Lista de Exercícios III

Universidade Federal do Ceará
Campus de Quixadá
Ciência da Computação
Programação Funcional
Prof.º Ricardo Reis

27 de Maio de 2014

Construa e teste modelos de dados em Haskell que modelem os tipos de dados a seguir,

1. COMPLEX: Representa números complexos e seus respectivos operadores aritméticos
   Especificação,

```
data Complex = Complex { real :: Float
                       , img  :: Float
                       }
instance Num Complex
instance Eq Complex
instance Fractional Complex
instance Show Complex
```

   Exemplo de saída: `5.0 + 6.2j`

2. STACK: Representa uma pilha genérica e seus operadores fundamentais.
   Especificação,

```
data Stack a = Empty | Top a (Stack a)

push :: a -> Stack a -> Stack a
pop :: Stack a -> Stack a
height :: Stack a -> Int
top:: Stack a -> Maybe a
isEmpty:: Stack a -> Bool

instance Show Stack
```

   Exemplo de saída: `Pilha de altura 5 e topo 12`

3. QUEUE: representa uma fila genérica e seus operadores fundamentais.
   Especificação,

```
data Queue a = Empty | Start a (Queue a)

startQueue :: Queue a -> Maybe a
endQueue :: Queue a -> Maybe a
pushQueue :: a -> Queue a -> Queue a
popQueue :: Queue a -> Queue a
isEmptyQueue :: Queue a -> Bool
lenQueue :: Queue a -> Int

-- Enquanto uma dada ila de entrada
-- não for vazia então o elemento
-- inicial (startQueue) é removido e
-- processado por uma função de entrada.
-- A saída é a lista dos valores obtidos
whileNotEmpty :: (a -> b) -> Queue a -> [b]

instance Show Queue
```

4. MATRIX: representa uma matriz numérica e algumas de suas operações fundamentais.
   Especificação,

```
type Row = [Float]
data Matrix = Matrix { ncols :: Int
                     , nrows :: Int
                     , rows  :: [Row]
                     }

-- matriz de zeros
zeroMatrix :: Int -> Int -> Matrix
-- matriz de uns
oneMatrix :: Int -> Int -> Matrix
-- matriz identidade : recebe ordem
identMatrix :: Int -> Matrix
-- soma duas matrizes
sumMatrix :: Matrix -> Matrix -> Matrix
-- produto de escalar por matriz
prodScalar :: Float -> Matrix -> Matrix
-- produto entre matrizes
prodMatrix :: Matrix -> Matrix -> Matrix
-- transforma listas de listas de
-- floats numa matriz
listToMatrix :: [Row] -> Matrix

instance Show Matrix
```

5. DATE: representa uma data com campos de dia, mês e ano. A especificação seguinte é incompleta.
   Especificação,

```
data Day = ...
data Month = ...
data Year = ...
data Date = ...

-- 'Date' deve ser instância de
-- Eq, Ord e Show
-- 'Month' deve instância de
-- Eq, Ord, Show, Enum, Bounded
```

```
sortListDates :: [Date] -> [Date]
```

Exemplo de saída de `date`:
`5 de Janeiro de 2014`

6. PESSOA: representa uma pessoa trazendo informações de nome, idade e salário
   Especificação,

```
data Pessoa = { nome :: String
              , idade :: Int
              , salario :: Float }

data Criterio = ByNome | ByIdade | BySalario

-- classifica lista de pessoa por critério
sortListPessoa :: [Pessoa]
               -> Criterio
               -> [Pessoa]

instance Show Pessoa
```

7. OLIST: representa uma lista ligada ordenada, ou seja, uma lista que mantém suas chaves ordenadas durante seu ciclo de vida se requerer a funções de ordenação.
   Especificação,

```
data OList a = Empty | Node a (OList a)

-- insere em lista ordenada
infixr 5 >>>
(>>>) :: (Ord a) => a -> OList a -> OList a

-- indica se chave está ou não numa lista
hasKey :: (Ord a) => a -> OList a -> Bool

-- remove chave de lista ordenada
remKey :: (Ord a) => a -> OList a -> OList a

-- identifica n-ésima chave de lista ordenada
key :: Int -> OList a -> Maybe a

instance Show OList
```