CEFET-MG
CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS

Alunos e Matrículas:

Alex Meireles Santos Almeida 20193020407
Vitor Theodoro Rocha Domingues 20193017359

Trabalho Prático III

1) Defina uma função recursiva para o cálculo de potência de dois números inteiros, onde o primeiro número é elevado ao segundo. Não se pode usar o operador de potência (^).

Ex.: **> potencia 2 3** -- 8=2*2*2
8

```haskell
potencia:: Integer -> Integer -> Integer
potencia a b | a == 1 = 1 | b == 0 = 1 | b == 1 = a
potencia a b =  a * potencia a (b-1)

main = do
print( potencia 2 3)
```

```
> ghc -o main main.hs
[1 of 1] Compiling Main
Linking main ...
> ./main
8
>
```

2) Calcular o somatório dos elementos ímpares de uma lista de inteiros.

Ex.: **> somaImpares [1,3,2,7,4,6,5]** -- 16=1+3+7+5
16

```haskell
somaImpares :: [Integer] -> Integer
somaImpares[] = 0
somaImpares (n:x)
    | mod n 2 /= 0 = n + somaImpares x
    | mod n 2 == 0 = somaImpares x

main = do

print( somaImpares [1,2,3,5] )
```

```
> ghc -o main main.hs
[1 of 1] Compiling Main
Linking main ...
> ./main
9
>
```

3) Substituir todos elementos de um determinado valor de uma lista de inteiros por um outro valor.

Ex.: **> substituir 1 0 [1,2,1,3,1]**
[0,2,0,3,0]

```haskell
substituir ::  Int -> Int -> [Int] -> [Int]
substituir a b [] = []
substituir a b (x:xs) | (a == x) = b:(substituir a b xs)|otherwise = x:substituir a b xs


main = do
print( substituir 1 0 [1,2,1,3,1])
```

```
$ghc -O2 --make *.hs -o main -threaded -rtsopts
[1 of 1] Compiling Main             ( main.hs, main.o )
Linking main ...
$main
[0,2,0,3,0]
```

4) Verificar se um número é primo.

Ex.: > primo 17
True
> primo 0
False

```haskell
comparar :: [Integer] -> Integer-> Bool
comparar(a:x) n
    | a == n, a/=1 = True
    | mod n a == 0, a/= n = False
    | otherwise = comparar x n

primo :: Integer -> Bool
primo 0 = False
primo n = comparar[2..n] n

main = do
print( primo 17 )
```

```
> ghc -o main main.hs
[1 of 1] Compiling Main
Linking main ...
> ./main
True
>
```

5) Verifique se um número é perfeito, isto é, é igual a soma de seus divisores (exceto o próprio número).

Ex.: **> perfeito 28** -- 28=1+2+4+7+14
True

```haskell
divisores:: Int -> Int -> Int
divisores a b | b == 1 = 1
divisores a b | a == b = divisores a (b-1)
divisores a b | ((a `mod` b) == 0) = b + divisores a (b-1) | otherwise = divisores a (b-1)

perfeito ::  Int -> Bool
perfeito n | ((divisores n n) == n) = True | otherwise = False


main = do
print( perfeito 28)
print (divisores 28 28)

```

```
$ghc -O2 --make *.hs -o main -threaded -rtsopts
[1 of 1] Compiling Main             ( main.hs, main.o )
Linking main ...
$main
True
28
```

6) Função que retorna uma lista com a representação em binário de um número inteiro.

Ex.: **> binario 20**
[1,0,1,0,0]

```haskell
binario :: Integer -> [Integer]
binario 0 = []
binario n
    | mod n 2 == 1 = (binario ( div n 2)) ++ [1]
    | otherwise = (binario ( div n 2)) ++ [0]

main = do

print (binario 13)
```

```
> ghc -o main main.hs
[1 of 1] Compiling Main
Linking main ...
> ./main
[1,1,0,1]
```

7) Verificar se todos os elementos de uma lista são distintos.

Ex.: **> distintos [1,2,4,2,5]**
False
> distintos [3,2,1]
True

```haskell
igual:: Int -> [Int] -> Int
igual a [] = 1
igual a (x:xs) | a == x = 0 | otherwise = igual a xs

distintos:: [Int] -> Bool
distintos [] = True
distintos (x:xs) | ((igual x xs) == 1) = distintos xs | otherwise = False


main = do
print(distintos [1,2,4,6,5])
```

```
$ghc -O2 --make *.hs -o main -threaded -rtsopts
[1 of 1] Compiling Main             ( main.hs, main.o )
Linking main ...
$main
True
```

8) Verificar se duas listas são disjuntas.

Ex.: **> disjuntas [1,2,3] [5,4,6,0]**
True

```haskell
compararListas :: Integer -> [Integer] -> Bool
compararListas _ [] = False
compararListas d (e:x)
   | d == e = True
   | otherwise = compararListas d x

disjuntas :: [Integer] -> [Integer] -> Bool
disjuntas [] [] = False
disjuntas a [] = True
disjuntas [] b = True
disjuntas (c:b) a
    | compararListas c a == True = False
    | otherwise = disjuntas a b

main = do

print (disjuntas [1,2,3] [4,5,6,0])
```

```
> ghc -o main mai
[1 of 1] Compilin
Linking main ...
> ./main
True
>
```

9) Verificar se uma lista de inteiros é palíndromo.

Ex.: **> palindromo [1,2,3,4,3,2,1]**
True

```haskell
palindromo:: [Int] -> Bool
palindromo [] = True
palindromo x | x == inverso x = True | otherwise = False

inverso:: [Int] -> [Int]
inverso [] = []
inverso (x:xs) = inverso xs ++ [x]


main = do
  print(palindromo[1,2,3,4,3,2,1])
```

```
> stack -
True
>
```

10) Calcular todas as somas parciais de uma lista de inteiros.

Ex.: **> somaParciais [1,2,3,4]** -- [1,1+2,1+2+3,1+2+3+4]
[1, 3, 6, 10]

```haskell
retira:: [Integer] -> [Integer]
retira [a] = []
retira (a:x) = [a] ++ retira x

soma:: [Integer] -> Integer
soma [] = 0
soma (a:x) = a + soma x

somaParciais:: [Integer] -> [Integer]
somaParciais [] = []
somaParciais x = (somaParciais (retira x)) ++ [(soma x)]

main = do
 print (somaParciais [1,2,3,4])
```

```
> stack --verbosity warn run
[1,3,6,10]
>
```

11) Linearizar uma lista de listas de inteiros.

Ex.: **> linearizar [[1,2], [5], [0,4,2]]**
[1,2,5,0,4,2]

```haskell
linearizar:: [[Int]] -> [Int]
linearizar [] = []
linearizar (x:xs) = x ++ linearizar xs

main = do

 print(linearizar [ [1,2], [5], [0,4,2] ])
```

```
> stack --verbos
[1,2,5,0,4,2]
>
```

12) Deslocar todos elementos de uma lista de inteiros k posições para a esquerda.

Ex.: **> shift 3 [1,5,6,7,3,4,1]** -- k=3
[7,3,4,1,1,5,6]

```haskell
trocar :: [Integer] -> [Integer]
trocar [] = []
trocar(c:x) = x ++ [c]
shift:: Integer -> [Integer] -> [Integer]
shift _ [] = []
shift 0 (b:x) = [b] ++ shift 0 x
shift a b = (shift(a-1)(trocar b))
main = do
 print(shift 3 [1,5,6,7,3,4,1])
```

```
bal-simple_mPHDZzAJ
[7,3,4,1,1,5,6]
>
```

13) Remover os n últimos elementos de uma lista de inteiros.

Ex.: **> removerFim 2 [1,2,3,4,5,6]** -- n=2
[1,2,3,4]

```haskell
retirar:: [Int] -> [Int]
retirar [x] = []
retirar (x:xs) = [x] ++ retirar xs

removerFim:: Int -> [Int] -> [Int]
removerFim x xs  | x /= 0 = (removerFim (x-1) (retirar xs)) | otherwise = xs

main = do
  print( removerFim 2 [1,2,3,4,5,6] )
```

```
> stack --verbosity w
[1,2,3,4]
>
```

14) Dadas duas listas ordenadas de forma crescente, obter a lista ordenada resultante da intercalação delas.

Ex.: > **intercalar [1,5,10] [2,7,9,20,25]**
[1,2,5,7,9,10,20,25]

```
intercalar :: [Integer] -> [Integer] -> [Integer]
intercalar [] [] = []
intercalar (c:a) [] = [a] ++ intercalar a []
intercalar (d:c) [] = [b] ++ intercalar [] b
intercalar (c:a) (d:b)
  | c <= b = [c] ++ [d] ++ intercalar a b
  | otherwise = [d] ++ [c] ++ intercalar a b

main = do

print(intercalar[1,5,10][2,7,9,20,25])
```

```
> stack --verbosity warn run
[1,2,5,7,9,10,20,25]
>
```

15) Desenvolver uma solução para um quiosque de saque eletrônico que, para um determinado valor, deve entregar o menor número de cédulas de R$1, R$5, R$10, R$50 e R$100, da menor para a maior.

Ex.: > **trocar 162**
[1, 1, 10, 50, 100]

```
trocar:: Int -> [Int]
trocar x | x >= 100 = trocar (x-100) ++ [100]
  |x >= 50 = trocar (x-50) ++ [50]
  |x >= 10 = trocar (x-10) ++ [10]
  |x >= 1 = trocar (x-1) ++ [1]
  |otherwise = []
main = do

 print(trocar 162)
```

```
> stack --verbosity
[1,1,10,50,100]
>
```