Universidade Federal do Ceará - Campus de Quixadá
Programação Funcional
1° Avaliação Parcial 01/04/2015
Nome:

1) (2pt) Defina cada uma das seguintes funções usando apenas funções pré-definidas do módulo Prelude:

a) Defina a função final tal que (final n xs) é a lista formada pelos n elementos finais de xs. Por exemplo,

```
final 3 [2,5,4,7,9,6] == [7,9,6]
```

b) Defina a função rota tal que (rota n xs) é uma lista formada colocando n primeiros elementos de xs no final da lista. Por exemplo,

```
rota 1 [3,2,5,7] == [2,5,7,3]
rota 2 [3,2,5,7] == [5,7,3,2]
rota 3 [3,2,5,7] == [7,3,2,5]
```

2) (2pt) Defina as seguintes funções usando compreensão de listas:

a) (1pt) `interseccao :: Eq a => [a] -> [a] -> [a]` tal que (interseccao xs ys) é a interseccao dos conjuntos xs e ys. Por exemplo,

```
interseccao [3,2,5] [5,7,3,4] == [3,5]
```

b) (1pt)`diferencia :: Eq a => [a] -> [a] -> [a]` tal que (diferencia xs ys) é a diferença entre os conjuntos xs e ys. Por exemplo,

```
diferencia [3,2,5,6] [5,7,3,4] == [2,6]
diferencia [3,2,5] [5,7,3,2] == []
```

3)Escreva uma definição recursiva para as seguintes funções:

a) (1pt) A função `duplicar :: String -> String` repete duas vezes cada vogal (letras 'a', 'e', 'i', 'o', 'u' minúsculas ou maiúsculas) numa cadeia de carateres; os outros carateres devem ficar inalterados.

Exemplo: `duplicar "Ola, mundo!"== "OOlaa, muundoo!"`

Dica: Crie uma lista com as vogais minúsculas e maiúsculas.

b) (1pt) A função `aplica :: [a -> a] -> a -> [a]` recebe uma lista de funções e um valor retornando uma lista com os resultados das aplicações das funções. Por exemplo,

```
aplica [(*4), (+2), (-4)] 2 == [8,4,-2]
```

4) (2pt) Escreva uma função `paridade :: [Bool] -> Bool` que calcule a paridade de uma sequência de bits (representados como uma lista de boleanos): se o número de bits de valor True for ímpar então a paridade é True, caso contrário é False.

Exemplo: `paridade [True,True, False,True] = True`

a) (1pt) Escreva uma definição recursiva para a função paridade.

b) (1pt) Escreva uma definição usando o foldr para a função paridade.

5) (2pt) A função length, que computa o número de elementos de uma lista, pode ser definida do seguinte modo:

```
length xs = length' 0 xs
            where length' n [] = n
length' n (x:xs) = length' (n+1) xs
```

Essa função usa a função auxiliar length', que possui um parâmetro adicional para acumular o resultado. Defina recursivamente em Haskell função media que recebe uma lista de valores numéricos como parêmetro e retorna a média desses valores.

Obs1: em ambos os casos, percorra a lista apenas uma vez para isso uma função auxiliar(ou seja, não use uma função que percorre a lista uma vez para calcular a soma dos valores da lista e outra vez para determinar o número de elementos da lista).

Obs2: Você pode supor que a lista não é vazia.

Obs3: Use fromIntegral para converter um inteiro em um valor da classe Rational (para poder usar o operador / de divisão não-inteira).

6)(2pt) Escreva uma função diamonds que receba como argumento um valor inteiro positivo n e retorne uma lista de listas $[l_1, l_2, \ldots, l_{n-1}, l_n, l_{n-1}, \ldots, l_2, l_1]$, onde l_i é a lista dos i múltiplos de i (sendo i o primeiro múltiplo de i).

Exemplo:

```
diamonds 2  [[1],[2,4],[1]]
diamonds 3  [[1],[2,4],[3,6,9],[2,4],[1]]
diamonds 4  [[1],[2,4],[3,6,9],[4,8,12,16],[3,6,9],[2,4],[1]]
```

7) (2pt) A função `intersperse :: a -> [a] -> [a]` que intercala um elemento entre valores consecutivos numa lista; se a lista tiver menos de dois valores deve ficar inalterada. Exemplos:

```
> intersperse 0 [1..4]
[1,0,2,0,3,0,4]
> intersperse ',' "abcd"
"a,b,c,d"
[]
> intersperse ',' "a"
"a"
```

Dica: Use o padrão (x:y:xs)

8) (2pt) Defina uma função `group` em Haskell que recebe como argumento uma lista l de valores e agrupa os valores repetidos, retornando uma lista de pares nos quais o primeiro componente é um valor de l e o segundo é o número de repetições consecutivas desse valor.

Por exemplo:
`group [1,1,2,2,2,3,4,5,5,1] retorna [(1,2),(2,3),(3,1),(4,1),(5,2),(1,1)]`

a) (0,5pt) Usando a função takeWhile, encontre uma sub-cadeia inicial que contém todos os números iguais.

b) (0,5pt) Usando a função dropWhile, remova uma subcadeia inicial que contém todos os números iguais.

c) (1pt) Escreva uma definição recursiva para a função group.

9) (2pt) A função `scanSum :: [Int] -> [Int]` recebe uma lista de inteiros e retorna uma lista com as somas das somas acumuladas. Por exemplo,

```
scanSum [1,2,3,4] == [1,3,6,10]
scanSum [1,3,5,7] == [1,4,9,16]
```

a) (1pt) Defina uma função prefixos :: [Int] -¿ [[Int]] que retorna uma lista de prefixos de todos os prefixos de uma lista.

```
prefixos [1,2,3,4] == [ [1], [1,2], [1,2,3], [1,2,3,4] ]
```

```
prefixos []  = --------
prefixos xs  = prefixos  ---------------------- ++ [xs]
```

b) (1pt) Defina scanSum usando compreensão de listas com o auxílio da função prefixos.

c) (Desafio) Defina a função scanSum recursivamente usando uma função auxiliar.