Programare funcțională

Introducere în programarea funcțională folosind Haskell
C04

Ana Iova
Denisa Diaconescu

Departamentul de Informatică, FMI, UB

Procesarea fluxurilor de date: Map, Filter, Fold

Transformarea fiecărui element dintr-o listă - map

Exemplu - Pătrate

Definiți o funcție care pentru o listă de numere întregi dată ridică la pătrat fiecare element din listă.

```
Prelude> squares [1,-2,3]
[1,4,9]
```

Exemplu - Pătrate

Definiți o funcție care pentru o listă de numere întregi dată ridică la pătrat fiecare element din listă.

```
Prelude> squares [1,-2,3]
[1,4,9]
```

Soluție descriptivă

```
squares :: [Int] -> [Int]
squares xs = [ x * x | x <- xs ]
```

Exemplu - Pătrate

Definiți o funcție care pentru o listă de numere întregi dată ridică la pătrat fiecare element din listă.

```
Prelude> squares [1,-2,3]
[1,4,9]
```

Soluție descriptivă

```
squares :: [Int] -> [Int]
squares xs = [ x * x | x <- xs ]
```

Soluție recursivă

```
squares :: [Int] -> [Int]
squares []     = []
squares (x:xs) = x*x : squares xs
```

Exemplu - Coduri ASCII

Transformați un șir de caractere în lista codurilor ASCII ale caracterelor.

```
Prelude> ords "a2c3"
[97,50,99,51]
```

Exemplu - Coduri ASCII

Transformați un șir de caractere în lista codurilor ASCII ale caracterelor.

```
Prelude> ords "a2c3"
[97,50,99,51]
```

Soluție descriptivă

```
ords :: [Char] -> [Int]
ords xs = [ ord x | x <- xs ]
```

Exemplu - Coduri ASCII

Transformați un șir de caractere în lista codurilor ASCII ale caracterelor.

```
Prelude> ords "a2c3"
[97,50,99,51]
```

Soluție descriptivă

```
ords :: [Char] -> [Int]
ords xs = [ ord x | x <- xs ]
```

Soluție recursivă

```
ords :: [Char] -> [Int]
ords []     = []
ords (x:xs) = ord x : ords xs
```

Funcția map

Date fiind o funcție de transformare și o listă, aplicați funcția fiecărui element al unei liste date.

Soluție descriptivă

```
map :: (a -> b) -> [a] -> [b]
map f xs = [ f x | x <- xs ]
```

Soluție recursivă

```
map :: (a -> b) -> [a] -> [b]
map f []     = []
map f (x:xs) = f x : map f xs
```

Exemplu — Pătrate

Soluție descriptivă

```
squares :: [Int] -> [Int]
squares xs = [ x * x | x <- xs ]
```

Soluție recursivă

```
squares :: [Int] -> [Int]
squares []     = []
squares (x:xs) = x*x : squares xs
```

Soluție folosind map

```
squares :: [Int] -> [Int]
squares xs = map sqr xs
  where sqr x = x * x
```

Exemplu — Coduri ASCII

Soluție descriptivă

```
ords :: [Char] -> [Int]
ords xs = [ ord x | x <- xs ]
```

Soluție recursivă

```
ords :: [Char] -> [Int]
ords []     = []
ords (x:xs) = ord x : ords xs
```

Soluție folosind map

```
ords :: [Char] -> [Int]
ords xs = map ord xs
```

Quiz time!

Seria 23: https://www.questionpro.com/t/AT4qgZpb7W

Seria 24: https://www.questionpro.com/t/AT4NiZpbVh

Seria 25: https://www.questionpro.com/t/AT4qgZpb7g

Selectarea elementelor dintr-o listă - filter

Exemplu - Selectarea elementelor pozitive dintr-o listă

```
Prelude> positives [1,-2,3]
[1,3]
```

Soluție descriptivă

```
positives :: [Int] -> [Int]
positives xs = [ x | x <- xs, x > 0 ]
```

Soluție recursivă

```
positives :: [Int] -> [Int]
positives []                  = []
positives (x:xs) | x > 0      = x : positives xs
                 | otherwise  = positives xs
```

Exemplu - Selectarea cifrelor dintr-un șir de caractere

```
Prelude> digits "a2c3"
"23"
```

Soluție descriptivă

```
digits :: [Char] -> [Char]
digits xs = [ x | x <- xs, isDigit x ]
```

Soluție recursivă

```
digits :: [Char] -> [Char]
digits []                   = []
digits (x:xs) | isDigit x = x : digits xs
              | otherwise = digits xs
```

Funcția filter

Date fiind un predicat (funcție booleană) și o listă, selectați elementele din listă care satisfac predicatul.

Soluție descriptivă

```
filter :: (a -> Bool) -> [a] -> [a]
filter p xs = [ x | x <- xs, p x ]
```

Soluție recursivă

```
filter :: (a -> Bool) -> [a] -> [a]
filter p []                    = []
filter p (x:xs) | p x          = x : filter p xs
                | otherwise = filter p xs
```

Exemplu — Selectarea elementelor pozitive dintr-o listă

Soluție descriptivă

```
positives :: [Int] -> [Int]
positives xs = [ x | x <- xs, x > 0 ]
```

Soluție recursivă

```
positives :: [Int] -> [Int]
positives []                  = []
positives (x:xs) | x > 0      = x : positives xs
                 | otherwise  = positives xs
```

Soluție folosind filter

```
positives :: [Int] -> [Int]
positives xs = filter pos xs
  where pos x = x > 0
```

Exemplu — Selectarea cifrelor dintr-un șir de caractere

Soluție descriptivă

```
digits :: [Char] -> [Char]
digits xs = [ x | x <- xs, isDigit x ]
```

Soluție recursivă

```
digits :: [Char] -> [Char]
digits []                   = []
digits (x:xs) | isDigit x = x : digits xs
              | otherwise = digits xs
```

Soluție folosind filter

```
digits :: [Char] -> [Char]
digits xs = filter isDigit xs
```

Quiz time!

Seria 23: https://www.questionpro.com/t/AT4qgZpb7o

Seria 24: https://www.questionpro.com/t/AT4NiZpbYV

Seria 25: https://www.questionpro.com/t/AT4qgZpb8T

Agregarea elementelor dintr-o listă - fold

Exemplu - Suma

Definiți o funcție care dată fiind o listă de numere întregi calculează suma elementelor din listă.

```
Prelude> sum [1,2,3,4]
10
```

Exemplu - Suma

Definiți o funcție care dată fiind o listă de numere întregi calculează suma elementelor din listă.

```
Prelude> sum [1,2,3,4]
10
```

Soluție recursivă

```
sum :: [Int] -> Int
sum []     = 0
sum (x:xs) = x + sum xs
```

Exemplu - Produs

Definiți o funcție care dată fiind o listă de numere întregi calculează produsul elementelor din listă.

```
Prelude> product [1,2,3,4]
24
```

Exemplu - Produs

Definiți o funcție care dată fiind o listă de numere întregi calculează produsul elementelor din listă.

```
Prelude> product [1,2,3,4]
24
```

Soluție recursivă

```
product :: [Int] -> Int
product []     = 1
product (x:xs) = x * sum xs
```

Exemplu - Concatenare

Definiți o funcție care concatenează o listă de liste.

```
Prelude> concat [[1,2,3],[4,5]]
[1,2,3,4,5]

Prelude> concat ["con","ca","te","na","re"]
"concatenare"
```

Exemplu - Concatenare

Definiți o funcție care concatenează o listă de liste.

```
Prelude> concat [[1,2,3],[4,5]]
[1,2,3,4,5]

Prelude> concat ["con","ca","te","na","re"]
"concatenare"
```

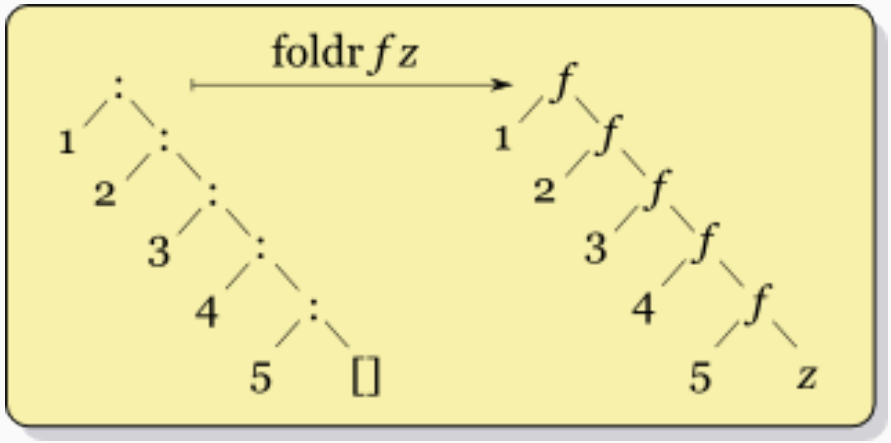
Soluție recursivă

```
concat :: [[a]] -> [a]
concat []       = []
concat (xs:xss) = xs ++ concat xss
```

Funcția foldr

foldr :: (a –> b –> b) –> b –> [a] –> b

Date fiind o funcție de actualizare a valorii calculate cu un element curent, o valoare inițială, și o listă, calculați valoarea obținută prin aplicarea repetată a funcției de actualizare fiecărui element din listă.



Funcția foldr

```
foldr :: (a -> b -> b) -> b -> [a] -> b
```

Soluție recursivă

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

Soluție recursivă cu operator infix

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

Exemplu — Suma

Soluție recursivă

```
sum :: [Int] -> Int
sum []     = 0
sum (x:xs) = x + sum xs
```

Soluție folosind foldr

```
sum :: [Int] -> Int
sum xs = foldr (+) 0 xs
```

Exemplu

```
foldr (+) 0 [1, 2, 3] == 1 + (2 + (3 + 0))
```

Exemplu — Produs

Soluție recursivă

```
product :: [Int] -> Int
product []     = 1
product (x:xs) = x * sum xs
```

Soluție folosind foldr

```
product :: [Int] -> Int
product xs = foldr (*) 1 xs
```

Exemplu

```
foldr (*) 1 [1, 2, 3] == 1 * (2 * (3 * 1))
```

Exemplu — Concatenare

Soluție recursivă

```
concat :: [[a]] -> [a]
concat []       = []
concat (xs:xss) = xs ++ concat xss
```

Soluție folosind foldr

```
concat :: [Int] -> Int
concat xs = foldr (++) [] xs
```

Exemplu

```
foldr (++) [] ["Ana ", "are ", "mere."]
  == "Ana " ++ ("are " ++ ("mere." ++ []))
```

Quiz time!

Seria 23: https://www.questionpro.com/t/AT4qgZpb8Y

Seria 24: https://www.questionpro.com/t/AT4NiZpbcj

Seria 25: https://www.questionpro.com/t/AT4qgZpb8i

Map, Filter, Fold — combinate

Exemplu – Suma pătratelor numerelor pozitive

```
f :: [Int] -> Int
f xs = sum (squares (positives xs))
```

Exemplu – Suma pătratelor numerelor pozitive

```
f :: [Int] -> Int
f xs = sum (squares (positives xs))

f :: [Int] -> Int
f xs = sum [ x*x | x <- xs, x > 0 ]
```

Exemplu – Suma pătratelor numerelor pozitive

```
f :: [Int] -> Int
f xs = sum (squares (positives xs))

f :: [Int] -> Int
f xs = sum [ x*x | x <- xs, x > 0 ]

f :: [Int] -> Int
f []                  = 0
f (x:xs)| x > 0       = (x*x) + f xs
        | otherwise = f xs
```

Exemplu – Suma pătratelor numerelor pozitive

```
f :: [Int] -> Int
f xs = sum (squares (positives xs))

f :: [Int] -> Int
f xs = sum [ x*x | x <- xs, x > 0 ]

f :: [Int] -> Int
f []                  = 0
f (x:xs)| x > 0       = (x*x) + f xs
        | otherwise = f xs

f :: [Int] -> Int
f xs = foldr (+) 0 (map sqr (filter pos xs))
  where
    sqr x = x * x
    pos x = x > 0
```

Foldr cu secțiuni — Exemplu

```
f :: [Int] -> Int
f xs = foldr (+) 0 (map sqr (filter pos xs))
  where
    sqr x = x * x
    pos x = x > 0
```

Folosind λ-expresii

```
f :: [Int] -> Int
f xs = foldr (+) 0
         (map (\ x -> x * x)
           (filter (\ x -> x > 0) xs))
```

Foldr cu secțiuni — Exemplu

```
f :: [Int] -> Int
f xs = foldr (+) 0 (map sqr (filter pos xs))
  where
    sqr x = x * x
    pos x = x > 0
```

Folosind λ-expresii

```
f :: [Int] -> Int
f xs = foldr (+) 0
         (map (\ x -> x * x)
           (filter (\ x -> x > 0) xs))
```

Folosind secțiuni

```
f :: [Int] -> Int
f xs = foldr (+) 0 (map (^2) (filter (>0) xs))
```

Operatorul . — stilul funcțional

Definiție cu parametru explicit

```
f :: [Int] -> Int
f xs = foldr (+) 0 (map (^2) (filter (>0) xs))
```

Definiție compozițională

```
f :: [Int] -> Int
f = foldr (+) 0 . map (^2) . filter (>0)
```

Map/Filter/Fold în alte limbaje

Map/Filter/Reduce în Haskell

Exemplu. Aflați lungimea celui mai lung cuvânt care începe cu litera ’c’ dintr-o listă dată.

Map/Filter/Reduce în Haskell

Exemplu. Aflați lungimea celui mai lung cuvânt care începe cu litera 'c' dintr-o listă dată.

```
strs = ["cezara", "petru", "claudia", "", "virgil"];
maxLengthFn = foldr max 0 .
              map length .
              filter testC
  where testC ('c':_) = True
        testC _       = False
maxLength = maxLengthFn strs
```

Map/Filter/Reduce în Python

http://www.python-course.eu/lambda.php

```
strs = ["cezara", "petru", "claudia", "", "virgil"];
def maxLengthFn(strs):
  return reduce(max,
           map(len,
              filter(lambda s: len(s) > 0 and s[0] == 'c',
                strs)));
maxLength = maxLengthFn(strs);
print(maxLength);
```

Map/Filter/Reduce în Javascript

http://cryto.net/~joepie91/blog/2015/05/04/functional-programming-in-javascript-map-filter-reduce/

```
var strs = ["cezara", "petru", "claudia", "", "virgil"];
var maxLength = strs
                .filter(function(s){ return s[0]=='c'; })
                .map(function(s){ return s.length; })
                .reduce(function(a,b){ return Math.max(a,b);
```

Map/Filter/Reduce în PHP

http://eddmann.com/posts/mapping-filtering-and-reducing-in-php/

```
$strs = array("cezara", "petru", "claudia", "", "virgil");
$max_length = array_reduce(
  array_map(
    "strlen",
    array_filter(
      $strs,
      function($s){return isset($s[0]) && $s[0]=='c';})),
  "max",
  0);
echo $max_length;
```

Map/Filter/Reduce în Java 8

http://winterbe.com/posts/2014/07/31/java8-stream-tutorial-examples/

```
package edu.unibuc.fmi;
import java.util.*;

public class Main {
  public static void main(String[] args) {
    List<String> myList = Arrays.asList(
        "cezara", "petru", "claudia", "", "virgil");
    int l =
        myList
            .stream()
            .filter(s -> s.startsWith("c"))
            .map(String::length)
            .reduce(0, Integer::max);
        System.out.println(l);
    }
}
```

Map/Filter/Reduce în C++11

https://meetingcpp.com/tl_files/mcpp/slides/12/FunctionalProgrammingInC++11.pdf

```
#include <algorithm>
#include <string>
#include <iostream>
using namespace std;
int main() {
  vector<string>strs {"cezara","petru","claudia","","virgil"};
  strs.erase(remove_if(strs.begin(), strs.end(),
                  [](string x){return x[0]!='c';}),
        strs.end());
  vector<int>lengths;
  transform(strs.begin(), strs.end(), back_inserter(lengths),
            [](string x) { return x.length();});
  int max_length = accumulate(lengths.begin(), lengths.end(),
                    0,  [](int a,int b){ return a>b?a:b; });
  cout << max_length;
    }
```

Pe săptămâna viitoare!