

# Programare declarativă

## Introducere în programarea funcțională folosind Haskell

Ioana Leuștean  
Traian Florin Șerbănuță

Departamentul de Informatică, FMI, UB  
ioana.leustean@unibuc.ro  
traian.serbanuta@unibuc.ro

# Plan curs

- Partea I: Programare funcțională (folosind Haskell)
  - Funcții, recursie, funcții de ordin înalt, tipuri
  - Operații pe liste: filtrare, transformare, agregare
  - Polimorfism, clase de tipuri, modularizare
  - Tipuri de date algebrice - evaluarea expresiilor
  - Operațiuni Intrare/Ieșire
  - Agregare pe tipuri algebrice
  - Functori, clasa Applicative, monade
- Partea II: Introducere în teoria tipurilor
  - $\lambda$ -calcul, tipuri,  $\lambda\mu$ -calcul
  - Sisteme formale de verificarea tipurilor
  - Detectarea automată a tipurilor

# Resurse

- Paginile cursului:
  - <http://moodle.fmi.unibuc.ro/course/view.php?id=449> (Seria 33)
  - <http://moodle.fmi.unibuc.ro/course/view.php?id=367> (Seria 34)
  - <http://unibuc.ro/~ileustean/PD.html>
  - Prezentările cursurilor, forumuri, resurse electronice
  - Știri legate de curs vor fi postate pe ambele pagini și pe moodle
- <http://bit.do/progdecl>
  - Cele mai noi variante ale cursurilor si laboratoarelor.
- Canal de discuții dedicat pe Slack
  - Asistență în probleme de Haskell și nu numai
- Cartea online „Learn You a Haskell for Great Good”  
<http://learnyouahaskell.com/>
- Pagina Haskell <http://haskell.org>
  - Hoogle <https://www.haskell.org/hoogle>
  - Haskell Wiki <http://wiki.haskell.org>

# Evaluare

## Notare

- Testare laborator (lab), examen (ex)
- Nota finală: 1 (oficiu) + lab + ex

## Condiție de promovabilitate

- Nota finală **cel puțin 5**
  - $5 > 4.99$

## Activitate laborator

- La sugestia profesorului coordonator al laboratorului, se poate nota activitatea în plus față de cerințele obșnuite.
- Maxim 1 punct (bonus la nota finală)

# Test laborator

- Valorează 7 puncte din nota finală
- După săptămâna a 10-a
- Pe calculatoare
- Durată: 2 ore
- Acoperă materia din Partea I
- Cu acces la materiale descărcate pe calculator
- Fără acces la rețea/internet

**Atenție!**

**În restanță această probă se dă pe hârtie!**

# Examen final

- Valorează 2 puncte din nota finală
- În sesiune
- Pe hârtie
- Acoperă materia din Partea II
- Durată: 2 ore
- Cu acces la materiale tipărite
- Fără acces la rețea/internet

# Programare declarativă vs. imperativă

Ce vs. cum

## Programare imperativă (Cum)

Explic mașinii, pas cu pas, algoritmic, **cum** să facă ceva și se întâmplă **ce** voiam să se întâmple ca rezultat al execuției mașinii.

- limbaje procedurale
- limbaje de programare orientate pe obiecte

## Programare declarativă (Ce)

Îi spun mașinii **ce** vreau să se întâmple și o las pe ea să se descurce **cum** să realizeze acest lucru. :-)

- limbaje de programare logică
- limbaje de interogare a bazelor de date
- limbaje de programare funcțională

# Programare imperativă vs. declarativă

## Diferențe

- Modelul de computație: **algoritm** vs. **relație**
- Ce exprimă un program: **cum** vs. **ce**
- Variabile/parametrii: atribuire **distructivă** vs. **non-distructivă**
- Structuri de date: **alterabile** vs. **explicite**
- Ordinea de execuție: **efecte laterale** vs. **neimportantă**
- Expresii ca valori: **nu** vs. **da**
- Controlul execuției: responsabilitatea **programatorului** vs **a mașinii**



# Agregarea datelor dintr-o colecție (JS)

C. Boesch, Declarative vs Imperative Programming - Talk.JS

<https://www.youtube.com/watch?v=M2e5sq1rnvc>



Engineers.SG  
Meetups Videos



Will I ever prefer to read declarative javascript?

```
function multiply(array) {  
  return array.reduce( (a,b) => a*b, 1);  
}
```

```
function multiply(array) {  
  var total = 1;  
  for (var i = 0; i < array.length; i++){  
    total = total * array[i];  
  }  
  return total;  
}
```

# Agregarea datelor dintr-o colecție (JS)

C. Boesch, Declarative vs Imperative Programming - Talk.JS

<https://www.youtube.com/watch?v=M2e5sq1rnvc>



Engineers.SG  
Meetups Videos

## Reasons to be More Declarative

- Better readability
- Better scalability
- Fewer state-related bugs
- Stand on the shoulders of giants



# Programare funcțională în Haskell

- Programarea funcțională e din ce în ce mai importantă în industrie
  - Haskell e folosit în proiecte de Facebook, Google, Microsoft, ...
  - mai multe la [https://wiki.haskell.org/Haskell\\_in\\_industry](https://wiki.haskell.org/Haskell_in_industry)
- Programare funcțională în limbajul vostru preferat de programare:
  - Java 8, C++11, C#, Python, PHP, JavaScript, Ruby, ...
  - Funcții anonime ( $\lambda$ -abstracții)
  - Funcții de procesare a fluxurilor de date: filter, map, reduce



## De ce Haskell? (din cartea Real World Haskell)

The illustration on our cover is of a **Hercules beetle**. These beetles are among the largest in the world. They are also, in proportion to their size, the strongest animals on Earth, able to lift up to 850 times their own weight. Needless to say, we like the association with a creature that has such a high power-to-weight ratio. [It] is a deep language and [...] learning it is a hugely rewarding experience.

# Nou

## Programarea funcțională

O cale profund diferită de a concepe ideea de software

- În loc să modificăm datele existente, calculăm valori noi din valorile existente, folosind funcții
- Funcțiile sunt **pure**: aceleași rezultate pentru aceleași intrări
- Distincție clară între părțile pure și cele care comunică cu mediul extern
- Haskell e leneș: orice calcul e amânat cât de mult posibil
  - Schimbă modul de concepere al programelor
  - Permite lucrul cu colecții potențial infinite de date precum [1..]

# Puternic

- Puritatea asigură consistență
  - O bucată de cod nu poate corupe datele altei bucăți de cod.
  - Mai ușor de testat decât codul care interacționează cu mediul
- Evaluarea leneșă poate fi exploatată pentru a reduce timpul de calcul fără a denatura codul

```
firstK k xs = take k (sort xs)
```

- Minimalism: mai puțin cod, în mai puțin timp, și cu mai puține defecte
  - ...rezolvând totuși problema :-)

```
numbers = [1,2,3,4,5]
total = foldl (*) 0 numbers
doubled = map (* 2) numbers
```

- Oferă suport pentru paralelism și concurență

# Elegant

- Idei abstracte din matematică devin instrumente puternice practice
  - recursivitate, compunerea de funcții, functori, monade
  - folosirea lor permite scrierea de cod compact și modular
- Rigurozitate: ne forțează să gândim mai mult înainte, dar ne ajută să scriem cod mai corect și mai curat
- Curbă de învățare în trepte
  - Putem scrie programe mici destul de repede
  - Expertiza în Haskell necesită multă gândire și practică
  - Descoperirea unei lumi noi poate fi un drum distractiv și provocator  
<http://wiki.haskell.org/Humor>

- În 1929-1932 Church a propus  $\lambda$ -calculul ca sistem formal pentru logica matematică. În 1935 a argumentat că orice funcție calculabilă peste numere naturale poate fi calculată în  $\lambda$ -calcul.
- În 1935, independent de Church, Turing a dezvoltat mecanismul de calcul numit astăzi Mașina Turing. În 1936 și el a argumentat că orice funcție calculabilă peste numere naturale poate fi calculată de o mașină Turing. De asemenea, a arătat echivalența celor două modele de calcul. Această echivalență a constituit o indicație puternică asupra "universalității" celor două modele, conducând la ceea ce numim astăzi "Teza Church-Turing".

# Lambda Calcul / Funcții anonime

## Lambda Calcul - sintaxă

$$\begin{array}{ll} t = & x \quad \text{(variabilă)} \\ & | \lambda x. t \quad \text{(abstractizare)} \\ & | t t \quad \text{(aplicare)} \end{array}$$

Exemple:

$\lambda x. x * x$

$\lambda x. x > 0$

## Funcții anonime în Haskell

În Haskell,  $\backslash$  e folosit în locul simbolului  $\lambda$  și  $\rightarrow$  în locul punctului.

$\lambda x. x * x$  este  $\backslash x \rightarrow x * x$

$\lambda x. x > 0$  este  $\backslash x \rightarrow x > 0$



# Funcții anonime în Haskell

```
Prelude> (\x -> x+1) 3
```

4

```
Prelude> (\ x y -> x + y) 4 5
```

9

```
Prelude> (\ f x y -> f x y) (*) 10 3
```

30

**Enjoy!**

