Faculdade de Computação - Programação Funcional (BCC) - 1° Período

Gabarito da 1a. Avaliação – 30 pontos

1) Avalie as funções definidas abaixo e responda marcando um X sobre a alternativa correta: **(6 pontos)**

```
resultado::Int->String
resultado n
   | n >= 60 = "aprovado"
   | otherwise = "reprovado"
   | n < 60 && n >= 45 = "segunda epoca"

f1::Bool->Bool
f1 v | True = True
     | False = False

f2::Bool->Bool
f2 v = (v==v)

f3::Bool->Bool
f3 v = not (not v)

dvs :: Integer -> [(Integer,Integer)]
dvs n = [ (d, div n d) | d <- [1..k], mod n d == 0 ]
          where k = floor (sqrt (fromInteger n))

reverso::[a]->[a]
reverso [] = []
reverso (h:t) = ?????
                where primeiro = [h]
                      resto = t

geraLista1 = [x>6| x <- [1..10]]
geraLista2 = [(x,y,x==y)|x<-[4..6],y<-[5..6]]
```

A) Quando é que se tem **`resultado n = "segunda epoca"`**? Nunca

B) Entre as funções **`f1`**, **`f2`** e **`f3`**, quais delas possuem o mesmo comportamento? f1 e f2

C) O resultado da chamada **`dvs 9`** é: [(1,9),(3,3)]

D) O que falta em ???? para que a função **`reverso`** defina o código para inverter os elementos de uma lista? [h] ++ reverso t

E) Quais os respectivos retornos das funções **`geraLista1`** e **`geraLista2`** ?

[False,False,False,False,False,False,True,True,True,True]
[(4,5,False), (4,6,False), (5,5,True), (5,6,False), (6,5,False), (6,6,True)]

2) Sejam as funções abaixo. Explique a finalidade de cada uma e mostre a execução passo-a-passo da chamada indicada. **(4 pontos)**

```
tira :: Int -> [a] -> [a]
tira 0 s = []
tira (n+1) (x:xs) = x: tira n xs

cria :: Int -> [Int]
cria v = v: cria (v+1)
```

> tira 3 (cria 50)

cria 50
50: cria 51
50: (51: cria52)
50: (51: (52: cria 53))
50: (51: (52: (53: cria 54)))
....
A função cria define a criação de uma lista infinita a partir de um dado número inteiro.

tira 3 (50: (51: (52: (53: cria 54))))
50 : tira 2 (51: (52: (53: cria 54)))
51: tira 1 (52: (53: cria 54))
52: tira 0 (53: cria 54)
[ ]
==> [50,51,52]

A função tira devolve os n primeiros elementos de uma lista, mesmo sendo ela infinita.

3) Faça duas funçoes em linguagem Haskell, uma recursiva tradicional e outra recursiva em cauda, para contar o número de ocorrências de um elemento numa lista. A função recebe um elemento e uma lista qualquer e retorna um valor inteiro. **(6 pontos)**

```
> contaOcorrencias 10 [4,6,10,3,15,3,10,7]
2
```

```
contaOc1  n [ ] = 0
contaOc1  n (h :t)  =  if ( n == h)  then  1 + contaOc1  n  t
                                     else   contaOc1  n  t
```

```
contaOc2   n  lista =  contaOc2Aux  n  lista  0
    where
       contaOc2Aux  n [ ]  res  =  res
       contaOc2Aux  n  (h : t)  res  =  if ( n == h)  then   contaOc2Aux  n  t (res+1)
                                                      else   contaOc2Aux   n  t  res
```

4) Faça um programa que receba duas listas e retorne verdadeiro se a primeira lista for uma subsequencia da segunda. **(5 pontos)**

```
> subseq [6,9] [3,3,6,9,12]
True
```

```
subseq::Eq x => [x]->[x]->Bool
subseq [] ys = True
subseq (x:xs) [] = False
subseq (x:xs) (y:ys) = (x == y && subseq2 xs ys) || subseq (x:xs) ys


subseq2 [] ys = True
subseq2 xs [] = False
subseq2 (x:xs) (y:ys) = if (x==y)  then  subseq2 xs ys
                                   else False
```

5) Seja o programa abaixo. Analise o código e explique a sua funcionalidade. Faça um teste mostrando um exemplo de aplicação da função e indique o resultado obtido.**(5 pontos)**

```
myst::[a]->[a]->[a]
myst l1 l2 = mystq l1 l2 []
        where mystq [] _ x = x
              mystq _ [] x = x
              mystq (h:t) (h1:t1) z = mystq t t1 (z++[h]++[h1])
```

A função **myst** intercala os valores de duas listas, enquanto ambas possuem o mesmo tamanho:

```
> myst [1,2,3] [4,6]
 mystq [1,2,3] [4,6] [ ]
 mystq [2,3] [6] ([ ] ++ [1] ++ [4]) => mystq [2,3] [6] [1,4]
 mystq [3] [ ] ([1,4] ++ [2] ++ [6]) => mystq [3] [ ] [1,4,2,6]
 [1,4,2,6]
```

6) Faça um programa em Haskell para multiplicar por 3 os elementos pares positivos de uma lista. Os demais elementos devem ser mantidos na lista resultante, nas suas devidas posições.**(4 pontos)**

```
mult [ ] = [ ]
mult (h : t) = if (h > 0) && (mod h 2 == 0)
               then h*3 :  mult  t
               else   h : mult t
```

Boa Prova!!