Centro de Informática - UFPE
Primeiro Exercício Escolar — IF686 (2018.2)
Data: 11/10/2018
Docente: Márcio Lopes Cornélio
Discente:

Turma: E5
Horário: 15h-17h

Instruções para entrega: Enviar dois arquivos para *mlc2@cin.ufpe.br* : (1) arquivo .hs, com todas as respostas (copiar e colar os códigos das respostas) e (2) um arquivo no formato pdf com todas as respostas. Escrever o nome na primeira linha dos arquivos. O assunto do email deve estar no formato: "Nome completo 1ee_plc_2018_2"

(1,5) 1. Dada uma função f :: **Ord** a =>[a] -> (a -> [b]) -> [[b]], determine o tipo da expressão f . **map** (+1). Desenvolva a solução.

2. Dado o tipo algébrico Nat

```
data Nat = Zero | Succ Nat deriving (Eq,Show)
```

Defina:

(0.5) (a) Uma função que converte números inteiros em números naturais.

(0.5) (b) Uma função que converte números naturais em números inteiros.

(0.5) (c) Defina a função soma que soma dois números naturais

(0.5) (d) Defina a função mult que multiplica dois números naturais

3. Desenvolva o que se pede.

(0,5) (a) Defina um tipo de dados Tree correspondente às árvores binárias cujos nós contêm valores que podem ser comparados através de operadores relacionais básicos.

(1,5) (b) Implemente uma função chamada buildSearchTree que, dada uma lista de valores para os quais os operadores relacionais básicos estão definidos, constroi uma áarvore de busca (não necessariamente balanceada).

(1,5) (c) Implemente uma função chamada searchTreeSort que, dada uma lista de valores para os quais os operadores relacionais básicos estão definidos, produz uma lista que contém todos os elementos da lista fornecida como entrada, ordenados do menor para o maior, e necessariamente usando a função buildSearchTree para alcançar este fim.

(3,0) 4. Construa uma função listPartitioner :: **Num** a =>[a] -> ([a]->[[a]]) que, dada uma lista l de tamanho n contendo apenas números não-duplicados, devolve uma outra funçãoo que, ao ser chamada com uma lista l' de tamanho m $\geq$ n contendo apenas números como argumento, particiona os elementos de l' em regiões (listas) de acordo com os valores dos elementos de l. O particionamento consiste em criar listas contendo elementos de l'que são menores ou iguais a um elemento de l, mais uma lista contendo apenas os elementos maiores que qualquer elemento de l, se houver algum. O resultado da função é uma lista contendo todas as listas produzidas. Os seguinte exemplo ilustra o funcionamento de listPartitioner. Dada uma lista [4, 13, 8], a chamada

```
listPartitioner [4,13,8]
```

devolve uma funçãao que, se for chamada com a lista [1..15], devolve como resultado a lista [[1,2,3,4], [5,6,7,8], [9,10,11,12,13], [14,15]]. No exemplo acima, [1,2,3,4] é a lista com todos os elementos da lista [1..15] menores ou iguais a 4, [5,6,7,8] é a lista com todos os elementos menores ou iguais a 8, [9,10,11,12,13] é a lista com todos os elementos menores ou iguais a 13 e [14,15] é lista com elementos maiores que o maior elemento da lista [4,13,8]. Note que nenhuma das listas recebidas como entrada precisa estar ordenada (mas as saídas precisam).

```
(.) :: (b-> c) -> (a -> b) -> a -> c
map :: (a->b)->[a]->[b]
filter :: (a -> Bool) -> [a] -> [a]
foldr :: (a -> b -> b) -> b -> [a] -> b
foldr1 :: (a -> a -> a) -> [a] -> a
and :: [Bool] -> Bool
```