Course description for
Higher-Order Functional Programming

Lecturer

Dr. Péter Hanák, senior lecturer, Dept. of Control Engineering and Information Technology, Faculty of Electrical Engineering and Informatics

Synopsis & Objectives

Short summary of the knowledge transmitted by the course and outlining the skills students will obtain by completing the course.

After the widely used imperative programming paradigm the course makes the students acquainted with the declarative (or more specifically, functional) programming paradigm. Functional programming is characterised, on the one hand, by the extensive use of procedural and data recursion, polymorphism, and the existence of higher-order functions, while on the other by the absence of refreshable variables, assignments, side-effects and loops. Throughout the course several functional languages will be used (Standard ML, Alice and Haskell are planned) each owning interesting features not found in the other ones.

Students will learn how to formulate programming problems recursively. This will improve their abstraction capability, and contribute to their understanding of program correctness and reasoning about correctness. They will be able to compare the different programming paradigms and languages, and weigh their advantages against their disadvantages.

Prerequisites

Basic programming skills and working knowledge of at least one programming language.

Attendance

Attending the course is obligatory. Any student being absent more than 30% of the classes cannot complete the course. Attendance will be checked on a regular basis throughout the course.

Special requirements

None.

Subject outline - Main topics

Break-down of course material to an hour-by-hour topic description.

1. week: Introduction to functional programming and Standard ML. Values, names, types, expressions. λ-functions, named functions, declarations.
2. week: Procedural abstraction. Strict and lazy evaluation. Conditional expressions. Lazy operators.
3. week: Top-down program development. Example: square root by Newton's method. Linear recursion and iteration.
4. week: Polymorphism. Curried functions. Lists. Basic operations and functions on lists.
5. week: More operations and functions on lists (map, filter, take, drop, etc.). Records and tuples. Expression with local declarations.
6. week: Data abstraction (type rat). Unit. Sequential expressions. foldr and foldl. More examples for list processing.
7. week: Tree recursion. The datatype declaration. Case. Optional values. Exceptions. List sorting.
8. week: More recursive functions. Trees, tree traversal. Backtracking.
9. week: Syntactic enhancements over SML, lazy evaluation, futures in Alice.
10. week: Modules, packages, pickling, components, distributed programming in Alice.
11. week: Constrained programming in Alice.
12. week: Types, values, functions, operators, data constructors, pattern, guards, lazy expressions, infinite lists in Haskell.
13. week: Type classes, overloading, inheritance in Haskell.
14. week: "Imperative" elements: monads and do-notation in Haskell.

Literature

T*extbooks, internet websites ....etc.*

The website of the course is at <http://dp.iit.bme.hu/hfp>.

Slides used during classes, links to on-line literature and references to textbooks or papers will be announced on the website.

Assessment

Throughout the course small- and medium-sized homeworks will regularly be given to the students. At least 50% of these homeworks must be solved by each student independently from others.

At the end of the course the examination consists of two parts: first, each student has to solve a medium-sized programming task using a computer, and then answer some theoretical question related to functional programming.