UNIVERSITY OF GOTHENBURG

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DIT260 Advanced Functional Programming, 7.5 credits

Avancerad funktionell programmering, 7,5 högskolepoäng

Second Cycle

Confirmation

This course syllabus was confirmed by The IT Faculty Board on 2009-09-19 and was last revised on 2017-12-19 by Department of Computer Science and Engineering to be valid from 2018-08-19, autumn semester of 2018.

Field of education: Science 100%
Department: Department of Computer Science and Engineering

Position in the educational system

The course is offered within several programmes. It is also a single subject course at the University of Gothenburg.

The course can be part of the following programmes: 1) Computer Science, Master's Programme (N2COS), 2) Applied Data Science Master's Programme (N2ADS) and 3) Computer Science, Bachelor´s Programme (N1COS)

Main field of studies	*Specialization*
Computer Science	A1F, Second cycle, has second-cycle course/s as entry requirements

Entry requirements

To be eligible to the course, the student should have successfully completed 120 credits of studies in computer science or equivalent. Specifically, the following courses are required, or equivalent:

- DIT143 Functional Programming, 7.5 credits, or DIT440 Introduction to Functional Programming, 7.5 credits,
- DIT980 Discrete Mathematics for Computer Scientists, 7.5 credits,
- DIT231 Programming Language Technology, 7.5 credits.

Applicants must prove knowledge of English: English 6/English B or the equivalent level of an internationally recognized test, for example TOEFL, IELTS.

Learning outcomes

After completion of the course the student should be able to:

Knowledge and understanding

- explain advanced type system features, such as type classes, generalized algebraic datatypes, functors, monads and monad transformers, and relate them to each other

Competence and skills

- design embedded domain specific languages (EDSLs); explain and exemplify their abstract and concrete syntax and semantics; and implement them in Haskell as combinator libraries
- use specification-based development techniques to formulate and test properties about programs
- reason about the correctness of functional programs, and transform them on the basis of such reasoning
- analyse and extend Haskell programs which use advanced type system features

Judgement and approach

- discuss the above topics (i.e., type system features, EDSLs, specification-based techniques and correctness), and how they relate to each other

Course content

The aim of the course is to explore the powerful mechanisms that functional programming languages offer to solve real problems and structure larger programs. The focus lies on library design and the concept of embedded languages.

The big advantage with functional languages is that language constructions can be given names and thereby reused, using higher order functions. Functional programs can therefore often be constructed by composing constructions from a library. This method enables a way to construct programs quickly and with a high degree of correctness. This is the central idea in this course.

We can learn a lot from studying the standard library of list functions such as map, fold and so on. These functions can be generalised to operate on other datatypes.

Realistic functional programs must also handle changes in state, exceptions, backtracking and other "non-functional" behaviours. We will look at how these can be modelled in a purely functional manner. The concept of "monads" will help us here.

Armed with this knowledge we will construct domain specific libraries, designed to construct programs in a certain application domain. This type of library can be said to define a domain specific language, since the constructions the programmer uses to construct larger programs mainly consists of library functions. We will study libraries for parsing, pretty printing, graphics, pseudo-parallel programming and interaction. The course will also present some recent research which can make the contents of the course vary to some degree. The programming language used in the course is Haskell.

Sub-courses

1. **Written examination** *(Skriftlig tentamen),* 3 higher education credits
   Grading scale: Pass with Distinction (VG), Pass (G) and Fail (U)
2. **Laboratory work** *(Laboration),* 4.5 higher education credits
   Grading scale: Pass with Distinction (VG), Pass (G) and Fail (U)

Form of teaching

Lectures, laborations, supervision and self-studies. The students are expected to do a lot of independent programming and self-study.

Language of instruction: English

Assessment

The course is examined by 2-3 programming laborations (U-VG) normally done in pairs during the course, and an individual exam (U-VG) given in an examination hall at the end.

If a student, who has failed the same examined component twice, wishes to change examiner before the next examination, a written application shall be sent to the department responsible for the course and shall be granted unless there are special reasons to the contrary (Chapter 6, Section 22 of Higher Education Ordinance).

In cases where a course has been discontinued or has undergone major changes, the student shall normally be guaranteed at least three examination occasions (including the ordinary examination) during a period of at least one year from the last time the course was given.

Grades

The grading scale comprises: Pass with Distinction (VG), Pass (G) and Fail (U).
A Pass grade (G) for the entire course requires at least a Pass grade for all sub-courses.

The final grade of the full course is based 60% on the result of the laborations and 40% on the result of the written examination.

Course evaluation

The course is evaluated through meeting after the course between teachers and student representatives. Further, an anonymous questionnaire is used to ensure written information. The outcome of the evaluations serves to improve the course by indicating which parts could be added, improved, changed or removed.

Additional information

The course is a joint course together with Chalmers.

Course literature to be announced the latest 8 weeks prior to the start of the course.

It is recommended, but not required, to read the following courses beforehand: DIT602 Algorithms, and one of DIT201 Logic in Computer Science or DIT321 Finite Automata Theory and Formal Languages.