Programação Funcional
Lista 03
Tipo Abstrato de Dados

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

1. FRAÇÃO: Representa os números fracionários.

```
data Fracao = F Int Int
instance Num Fracao
instance Eq Fracao
instance Show Fracao
```

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

```
data Stack a = Empty | Top a (Stack a)
push :: a -> Stack a -> Stack a
pop :: Stack a -> Maybe Stack a
height :: Stack a -> Int
top:: Stack a -> Maybe a
empty :: Stack a
isEmpty:: Stack a -> Bool

instance Show Stack
```

Exemplos:

```
push (Top 4 Empty)  == Top 4 Empty
push 5 (Top 4 Empty) == Top 5 (Top 4 Empty)
pop Empty     == Nothing
pop (Top 4 Empty) == Just Empty
height Empty     == 0
height (Top 4 Empty) == 1
isEmpty Empty == False
isEmpty Top 4 Empty
top Empty = Nothing
top (Top 4 Empty) == Just 4
show (Top 4 Empty) == "Pilha de altura 1 e topo 4"
show (Empty) == "Pilha de altura 0"
```

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

```
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 Fila 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.

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

```
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
```

6. 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.

```
data OList a = Empty | Node a (OList a)
-- insere em lista ordenada
insere :: (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
```

7. POLINÔMIO: representa um polinômio

```
data Polinomio a = PolZero
                 | ConsPol Int a (Polinomio a) deriving Eq
polZero :: Polinomio a
ehPolZero :: Num a => Polinomio a -> Bool
consPol :: Num a => Int -> a -> Polinomio a -> Polinomio a
grau :: Polinomio a -> Int
coefLider :: Num a => Polinomio a -> a
restoPol :: Polinomio a -> Polinomio a
instance Show Polinomio
instance Num Polinomio
```

onde o significado das operações é:

- polZero devolve o polinômio Zero
- (ehPolZero p) verifica se p é o polinômio Zero
- (consPol n b p) devolve o polinômio $bx^n + p$
- (grau p) devolve o grau do polinômio.
- (coefLider p) é o coeficiente do maior expoente do polinômio.
- (restoPol) é o polinômio obtido pela remoção do coeficiente líder.

```
pol1 = consPol 4 3 (consPol 2 (-5) (consPol 0 3 polZero))
pol2 = consPol 5 1 (consPol 2 5 (consPol 1 4 polZero))
pol3 = consPol 4 6 (consPol 1 2 polZero)

show pol1 == "3*x^4 + -5*x^2 + 3"
show pol2 == "x^5 + 5*x^2 + 4*x"
show pol3 == "6*x^4 + 2*x"
```

8. POLINÔMIO: um polinômio pode ser representado através de uma lista de pares (coeficiente, grau) ordenados em ordem decrescente de grau. Por exemplo, o polinômio $6x^4 - 5x^2 + 4x - 7$ é representado por [(6,4),(-5,2),(4,1),(-7,0)]. Essa representação se chama lista densa.

```
data Polinomio a = Pol [(a,Int)] deriving Eq
polZero :: Polinomio a
ehPolZero :: Num a => Polinomio a -> Bool
consPol :: Num a => Int -> a -> Polinomio a -> Polinomio a
grau :: Polinomio a -> Int
coefLider :: Num a => Polinomio a -> a
restoPol :: Polinomio a -> Polinomio a
instance Show Polinomio
instance Num Polinomio
```

9. CONJUNTO: O tipo conjunto pode ser representado como uma lista ordenada sem repetições com a seguinte especificação:

```
data Conj a = Cj [a] deriving Eq
--devolve o conjunto Vazio
vazio :: Conj a
--verifica se é o conjunto vazio
esVazio :: Conj a -> Bool
--verifica se um elemento pertence ao conjunto
pertence :: Ord a => a -> Conj a -> Bool
--insere um novo elemento no conjunto
insere   :: Ord a => a -> Conj a -> Conj a
--remove um elemento do conjunto
elimina  :: Ord a => a -> Conj a -> Conj a
--subconjunto p q testa se p é subconjunto de q
subconjunto :: Ord a => Conj a -> Conj a -> Bool
--subconjuntoProprio p q testa se p é subconjuntoProprio de q
subconjuntoProprio :: Ord a => Conj a -> Conj a -> Bool
--cardinal devolve a cardinalidade de um conjunto
cardinal :: Conj a -> Int
-- uniao p q devolve a uniao dos dois conjuntos
uniao :: Ord a => Conj a -> Conj a -> Conj a
--uniaoLista xs devolve a união de todos os conjuntos de xs
uniaoLista :: Ord a => [Conj a] -> Conj a
-- intersecao p q devolve a intersecção de p e q
interseccao :: Eq a => Conj a -> Conj a -> Conj a
-- disjuntos p q verifica se p e q são disjuntos
```

```
disjuntos :: Ord a => Conj a -> Conj a -> Bool
-- diferenca p q devolve a diferenca entre p e q
diferenca :: Eq a => Conj a -> Conj a -> Conj a
-- fitraConj pred p devolve um conjunto com elementos que satisfazem o predicado pred.
filtraConj :: (a -> Bool) -> Conj a -> Conj a
--mapConj f p devolve um conjunto cujos elementos são obtidos a partir da aplicação de f aos elementos de p.
mapConj :: (a->b) -> Conj a -> Conj b
-- potencia p devolve o conjuntos das partes ou potência de p
potencia :: Ord a => Conj a -> Conj (Conj a)
```

10. CONJUNTO: O tipo conjunto pode ser representado como uma árvore de pesquisa sem repetições.
11. COMPLEX : Representa os números complexos e seus respectivos operadores aritméticos

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