# SYLLABUS

## 1. Data about the program of study

| | |
|---|---|
| 1.1 Institution | The Technical University of Cluj-Napoca |
| 1.2 Faculty | Faculty of Automation and Computer Science |
| 1.3 Department | Computer Science |
| 1.4 Field of study | Computer Science and Information Technology |
| 1.5 Cycle of study | Bachelor of Science |
| 1.6 Program of study / Qualification | Computer science / Engineer |
| 1.7 Form of education | Full time |
| 1.8 Subject code | 32. |

## 2. Data about the subject

| | | | | | |
|---|---|---|---|---|---|
| 2.1 Subject name | ***Functional programming*** | | | | |
| 2.2 Course responsible / lecturer | Assoc. prof. dr. eng. Radu Răzvan Slăvescu - Radu.Razvan.Slavescu@cs.utcluj.ro | | | | |
| 2.3 Teachers in charge of seminars / Laboratory / project | Prof. dr. eng. Camelia Pintea - Camelia.Pintea@mi.utcluj.ro<br>Eng. Istvan Csaszar | | | | |
| 2.4 Year of study | III | 2.5 Semester | 5 | 2.6 Type of assessment (E - exam, C - colloquium, V - verification) | E |
| 2.7 Subject category | *DF – fundamentală, DD – în domeniu, DS – de specialitate, DC – complementară* | | | | DD |
| | *DI – Impusă, DOp – opțională, DFac – facultativă* | | | | DI |

## 3. Estimated total time

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 3.1 Number of hours per week | 4 | of which: | Course | 2 | Seminars | | Laboratory | 2 | Project | |
| 3.2 Number of hours per semester | 56 | of which: | Course | 28 | Seminars | | Laboratory | 28 | Project | |

| | |
|---|---|
| 3.3 Time budget (hours/semester) for study: | |
| (a) Manual, lecture material and notes, bibliography | 18 |
| (b) Supplementary study in the library, online and in the field | 10 |
| (c) Preparation for seminars/laboratory works, homework, reports, portfolios, essays | 10 |
| (d) Tutoring | 4 |
| (e) Exams and tests | 2 |
| (f) Other activities: | |
| 3.4 Total hours of individual study (suma (3.3(a)...3.3(f))) | 44 |
| 3.5 Total hours per semester (3.2+3.4) | 100 |
| 3.6 Number of credit points | 4 |

## 4. Pre-requisites (where appropriate)

| | |
|---|---|
| 4.1 Curriculum | Data Structures and Algorithms Course |
| 4.2 Competence | This course assumes no prior knowledge of functional programming, but it is advised to have at least one year of programming experience in a regular programming language such as Java, C, C++. |

## 5. Requirements (where appropriate)

| | |
|---|---|
| 5.1. For the course | Whiteboard, beamer, computer |
| 5.2. For the applications | Computers, interpreters/compilers for the studied languages<br>100% lab presence for final exam access |

## 6. Specific competence

| | |
|---|---|
| 6.1 Professional competences | **C2** Designing a software system in a functional manner<br>**C2.1** Identifying and describing the software components of the system |

| | |
|---|---|
| | **C2.2** Explaining the role, interaction and functioning of each component<br>**C2.3** Building software components of some computing systems using design methods, languages, technologies and tools specific to Functional Programming<br>**C2.4** Implementing the software components in functional style, in an idiomatic and efficient manner<br>**C2.5** Evaluating the functional and non-functional characteristics of the computing system using specific performance metrics and proving its corectness |
| 6.2 Cross competences | N/A |

**7. Discipline objective (as results from the *key competences gained*)**

| | |
|---|---|
| 7.1 General objective | Increasing the ability to develop correct and more concise code via the functional paradigm elements (immutability, high level of abstractization, formal proof of code correctness, easy code parellelization) and to understand its underpinning formalism (lambda calculus) |
| 7.2 Specific objectives | Writing better code with the concepts introduced by functional programming:<br>- to write code in a functiional manner, with no state variables<br>- to see the advantages and disadvantages of different programming styles<br>- to use recursivity and its optimization<br>- to use high order functions<br>- to exploit lazy evaluation mechanisms and infinte structures<br>- to build formal proofs of program corectness<br>- to manipulate basic lambda expressions |

**8. Contents**

| 8.1 Lectures | Hours | Teaching methods | Notes |
|---|---|---|---|
| Introduction. Programming Paradigms. Basic concepts of programming in Haskell, Elm: functions, identifiers, recursion. | 2 | (Onsite Slides, Demos on the whiteboard, New examples Quick individual work (1 minute) | |
| Basic concepts: recursion, constants, primitive data types, tuples, infix operators, evaluation. | 2 | | |
| Basic concepts: local declarations, data types, polymorphism. | 2 | | |
| Lists: list construction, basic operations on lists. | 2 | | |
| Lists: list operators (generators, guards, list comprehensions). | 2 | | |
| Trees: alternative data, pattern matching, exceptions, binary trees, list-tree conversions. | 2 | | |
| Trees: binary search trees, checking AVL balance property for trees, printing. | 2 | | |
| Implementing operations on sets. Propositional reasoner | 2 | | |
| Higher-order functions: anonymous functions, partial application, relations functions – data, combinator functions | 2 | | |
| Higher-order functions for lists (map, filter, fold). | 2 | | |
| Infinite data: lazy evaluation, unbounded objects, circular structures. | 2 | | |
| Lambda calculus: Lambda notation, conversions, combinators. | 2 | | |
| Reasoning on program correctness: structural induction, equivalence of functions, induction on the number of nodes. | 2 | | |
| Monads. Example of use cases. | 2 | | |

Bibliography


1. Haskell - A Purely Functional Language, www.haskell.org
2. Elm – A Delightful language for reliable web applications, elm-lang.org
3. G. Hutton. **Programming in Haskell, 2nd edition** Cambridge University Press, 2016
4. M. Lipovaca. **Learn You a Haskell for Great Good.** No Starch Press, 2011.
5. Raul Rojas, A Tutorial Introduction to the Lambda Calculus, FU Berlin, 2015


| 8.2 Applications – Seminars/Laboratory/Project | Hours | Teaching methods | Notes |
|---|---|---|---|

| | | | |
|---|---|---|---|
| Introduction in Functional Programming using Elm | 2 | (Onsite) Exercises and problem solving, implementing functions on the computer, Tracing algorithms Miniprojects | |
| Elm Types | 2 | | |
| Lists and Recursivity | 2 | | |
| Higher order Functions in Elm | 2 | | |
| Evaluation Elm | 2 | | |
| Miniapplication in Elm | 2 | | |
| Introduction in Haskell. Lists, Recursion | 2 | | |
| Haskell Type checking | 2 | | |
| Trees in Haskell | 2 | | |
| Haskell – High order functions | 2 | | |
| Haskell - Lazy evaluation, infinite lists. | 2 | | |
| Miniapplication in Haskell | 2 | | |
| Lambda Calculus | 2 | | |
| Evaluation Haskell | 2 | | |

Bibliography


1. www.haskell.org
2. elm-lang.org
3. M. Lipovaca. **Learn You a Haskell for Great Good.** No Starch Press, 2011.


**Se vor preciza, după caz: tematica seminariilor, lucrările de laborator, tematica și etapele proiectului.*

**9. Bridging course contents with the expectations of the representatives of the community, professional associations and employers in the field**

The content of the class is similar to the contents taught at other international universities (Programming Paradigms, Parallel and Concurrent Haskell). The course is focused on the techniques of Functional Programming which have been adopted by the modern (multiparadigm) languages and also on the possibility of proving program correctness in a formal manner. Students are encouraged to identify Functional Programming ideas in the current practice of local IT companies.

**10. Evaluation**

| Activity type | Assessment criteria | Assessment methods | Weight in the final grade |
|---|---|---|---|
| Course | Understanding functional programming elements and its theoretical background. Class participation, Homework | Written exam/Moodle test | 50% |
| Seminar | - | | |
| Laboratory | Quantity and quality of code in Elm, Haskell<br>Ability to find and fix code bugs | Individual tests and mini-applications | 50% |
| Project | - | | |

Minimum standard of performance:
Understanding and code writing for the following concepts; Recursion, High Order Functions, Pattern Matching.
Grade calculus: 50% laboratory + 50% final exam
Conditions for participating in the final exam: Laboratory Mark Average ≥ 5
Conditions for promotion: Exam Mark Average ≥ 5

| **Date of filling in:** | **Teachers** | **Title First name Last name** | **Signature** |
|---|---|---|---|
| 19.06.2023 | Course | Assoc. prof. dr. eng. Radu Răzvan Slăvescu | |
| | Applications | Prof. dr. eng. Camelia Pintea | |
| | | Eng. Istvan Csaszar | |

| | |
|---|---|
| **Date of approval in the department** | Head of department,<br>Prof. dr. eng. Rodica Potolea |
| **Date of approval in the Faculty Council** | Dean,<br>Prof. dr. eng. Liviu Miclea |