```haskell
-- Laboratorul 6

-- Exercitiul 1

-- Vom începe prin a scrie câteva functii definite folosind tipul de date Fruct:

data Fruct
  = Mar String Bool
  | Portocala String Int

-- O expresie de tipul Fruct este fie un Mar String Bool sau o Portocala String
-- Int. Vom folosi un String pentru a indica soiul de mere sau portocale, un Bool
-- pentru a indica dacă mărul are viermi si un Int pentru a exprima numărul de
-- felii dintr-o portocală.

-- De exemplu:

ionatanFaraVierme = Mar "Ionatan" False
goldenCuVierme = Mar "Golden Delicious" True
portocalaSicilia10 = Portocala "Sanguinello" 10
listaFructe = [Mar "Ionatan" False,
               Portocala "Sanguinello" 10,
               Portocala "Valencia" 22,
               Mar "Golden Delicious" True,
               Portocala "Sanguinello" 15,
               Portocala "Moro" 12,
               Portocala "Tarocco" 3,
               Portocala "Moro" 12,
               Portocala "Valencia" 2,
               Mar "Golden Delicious" False,
               Mar "Golden" False,
               Mar "Golden" True]

-- a) Scrieti o functie care indică dacă un fruct este o portocală de Sicilia sau
-- nu. Soiurile de portocale din Sicilia sunt Tarocco, Moro si Sanguinello.
-- test_ePortocalaDeSicilia1 = ePortocalaDeSicilia (Portocala "Moro" 12) == True
-- test_ePortocalaDeSicilia2 = ePortocalaDeSicilia (Mar "Ionatan" True) == False

ePortocalaDeSicilia :: Fruct -> Bool
ePortocalaDeSicilia (Mar soi viermi) = False
ePortocalaDeSicilia (Portocala soi felii) = soi `elem` ["Moro", "Tarocco",
"Sanguinello"]

ePortocalaDeSicilia2 :: Fruct -> Bool
ePortocalaDeSicilia2 (Mar _ _) = False
```

```haskell
ePortocalaDeSicilia2 (Portocala s _)
    | s `elem` ["Tarocco","Moro","Sanguinello"] = True
    | otherwise = False

-- b) Scrieti o functie care calculează numărul total de felii ale portocalelor
-- de Sicilia dintr-o listă de fructe.
-- test_nrFeliiSicilia = nrFeliiSicilia listaFructe == 52

nrFeliiPortocala :: Fruct -> Int
nrFeliiPortocala(Mar soi viermi) = 0
nrFeliiPortocala(Portocala soi felii) = felii

nrFeliiSicilia :: [Fruct] -> Int
nrFeliiSicilia []  = 0
nrFeliiSicilia (h:t) = if ePortocalaDeSicilia h then nrFeliiPortocala h +
nrFeliiSicilia t
                       else nrFeliiSicilia t

-- c) Scrieti o functie care calcuelază numărul de mere care au viermi dintr-o
-- lista de fructe.
-- test_nrMereViermi = nrMereViermi listaFructe == 2

areViermi :: Fruct -> Bool
areViermi(Mar soi viermi) = viermi
areViermi(Portocala soi felii) = False

nrMereViermi :: [Fruct] -> Int
nrMereViermi [] = 0
nrMereViermi (h:t) = if areViermi h then nrMereViermi t + 1
                     else nrMereViermi t


-- Exercitiul 2

type NumeA = String
type Rasa = String
data Animal = Pisica NumeA | Caine NumeA Rasa
    deriving Show

-- a) Scrieti o functie care întoarce "Meow!" pentru pisică si "Woof!" pentru
-- câine.

vorbeste :: Animal -> String
vorbeste (Pisica numePisica) = "Meow!"
vorbeste (Caine numeCaine rasaCaine) = "Woof!"
```

```haskell
-- Vă reamintiti tipul de date predefinit Maybe:
-- data Maybe a = Nothing | Just a
-- b) Scrieti o functie care întoarce rasa unui câine dat ca parametru sau
-- Nothing dacă parametrul este o pisică

rasaCaine :: Animal -> Rasa
rasaCaine (Caine numeCaine rasaCaine) = rasaCaine

rasa :: Animal -> Maybe String
rasa (Pisica numePisica) = Nothing
rasa (Caine numeCaine rasaCaine) = Just rasaCaine


-- Exercitiul 3

-- Se dau urmatoarele tipuri de date ce reprezintă matrici cu linii de lungimi
-- diferite:

data Linie = L [Int]
   deriving Show
data Matrice = M [Linie]
   deriving Show

-- a) Scrieti o functie care verifica daca suma elementelor de pe fiecare linie
-- este egala cu o valoare n. Rezolvati cerinta folosind foldr.
-- test_verif1 = verifica (M[L[1,2,3], L[4,5], L[2,3,6,8], L[8,5,3]]) 10 == False
-- test_verif2 = verifica (M[L[2,20,3], L[4,21], L[2,3,6,8,6], L[8,5,3,9]]) 25 ==
-- True

sumaLinii :: Matrice -> [Int]
sumaLinii (M []) = []
sumaLinii (M (L h:t)) = sum h : sumaLinii(M t)

verifica :: Matrice -> Int -> Bool
verifica (M mat) n = foldr (&&) True [if x == n then True else False | x<-
sumaLinii(M mat)]

-- b) Scrieti o functie doarPozN care are ca parametru un element de tip Matrice
-- si un numar intreg n, si care verifica daca toate liniile de lungime n din
-- matrice au numai elemente strict pozitive.
-- testPoz1 = doarPozN (M [L[1,2,3], L[4,5], L[2,3,6,8], L[8,5,3]]) 3 == True
-- testPoz2 = doarPozN (M [L[1,2,-3], L[4,5], L[2,3,6,8], L[8,5,3]]) 3 == False

extragereLinii :: Matrice -> [[Int]]
```

```haskell
extragereLinii (M []) = []
extragereLinii (M (L h:t)) = h : extragereLinii(M t)

doarPozN :: Matrice -> Int -> Bool
doarPozN (M mat) n = foldr (&&) True [if length(filter(>0) linie) == n then True
else False | linie <- extragereLinii(M mat), length linie == n]

-- c) Definiti predicatul corect care verifică dacă toate liniile dintr-o matrice
-- au aceeasi lungime.
-- testcorect1 = corect (M[L[1,2,3], L[4,5], L[2,3,6,8], L[8,5,3]]) == False
-- testcorect2 = corect (M[L[1,2,3], L[4,5,8], L[3,6,8], L[8,5,3]]) == True

extragereDimensiune :: Matrice -> Int
extragereDimensiune (M []) = 0
extragereDimensiune (M (L h:t)) = length h

corect :: Matrice -> Bool
corect (M mat) = foldr (&&) True [if length linie == extragereDimensiune(M mat)
then True else False | linie <- extragereLinii (M mat)]
```