

Laboratorul 5: Exerciții Fold

FOLD

Funcțiile `foldr` și `foldl` sunt folosite pentru agregarea unei colecții. Definițiile intuitive pentru `foldr` și `foldl` sunt:

```
foldr op unit [a1, a2, a3, ... , an] == a1 `op` (a2 `op` (a3 `op` .. `op` (an `op` unit))
```

```
foldr :: (a -> b -> b) -> b -> [a] -> b
foldr op i []      = i
foldr op i (x:xs) = x `op` (foldr op i xs)
```

```
foldl op unit [a1, a2, a3, ... , an] == (((unit `op` a1) `op` a2) `op` a3) `op` ..) `op`
```

```
foldl :: (b -> a -> b) -> b -> [a] -> b
foldl op i []      = i
foldl op i (x:xs) = foldl op (i `op` x) xs
```

```
ghci> foldr (+) 0 [1..5]
15
ghci> foldr (*) 1 [2,3,4]
24
ghci> foldr (++) [] ["abc","def","ghi"]
"abcdefghi"
ghci> foldl (++) "first" ["abc","def","ghi"]
"firstabcdefghi"
ghci> foldr (++) "last" ["abc","def","ghi"]
"abcdefghilast"
```

Exercitii

Rezolvați următoarele exerciții folosind `map`, `filter` și `fold` (fara recursivitate sau selectie). Pentru fiecare functie scrieti si prototipul acesteia.

1. Calculați suma pătratelor elementelor impare dintr-o listă dată ca parametru.

2. Scrieți o funcție care verifică faptul că toate elementele dintr-o listă sunt True, folosind foldr.
3. Scrieți o funcție care verifică dacă toate elementele dintr-o listă de numere întregi satisfac o proprietate dată ca parametru.

```
allVerifies :: (Int -> Bool) -> [Int] -> Bool
allVerifies = undefined
```

4. Scrieți o funcție care verifică dacă există elemente într-o listă de numere întregi care satisfac o proprietate dată ca parametru.

```
anyVerifies :: (Int -> Bool) -> [Int] -> Bool
anyVerifies = undefined
```

5. Redefiniți funcțiile map și filter folosind foldr. Le puteți numi mapFoldr și filterFoldr.
6. Folosind funcția foldl, definiți funcția listToInt care transformă o lista de cifre (un număr foarte mare stocat sub formă de listă) în numărul întreg asociat. Se presupune ca lista de intrare este dată corect.

```
listToInt :: [Integer] -> Integer
listToInt = undefined
-- listToInt [2,3,4,5] = 2345
```

7.

- (a) Scrieți o funcție care elimină un caracter din șir de caractere.

```
rmChar :: Char -> String -> String
rmChar = undefined
```

- (b) Scrieți o funcție recursivă care elimină toate caracterele din al doilea argument care se găsesc în primul argument, folosind rmChar.

```
rmCharsRec :: String -> String -> String
rmCharsRec = undefined

-- rmCharsRec ['a'..'l'] "fotbal" == "ot"
```

- (c) Scrieți o funcție echivalentă cu cea de la (b) care folosește foldr în locul recursiei și rmChar.

```
rmCharsFold :: String -> String -> String
rmCharsFold = undefined
```