Model Test Laborator

Exerciţiul 1 Scrieți o funcție care numără câte propoziții sunt într-un text dat. Puteți scrie o funcție auxiliară `sfChr` care verifică dacă un caracter e sfârșit de propoziție. Considerăm semne de sfarsit de propozitie: punct '.', semnul intrebarii '?', semnul exclamarii '!', doua puncte ':'.

Puteti folosi doar recursie, functii din categoria A si sfChr

(Ca temă același exercițiu de rezolvat folosind doar descrieri de liste/list comprehensions si functii din categoriile A si B)

Pentru punctaj maxim trebuie să scrieți și prototipul funcțiilor.

Exerciţiul 2 Scrieți o funcție liniiN care are ca parametru o matrice de numere întregi (`[[Int]]`) și un număr întreg n, și verifică dacă toate liniile de lungime n din matrice au doar elemente strict pozitive.

Puteti folosi doar functii din categoriile A, B, C (fara recursie si descrieri de liste și fără funcția `and`)

Exercițiul 3 Se dau următoarele tipuri de date ce reprezintă

Puncte cu număr variabil de coordonate întregi:

```
data Punct = Pt [Int]
           deriving Show
```

Arbori cu informația în frunze şi clasă de tipuri `ToFromArb`

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

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

Să se scrie o instanţă a clasei ToFromArb pentru tipul de date Punct astfel încât 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
Pt [1,2,3]
```

Categoria A. Functii de baza

```
div, mod :: Integral a => a -> a -> a
even, odd :: Integral a => a -> Bool
(+), (*), (-), (/) :: Num a => a -> a -> a
(<), (<=), (>), (>=) :: Ord => a -> a -> Bool
(==), (/=) :: Eq a => a -> a -> Bool
(&&), (||) :: Bool -> Bool -> Bool
not :: Bool -> Bool
max, min :: Ord a => a -> a -> a
isAlpha, isAlphaNum, isLower, isUpper, isDigit :: Char -> Bool
toLower, toUpper :: Char -> Char
```

```
ord :: Char -> Int
chr :: Int -> Char
Intervale
[first..], [first,second..], [first..last], [first,second..last]
```

Categoria B. Functii din biblioteci

```
sum, product :: (Num a) => [a] -> a
sum [1.0,2.0,3.0] = 6.0
product [1,2,3,4] = 24

and, or :: [Bool] -> Bool
and [True,False,True] = False
or [True,False,True] = True

maximum, minimum :: (Ord a) => [a] -> a
maximum [3,1,4,2]  =  4
minimum [3,1,4,2]  =  1

reverse :: [a] -> [a]
reverse "goodbye" = "eybdoog"

concat :: [[a]] -> [a]
concat ["go","od","bye"]  =  "goodbye"

(++) :: [a] -> [a] -> [a]
"good" ++ "bye" = "goodbye"

(!!) :: [a] -> Int -> a
[9,7,5] !! 1  =  7

length :: [a] -> Int
length [9,7,5]  =  3

head :: [a] -> a
head "goodbye" = 'g'

tail :: [a] -> [a]
tail "goodbye" = "oodbye"

init :: [a] -> [a]
init "goodbye" = "goodby"

last :: [a] -> a
last "goodbye" = 'e'

takeWhile :: (a->Bool) -> [a] -> [a]
takeWhile isLower "goodBye" = "good"

take :: Int -> [a] -> [a]
take 4 "goodbye" = "good"

dropWhile :: (a->Bool) -> [a] -> [a]
dropWhile isLower "goodBye" = "Bye"
```

```
drop :: Int -> [a] -> [a]
drop 4 "goodbye" = "bye"

elem :: (Eq a) => a -> [a] -> Bool
elem 'd' "goodbye" = True

replicate :: Int -> a -> [a]
replicate 5 '*' = "*****"

zip :: [a] -> [b] -> [(a,b)]
zip [1,2,3,4] [1,4,9] = [(1,1),(2,4),(3,9)
```

Categoria C. Map, Filter, Fold

```
map :: (a -> b) -> [a] -> [b]
map (+3) [1,2] = [4,5]

filter :: (a -> Bool) -> [a] -> [a]
filter even [1,2,3,4] = [2,4]

foldr :: (a -> b -> b) -> b -> [a] -> b
foldr max 0 [1,2,3,4] = 4

(.) :: (b -> c) -> (a -> b) -> a -> c
($) :: (a -> b) -> a -> b
(*2) . (+3) $ 7 = 20

flip :: (a -> b -> c) -> b -> a -> c
flip (-) 2 3 = 1
```