DERIVAR MAP

map -- função -> lista -> lista (a -> b) -> [a] -> [b] Aplica função na lista

. -- (b -> c) -> (a -> b) -> a -> c

map . (.) -- função -> lista -> lista (b -> c) -> [a -> b] -> [a -> c]

SUBLISTAS

sublistas :: [a] -> [[a]]

sublistas [] = [[]]

sublistas (x:xs) = [x:ys | ys <- sublistas xs] ++ sublistas xs

POLINOMIAL

3)a) poli :: Int -> Int -> Int -> Int -> Int

poli a b c = (\x -> a \* x \* x + b \* x + c) o

3)b) listaPoli :: [(Int, Int, Int)] -> [Int -> Int]

listaPoli l = [ poli a b c | (a,b,c) <- l ]

3)c) appListaPoli :: [Int -> Int] -> [Int] -> [Int]

appListaPoli a b = [f x | f <- a, x <- b]

ALTMAP

altMap :: (a->b) -> (a->b) -> [a] -> [b]

altMap f g l = aplica f g l 1 (length l)

aplica f g [] n tam = []

aplica f g (x:xs) n tam

| (mod n 2 == 0 ) && ( n <= tam ) = g x : aplica f g xs (n+1) tam

| (mod n 2 /= 0 ) && ( n <= tam ) = f x : aplica f g xs (n+1) tam

MOBILE

data Mobile = Pendente Int | Barra Mobile Mobile

peso :: Mobile −> Int

peso ( Pendente n ) = n

peso ( Barra m1 m2 ) = peso m1 + peso m2

balanceado :: Mobile −> Bool

balanceado ( Pendente \_ ) = True

balanceado ( Barra m1 m2) = peso m1 == peso m2 && balanceado m1 && balanceado m2

SUCESSÃO

compreensão :: [Int] -> [Int]

compreensão (x:xs) = [ x | (x,y) <- zip (x:xs) xs, x == y ]

recursão :: [Int] -> [Int]

recursão [x] = []

recursão (x:y:zs)

| x==y = x : recursão (y:zs)

| otherwise = recursão (y:zs)

MAP,FILTER E FOLDR

ehPar :: [Int] -> Bool

ehPar p = foldr (&&) True (map (\y -> mod y 2 == 0) (filter (\x -> 0 <= x && x <= 100) p))