Trabalho 4 - PLC

1º) A abordagem de usar polimorfismo de sobrecarga em Haskell, é para que tenha varias definições da mesma função.

Ex: A função “==”, igualdade, tem diversas definições, sendo uma para comparar Inteiros, outra Strings, Char etc..

Diferente da abordagem dinamica utilizada por java, o polimorfismo de sobreposição, que redefine a função herdada (herança é um polimorfismo).

Ex: Classe Formas que implementa a função Area, é rescrita para cada classe que herda de Formas, como a classe Círculo que implementa a função Area diferente da classe Quadrado, e ambas as classes herdam de Formas.

2)

```
cut:: Char -> String->String
cut c [] = []
cut c (x:xs)     | c==x = (cut c xs)
                          | otherwise = (x:xs)

count:: Char-> String-> Int->Int
count c [] i = i
count c (x:xs) i          | c==x = (count c xs  (i+1))
                                  | otherwise = i

next:: String->String
next [] = []
next (x:xs) = (show (count x xs 1))++[x]++(next (cut x xs))


lookAndSayString::String ->Int->String
lookAndSayString [] n = []
lookAndSayString (x:xs) n    |        n==0 = (x:xs)
                                  |         otherwise = (lookAndSayString (next (x:xs)) (n-1))



lookAndSay:: Int->String
lookAndSay n          | n==0 = []
                          | otherwise =lookAndSayString (show 1) (n-1)
```

3)

```
remove:: (Eq t)=>t->[(t,[t])]->[(t,[t])]
remove r [] = []
remove r (x:xs)         | (fst x)==r = xs
                                | otherwise = (x: (remove r xs))

lookEnd::(Eq t)=>[t]->t->Bool
lookEnd [] e = False
lookEnd (x:xs) e        | x==e  = True
                                        | otherwise = (lookEnd xs e)

findNode::Eq t=>[(t,[t])] ->t->(t,[t])
findNode [] n = (n,[])
findNode (x:xs) id      | (fst x) == id = x
                                        | otherwise = (findNode xs id)


searchFor:: (Eq t)=>[(t,[t])]->[t]->t->[t]
searchFor g [] e = []
searchFor g [x] e = search g x e
searchFor g (x:y:xs) e          |(lookEnd lx e) && (lookEnd ly e) && llx < lly = x:lx
                                                |(lookEnd lx e) && (lookEnd ly e)  = y:ly
                                                |(lookEnd lx e) = x:lx
                                                |(lookEnd ly e) = y:ly
                                                |otherwise = []
                                                where lx = (search g x e); ly = search g y e;  llx =
length lx; lly = length ly

search::(Eq t) =>[(t,[t])]->t->t->[t]
search [] s e = []
search g s e    |(length (snd ns)) == 0 = []
                                |(lookEnd (snd ns) e) = [e]
                                | otherwise = s:(searchFor (remove s g) (snd ns) e)
                                        where ns = (findNode g s)


grafoo::[(Int,[Int])]
grafoo = [(1,[2,3]),(2,[3]),(3,[4]),(4,[1])]
```

4)

```
------------------------
--- QUCK SORT AGAIN?

divide:: [Int]->Int->([Int],[Int])->([Int],[Int])
divide [] n p= p
divide (x:xs) n (me, ma) |  x>n = (divide xs n (me,(x:ma)))
                                        | otherwise = divide xs n ((x:me),ma)

qs :: [Int]->[Int]
qs [] = []
qs p = (qs me )++ (head p):(qs ma)
                where (me,ma) = (divide (tail p) (head p) ([],[]))

------FILTRO MEDIANA

getLinesUp::[[Int]]->Int->Int->Int->Int->[Int]
getLinesUp [] i j n v = []
getLinesUp img i j n v  | v == 0 = []
                                                | (i-v)<0 =(getLinesUp img i j n (v-1))
                                                | j == 0 = (take (n-pl) (drop (j-pl) (img !! (max (i-v)
0) )))++(getLinesUp img i j n (v-1))
                                                | otherwise = (take n (drop (j-pl) (img !! (max (i-v)
0) )))++(getLinesUp img i j n (v-1))
                                                where pl = (div (n-1) 2)

getLinesDown::[[Int]]->Int->Int->Int->Int->[Int]
getLinesDown [] i j n v = []
getLinesDown img i j n v       | v < 0  = []
                                                        | i+v >= length img  = (getLinesDown img i j
n (v - 1))
                                                        | j== 0 = (take (n-pl) (drop (j-pl) (img !! (min
(i+v) (length img)) )))++(getLinesDown img i j n (v - 1))
                                                        | otherwise = (take n (drop (j-pl) (img !! (min
(i+v) (length img)) )))++(getLinesDown img i j n (v - 1))
                                                        where pl = (div (n-1) 2)


getSquare::[[Int]]->Int->Int->Int->[Int]
getSquare [] i j n = []
getSquare img i j n  =(getLinesUp img i j n (div (n-1) 2))++(getLinesDown img i j n (div (n-1)
2))
```

```
mediana:: [Int]->Int
mediana [] = -1
mediana  b     | (mod (length b) 2) ==0 = (div ((sb!!(div (length sb) 2)) +(sb!!((div (length sb)
2)-1))) 2)
                              | otherwise= (sb!!((div (length sb) 2)))
                                      where sb = (qs b)



itOvery:: [[Int]]->Int->Int->[[Int]]
itOvery [] i n = []
itOvery img i n | i< sz2 = (itOverx img  i 0 n):(itOvery img (i+1) n)
                                      | otherwise = []
                                              where sz2 = length img


itOverx:: [[Int]]->Int->Int->Int->[Int]
itOverx [] i j n= []
itOverx img i j n         | j< sz2 = (mediana (getSquare img i (j) n)):(itOverx img i (j+1) n )
                                              | otherwise = []
                                              where sz2 = length (img!!0)

filtroMediana:: [[Int]]->Int-> [[Int]]
filtroMediana [] n =[]
filtroMediana img n | (mod n 2)== 0 = filtroMediana img (n+1)
                                              | otherwise = (itOvery img 0 n)




imagem::[[Int]]
imagem =
[[1,2,3,4,5,6],[7,8,9,10,11,12],[13,14,15,16,17,18],[19,20,21,22,23,24],[25,26,27,28,29,30]]
```