```haskell
-- Tema Laboratorul 3

import Data.Char

-- 1. Sa se scrie o functie nrVocale care pentru o lista de siruri de caractere, calculeaza numarul
-- total de vocale ce apar în cuvintele palindrom. Pentru a verifica daca un sir e palindrom,
-- puteti folosi functia reverse, iar pentru a cauta un element într-o lista puteti folosi functia
-- elem. Puteti defini oricâte functii auxiliare.

-- Varianta 1

nrVocale :: [String] -> Int
nrVocale [] = 0
nrVocale (h:t)
    | h == reverse h = nrVocale t + countVocale h
    | otherwise = nrVocale t

countVocale :: [Char] -> Int
countVocale [] = 0
countVocale (h:t)
    | h == 'a' = countVocale t + 1
    | h == 'e' = countVocale t + 1
    | h == 'i' = countVocale t + 1
    | h == 'o' = countVocale t + 1
    | h == 'u' = countVocale t + 1
    | h == 'A' = countVocale t + 1
    | h == 'E' = countVocale t + 1
    | h == 'I' = countVocale t + 1
    | h == 'O' = countVocale t + 1
    | h == 'U' = countVocale t + 1
    | otherwise = countVocale t

-- Varianta 2

nrVocale2 :: [String] -> Int
nrVocale2 [] = 0
nrVocale2 (h:t)
    | h == reverse h = nrVocale2 t + countVocale2 h
    | otherwise = nrVocale2 t

countVocale2 :: [Char] -> Int
countVocale2 [] = 0
```

```haskell
countVocale2 (h:t)
    | elem h "aeiouAEIOU" = countVocale2 t + 1
    | otherwise = countVocale2 t

-- Varianta 3

nrVocale3 :: [String] -> Int
nrVocale3 l = sum[countVocale3 x | x<-l, x == reverse x]

countVocale3 :: [Char] -> Int
countVocale3 s = sum [1 | c<-s, c `elem` "aeiouAEIOU"]


-- 2. Sa se scrie o functie care primeste ca parametru un numar si o lista de
întregi, si adauga
-- elementul dat dupa fiecare element par din lista. Sa se scrie si prototipul
functiei.

f :: Int -> [Int] -> [Int]
f n [] = []
f n (h:t)
    | even h = h : n : f n t
    | otherwise = h : f n t


-- 3. Sa se scrie o functie care are ca parametru un numar întreg si determina
lista de divizori ai
-- acestui numar. Sa se scrie si prototipul functiei.

divizori :: Int -> [Int]
divizori n = [x | x <- [1..n], mod n x == 0]


-- 4. Sa se scrie o functie care are ca parametru o lista de numere întregi si
calculeaza lista listelor
-- de divizori.

listadiv :: [Int] -> [[Int]]
listadiv l = [divizori x | x <- l]

-- 5. Scrieti o functie care date fiind limita inferioara si cea superioara
(întregi) a unui interval
-- închis si o lista de numere întregi, calculeaza lista numerelor din lista care
apartin intervalului.
```

```haskell
-- a. Folositi doar recursie. Denumiti functia inIntervalRec.

inIntervalRec :: Int -> Int -> [Int] -> [Int]
inIntervalRec x y [] = []
inIntervalRec x y (h:t)
    | elem h [x..y] = [h] ++ inIntervalRec x y t
    | otherwise = inIntervalRec x y t

-- b. Folositi descrieri de liste. Denumiti functia inIntervalComp.

inIntervalComp :: Int -> Int -> [Int] -> [Int]
inIntervalComp s d l = [x | x <- l, x >= s && x <= d ]

-- 6. Scrieti o functie care numara câte numere strict pozitive sunt într-o lista
data ca argument.

-- a. Folositi doar recursie. Denumiti functia pozitiveRec.
pozitiveRec :: [Int] -> Int
pozitiveRec [] = 0
pozitiveRec (h:t)
    | h > 0 = pozitiveRec t + 1
    | otherwise = pozitiveRec t

-- b. Folositi descrieri de liste. Denumiti functia pozitiveComp.
-- Nu puteti folosi recursie, dar veti avea nevoie de o functie de agregare.

pozitiveComp :: [Int] -> Int
pozitiveComp l = sum[1 | x<-l, x>0]

-- 7. Scrieti o functie care data fiind o lista de numere calculeaza lista
pozitiilor elementelor impare
-- din lista originala.

-- a. Folositi doar recursie. Denumiti functia pozitiiImpareRec. Indicatie:
folositi
-- o functie ajutatoare, cu un argument în plus reprezentând pozitia curenta din
lista.

pozitiiImpareRec :: Int -> [Int] -> [Int]
pozitiiImpareRec _ [] = []
pozitiiImpareRec i (h:t)
    | odd h = i : pozitiiImpareRec (i+1) t
    | otherwise = pozitiiImpareRec (1+i) t
```

```haskell
-- b. Folositi descrieri de liste. Denumiti functia pozitiiImpareComp.

pozitiiImpareComp :: [Int] -> [Int]
pozitiiImpareComp l = [poz |(elem, poz) <- zip l [0..], odd elem]


-- 8. Scrieti o functie care calculeaza produsul tuturor cifrelor care apar în
sirul de caractere dat ca
-- intrare. Daca nu sunt cifre în sir, raspunsul functiei trebuie sa fie 1 .
-- De exemplu:
-- -- multDigits "The time is 4:25" == 40
-- -- multDigits "No digits here!" == 1

-- a. Folositi doar recursie. Denumiti functia multDigitsRec.

multDigitsRec :: String -> Int
multDigitsRec "" = 1
multDigitsRec (h:t)
    | isDigit h = digitToInt h * multDigitsRec t
    | otherwise = multDigitsRec t

-- b. Folositi descrieri de liste. Denumiti functia multDigitsComp

multDigitsComp :: String -> Int
multDigitsComp s = product[digitToInt x| x <- s, isDigit x]

-- Indicatie: Veti avea nevoie de functia isDigit care verifica daca un caracter
e cifra
-- si functia digitToInt care transforma un caracter in cifra. Cele 2 functii se
afla în
-- pachetul Data.Char.
```