**UNIVERSIDADE FEDERAL FLUMINENSE**
**INSTITUTO DE CIÊNCIA E TECNOLOGIA**
**CURSO DE CIÊNCIA DA COMPUTAÇÃO**

**Lista de Exercício de Linguagens de Programação – Prog. Funcional**

1) Construa uma função del_posicao_n :: [Int] → Int → [Int] em que, dada uma lista de inteiros e a posição de um elemento qualquer, retorne uma nova lista sem aquele elemento na n-ésima posição. Exemplo de uso:

> del_posicao_n [1,3,4,1,3,2] 4
[1,3,4,3,2]
> del_posicao_n [1,3,4,1,3,2] 1
[3,4,1,3,2]

2) Implemente uma função que receba uma lista de inteiros (não necessariamente ordenada) e retorne uma lista ordenada (de forma crescente), formada somente pelos números ímpares da lista recebida.

> impares [3,6,4,8,1,9,7]
[1,3,7,9]

3) Construa uma função que retorne os n primeiros elementos da sequência de Fibonacci.

> fibonacci 10
[0,1,1,2,3,5,8,13,21,34]

4) (*) Crie um programa que faça uma codificação sobre uma sequência de caracteres iguais, substitua a sequência por !na, onde **n** é o número de vezes que o caracter **a** é repetido. Note que só é interessante comprimir sequências maiores que 3. Lembre que uma string (sequência de caracteres) é equivalente a uma lista de caracteres. Exemplo:

> comprime “asdffffghjjkllllpoooi”
“asd!4fghjjk!4lpoooi”

5) (*) Implemente uma função que realize a descompressão da lista acima.

6) Implemente uma função que calcula a interseção entre 2 listas.

> intersecao [3,6,5,7] [9,7,5,1,3]
[3,5,7]

7) Implemente uma função que faça uma busca por substrings de início. Considere sempre nos n-primeiros caracteres que o usuário passar na linha de comando. Exemplo:

> busca_sub :: String → [String] → [String]
> busca_sub “an” [“freddy mercury”, “antonio banderas”, “zorro”, “zebra” ]
[“antonio banderas”]

> busca_sub “z” [“freddy mercury”, “antonio banderas”, “zorro”, “zebra” ]
[“zorro”, “zebra”]

8) Defina uma função que repita as ocorrências até um determinado valor, no formato de uma lista, tal que:

> repete :: Int → [Int]
> repete 4
[4,4,4,4,3,3,3,2,2,1]

9) Determinar se o conteúdo de uma lista é um palíndromo. O retorno desta função deve ser True ou False. Exemplo:

> palindromo [1,2,3,4,5]
False
> palindromo [1,2,3,2,1]
True
> palindromo [1,2,2,1]
True

10) Implemente uma função que receba um número natural, uma lista e retorne uma nova lista, na qual a posição dos elementos mudou, como se eles tivessem sido “rodados”. Exemplo:

> rodar-esquerda 1 “asdfg”
“sdfga”
> rodar-esquerda 3 “asdfg”
“fgasd”

11) Faça uma função que inverta uma lista. Exemplo:

> inverte [1, 2, 3, 4, 5]
[5, 4, 3, 2, 1]

12) Dada uma lista de strings (linhas, por exemplo) e uma palavra, implemente uma função que conte quantas vezes a palavra ocorre na lista.

> ocorrencias [“abcdef”, “aaabbbccc abc abc”, “aaaa”, “bbb”] “abc”
3

13) Complete a função dada e exemplificada abaixo para que, dada uma lista de números inteiros positivos, retorne uma tupla com o maior e o menor elemento desta lista. O maior número inteiro em Haskell pode ser obtido pela expressão: **maxBound :: Int**

```
limites l = limites_aux l (maxBound :: Int) 0
limites_aux [] min max = (min, max)
...
```

> limites [4, 6, 3, 8, 9, 7, 2, 4, 5]

(2, 9)

14) Crie uma função que remova duplicatas numa lista. Exemplo:

> remDuplicatas [1, 2, 4, 2, 6, 5, 3, 4]
[1, 2, 4, 6, 5, 3]

15) Crie uma função para retornar a soma dos quadrados dos números de 1 a n. Exemplos:

> somaQuadrados 2
5 // $1^2 + 2^2$
> somaQuadrados 4
30 // $1^2 + 2^2 + 3^2 + 4^2$

16) Simule uma lista em Haskell como uma fila, onde a cabeça é o início da lista e a cauda são os próximos elementos da fila. Crie funções para adicionarmos elementos na fila, retirarmos e calcularmos seu tamanho.

17) Refaça o exercício 16) para pilhas, supondo a cabeça como o topo da pilha.

18) (*) Suponha que queiramos usar listas para representar números naturais. O valor 0 (zero) é representado pela lista vazia (**[]**), o valor 1 é a lista com 1 **[1]**, o valor 2 é a lista com 2 1's **[1, 1]**, e assim sucessivamente. Crie funções para: a) incrementar valores (+ 1); b) decrementar valores (- 1); c) comparar valores nessa representação (retornar -1, 0 ou 1 – menor, igual ou maior); d) soma de 2 valores; e) multiplicação; f) potência entre 2 valores.

19) Suponha que o polinômio com uma variável $3x^4 + 2x^2 + 3x + 7$ seja representado em Haskell como [(3, 4), (2, 2), (3, 1), (7, 0)]. Crie funções para: a) somar polinômios (soma de radicais de fatores com a mesma potência); b) calcular o valor da função polinomial quando x vale um número dado.

20) Considerando a representação de matrizes que vimos em sala, implemente uma função que receba 2 matrizes e retorne se estas são iguais.

21) Implemente a função somatório, ou seja, dados limites inferior (li), superior (ls) e uma função (f), esta função retorna o somatório das chamadas de f para os valores entre li e ls.

22) Dadas 2 listas ordenadas como entrada, faça uma função merge.
Exemplo: merge [1, 2, 3, 4, 5] [0, 2, 6, 7]
Saída: [0, 1, 2, 2, 3, 4, 5, 6, 7]

23) Defina uma função que converta uma lista de números (unitários, 0 a 9) em uma outra lista, que é a sua tradução em string. Ou seja, a função é um tradutor simplificado. Considere um dicionário do tipo: dic_10 = [(0, “zero”), (1, “um”), (2, “dois”), …, “(9, “nove”)]
Entrada: conv_int_str [2, 5, 0]
Saída: [“dois”, “cinco”, “zero”]

24) (*) Implemente as 4 operações básicas (adição, subtração, multiplicação e divisão) com uma restrição: tais funções devem ser implementadas utilizando somente as seguintes funções primitivas:

```
constanteZero = 0
sucessor x = x + 1
oposto x = -x
```

Ou seja, não é permitido o uso dos operadores +, -, * e /. Funções definidas com estas básicas podem ser utilizadas para criar outras funções. Exemplos de uso das novas funções:

```
> multiplicacao (-7) 3
-21
> divisao 7 3
2
> soma 7 3
10
> subtracao 7 3
4
```

25) Implemente uma função unpack, a qual deve separar uma string em uma lista de strings com caracteres iguais em sequência. Observe os exemplos abaixo:

```
> unpack "aabbccccdeedff"
["aa","bb","cccc","d","ee","d","ff"]
> unpack "abcabcaabbcc"
["a","b","c","a","b","c","aa","bb","cc"]
```

26) Implemente a função applylist, a qual recebe uma lista de chamadas parciais de funções e um valor e aplica cada chamada ao valor passado. Observe o exemplo abaixo:

```
> applylist [(+ 10),(* 3)] 2
[12,6]
```

27) Defina a função mydropwhile, a qual recebe um predicado e uma lista e retorna a lista sem os elementos que não satisfazem o predicado:

```
> mydropwhile (~=0) [1,2,1,1,2,1,0,1,2,1,1]
[0,1,2,1,1]
```

28) Considere a assinatura da função pré-definida zip em Haskell: zip :: [a] -> [b] -> [(a,b)]. Sem ler na documentação o que ela faz, apresente uma implementação que respeite esta assinatura.

29) Considere as funções pré-definidas *sum :: Num a => [a] -> a* e *map :: (a -> b) -> [a] -> [b]*. Defina a função length em função de sum e map.