```haskell
-- Laboratorul 8

-- Exercitiul 1

-- Se dau următoarele tipuri de date ce reprezinta puncte cu numar variabil de
-- coordonate intregi:

data Punct = Pt [Int]

-- Arbori cu informatia în frunze și clasă de tipuri ToFromArb

data Arb = Vid | F Int | N Arb Arb
    deriving Show

class ToFromArb a where
    toArb :: a -> Arb
    fromArb :: Arb -> a

-- a) Să se scrie o instantă a clasei Show pentru tipul de date Punct, astfel
-- încât lista coordonatelor sa fie afisată sub forma de tuplu.

instance Show Punct where
    show :: Punct -> String
    show (Pt []) = "()"
    show (Pt lista) = "(" ++ show (head lista) ++ concat [virgula : show (numar)
| (virgula, numar) <- zip[',', ',' ..](tail lista)] ++ ")";

-- b) Să se scrie o instanță a clasei ToFromArb pentru tipul de date Punct astfel
-- incat lista coordonatelor punctului sa coincidă cu frontiera arborelui.
-- toArb (Pt [1,2,3])
-- N (F 1) (N (F 2) (N (F 3) Vid))
-- fromArb $ N (F 1) (N (F 2) (N (F 3) Vid)) :: Punct
-- (1,2,3)

myToList :: Punct -> [Int]
myToList (Pt []) = []
myToList (Pt (x:xs)) = x : myToList (Pt xs)

instance ToFromArb Punct where
    fromArb :: Arb -> Punct
    fromArb Vid = Pt []
    fromArb (F a) = Pt [a]
    fromArb (N littleTree biggerTree) = Pt(myToList(fromArb littleTree) ++
myToList(fromArb biggerTree))
```

```haskell
    toArb :: Punct -> Arb
    toArb (Pt []) = Vid
    toArb (Pt [x]) = (F x)
    toArb (Pt (x:xs)) = N (F x) (toArb(Pt xs))

-- Exercitiul 2

-- Se dă următorul tip de date reprezentând figuri geometrice.

data Geo a = Square a | Rectangle a a | Circle a
    deriving Show

-- Si clasa GeoOps în care se definesc operatiile perimeter si area.

class GeoOps g where
    perimeter :: (Floating a) => g a -> a
    area :: (Floating a) => g a -> a

-- a) Să se instantieze clasa GeoOps pentru tipul de date Geo. Pentru valoarea pi
-- există functia cu acelasi nume (pi).
-- ghci> pi
-- 3.141592653589793

instance GeoOps Geo where
    perimeter :: Floating a => Geo a -> a
    perimeter (Square l) = 4*l
    perimeter (Rectangle l lm) = 2*(l+lm)
    perimeter (Circle r) = 2*pi*r

    area :: Floating a => Geo a -> a
    area (Square l) = l*l
    area (Rectangle l lm) = l*lm
    area (Circle r) = pi * r * r

-- b) Să se instantieze clasa Eq pentru tipul de date Geo, astfel încât două
-- figuri geometrice să fie egale dacă au perimetrul egal.

instance (Floating a, Eq a) => Eq (Geo a) where
    (==) :: (Floating a, Eq a) => Geo a -> Geo a -> Bool
    fig1 == fig2 = perimeter fig1 == perimeter fig2
```