** que recebe as coordenadas de uma torre em um ***tabuleiro vazio*** e retorne a lista das posições onde a torre pode se mover. Por exemplo, no caso da figura:



`torre (4,5) = [(4,1),(4,2),(4,3),(4,4),(4,6),(4,7),(4,8),(1,5),(2,5),(3,5),(5,5),(6,5),(7,5),(8,5)]`