Lista 06

João Vitor Espig

25 de junho de 2024

Usando LaTeX

Exercício 1

```haskell
import Text.Printf

areaCircunsferencia r = pi * r^2

main :: IO()
main = do
    let result = (let r = 10 :: Double in areaCircunsferencia r)
    printf "Area da circunferencia: %.2f\n" result
```

Exercício 2

```haskell
tipoTriangulo a b c
    | a <= 0 || b <= 0 || c <= 0 = "NAOTRI"
    | a == b && b == c = "Equilatero"
    | a + b <= c || a + c <= b || b + c <= a = "NAOTRI"
    | a == b || a == c || b == c = "Isosceles"
    | otherwise = "Escaleno"
```

Exercício 3

```haskell
multiplica x y
    | x == 0 || y == 0 = 0
```

```
    | y > 0 = x + multiplica x (y - 1)
    | y < 0 = -multiplica x (-y)
```

Exercício 4

```
multiplica x y
    | x == 0 || y == 0 = 0
    | y > 0 = x + multiplica x (y - 1)
    | y < 0 = -multiplica x (-y)

multiplicaNaturais x y
    | x < 0 || y < 0 = error "multiplicaNaturais: argumentos
       negativos"
    | otherwise = multiplica x y
```

Exercício 5

```
maxAux x y
    | x > y = x
    | otherwise = y

myMax x y z
    | x >= maxAux y z = x
    | y >= maxAux x z = y
    | z >= maxAux x y = z

myMin x y z = negate (myMax (-x) (-y) (-z))
```

Exercício 6

```
xor x y = x /= y
```

Exercício 7

```
clonaNumeros [] = []
clonaNumeros (x:xs) = x:x:clonaNumeros (xs)
```

Exercício 8

```
somaDoisPrimeiros (x:y:_) = x + y
somaDoisPrimeiros _ = error "somaDoisPrimeiros: lista com menos de
    dois elementos"
```

Exercício 9

```
listaAte x = [0..(abs x)]
```

Exercício 10

```
parOuImpar arr = map even arr
```

Exercício 11

```
soPar arr = filter even arr
```

Exercício 12

```
import Data.Char

soMinuscula arr = filter isLower arr
```

Exercício 13

```
import Data.Char

isVowel c = c `elem` "aeiouAEIOU"

substituiVogais :: [Char] -> [Char]
substituiVogais arr = map (\c -> if isVowel c then toUpper c else
    c) arr
```

Exercício 14

```
friboi arr = map (++ " Friboi") arr
```

Exercício 15

```
pertence x arr = x `elem` arr
```

Exercício 16

```
filtraAux [] newArr = newArr
filtraAux (x:xs) newArr = if x `elem` newArr
                          then filtraAux xs newArr
                          else filtraAux xs (x:newArr)

filtraLista arr = reverse (filtraAux arr [])
```

Exercício 17

```
nPrimeiros arr n = take n arr
```

Exercício 18

```
let arr = [x * 3 | x <- [0..100]]
```