Laboratorul 5

RECOMANDARE Înainte de a începe să lucrați exercițiile din acest laborator finalizați exercițiile din laboratoarele precedente.

Criptare si Decriptare

În criptografie, cifrul lui Cezar, numit și cifru cu deplasare, codul lui Cezar sau deplasarea lui Cezar, este una dintre cele mai simple și mai cunoscute tehnici de criptare. Este un tip de cifru al substituției, în care fiecare literă din textul inițial este înlocuită cu o literă care se află în alfabet la o distanță fixă față de cea înlocuită. De exemplu, cu o deplasare de cinci poziții în alfabetul limbii române, A este înlocuit cu D, Ă devine E și așa mai departe. Această metodă este numită așa după Iulius Cezar, care o folosea pentru a comunica cu generalii săi. Ideea este simplă: se ia un mesaj pe care dorim să îl criptăm și se deplasează toate literele ccu o anumită valoare între 0 și 26. De exemplu, dacă vrem să criptăm propoziția *"THIS IS A BIG SECRET"* cu deplasarea 5, va rezulta sirul "YMNX NX F GNL XJHWJY". În continuare vom implemeta o variantă a acestui cifru.

Codificarea unui mesaj

Criptarea șirurilor caracter cu caracter poate fi reprezentată de o cheie, folosind o listă de perechi. Fiecare pereche din listă indică pentru o literă care este codificarea ei. De exemplu un cifru pentru literele A-E poate fi dat de lista `[('A','C'), ('B','D'),('C','E'),('D','A'),('E','B')]`.

Exerciții

1. Putem roti o lista luând o parte de la început și adăugând-o la final:

```
Main> rotate 3 "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
"DEFGHIJKLMNOPQRSTUVWXYZABC"
```

Deschideți fișierul `lab5.hs` si completați funcția `rotate :: Int -> [Char] -> [Char]`. Pentru un număr `n`, `n > 0` și `n<lungimea listei`, funcția va roti lista cu n elemente. Funcția trebuie să arunce o eroare daca numărul n este negativ sau prea mare (hint: folosiți funcția `error`).

2. Observați funcția `prop_rotate`. Ce testează? Cum evită această funcție aruncarea erorii?
3. Folosind funcția `rotate`, scrieți o funcție `makeKey :: Int -> [(Char, Char)]` care intoarce cheia de criptare cu o anumită deplasare pentru lista de litere mari ale alfabetului englez.

```
Main> makeKey 5
[('A','F'),('B','G'),('C','H'),('D','I'),('E','J'),('F','K'),
('G','L'),('H','M'),('I','N'),('J','O'),('K','P'),('L','Q'),
('M','R'),('N','S'),('O','T'),('P','U'),('Q','V'),('R','W'),
('S','X'),('T','Y'),('U','Z'),('V','A'),('W','B'),('X','C'),
('Y','D'),('Z','E')]
```

4. Scrieți o funcție `lookUp :: Char -> [(Char, Char)] -> Char` care caută o pereche după prima componentă și întoarce a doua componentă a acesteia. Dacă nu există o pereche cu caracterul căutat pe prima poziție, funcția întoarce caracterul dat ca parametru.

```
Main> lookUp 'B' [('A', 'F'), ('B', 'G'), ('C', 'H')]
'G'
Main> lookUp '9' [('A', 'X'), ('B', 'Y'), ('C', 'Z')]
'9'
```

5. Scrieți o funcție `encipher :: Int -> Char -> Char` care criptează un caracter folosind cheia dată de o deplasare dată ca parametru.

```
Main> encipher 5 'C'
'H'
Main> encipher 7 'Q'
'X'
```

6. Pentru a fi criptat, textul trebuie să nu conțină semne de punctuație și să fie scris cu litere mari. Scrieți o funcție `normalize :: String -> String` care normalizează un șir, transformând literele mici în litere mari și eliminând toate caracterele care nu sunt litere sau cifre.

```
Main> normalize "July 4th!"
"JULY4TH"
```

7. Scrieți o funcție `encipherStr :: Int -> String -> String` care normalizează un șir și îl criptează folosind funcțiile definite anterior.

```
Main> encipherStr 5 "July 4th!"
"OZQD4YM"
```

Decodarea unui mesaj

8. Scrieți o funcție `reverseKey :: [(Char, Char)] -> [(Char, Char)]` pentru a inversa cheia de criptare, schimbând componentele din fiecare pereche între ele.

```
Main> reverseKey [('A', 'G'), ('B', 'H') , ('C', 'I')]
[('G', 'A'), ('H', 'B') , ('I', 'C')]
```

9. Scrieți funcțiile

```
decipher :: Int -> Char -> Char
decipherStr :: Int -> String -> String
```

pentru a decripta un caracter si un string folosind cheia generată de o deplasare dată. Funcția va lăsa nemodificate cifrele și spațiile, dar va șterge literele mici sau alte caractere.

```
Main> decipherStr 5 "OZQD4YM"
"JULY4TH"
```

Tipuri de date abstracte

Fructe

```haskell
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 și un `Int` pentru a exprima numărul de felii dintr-o portocală. De exemplu:

```haskell
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) Scrieți o funcție

```
ePortocalaDeSicilia :: Fruct -> Bool
ePortocalaDeSicilia = undefined
```

care indică dacă un fruct este o portocală de Sicilia sau nu. Soiurile de portocale din Sicilia sunt `Tarocco`, `Moro` și `Sanguinello`. De exemplu,

```
test_ePortocalaDeSicilia1 =
    ePortocalaDeSicilia (Portocala "Moro" 12) == True
test_ePortocalaDeSicilia2 =
    ePortocalaDeSicilia (Mar "Ionatan" True) == False
```

b) Scrieți o funcție

```
nrFeliiSicilia :: [Fruct] -> Int
nrFeliiSicilia = undefined

test_nrFeliiSicilia = nrFeliiSicilia listaFructe == 52
```

care calculează numărul total de felii ale portocalelor de Sicilia dintr-o listă de fructe.

c) Scrieți o funcție

```
nrMereViermi :: [Fruct] -> Int
nrMereViermi = undefined

test_nrMereViermi = nrMereViermi listaFructe == 2
```

care calcuelază numărul de mere care au viermi dintr-o lista de fructe.

Matrice

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

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

a) Scrieti o functie care verifica daca suma elementelor de pe fiecare linie este egala cu o valoare `n`. Rezolvati cerinta folosind `foldr`.

```
verifica (M[L[1,2,3], L[4,5], L[2,3,6,8], L[8,5,3]]) 10
  False

verifica (M[L[2,20,3], L[4,21], L[2,3,6,8,6], L[8,5,3,9]]) 25
  True
```

b) Scrieti o instanta a clasei `Show` pentru tipul de date `Matrice` astfel incat fiecare linie sa fie afisata pe un rand nou.

```
M[L[1,2,3], L[4,5], L[2,3,6,8], L[8,5,3]]
   1 2 3
   4 5
   2 3 6 8
   8 5 3
```

c) 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.

```
doarPozN (M [L[1,2,3], L[4,5], L[2,3,6,8], L[8,5,3]]) 3
  True
```

```
doarPozN (M [L[1,2,-3], L[4,5], L[2,3,6,8], L[8,5,3]]) 3
  False
```