



## DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

### **DIT391 Principles of Concurrent Programming, 7.5 credits**

Principer för parallell programmering, 7,5 högskolepoäng

*First Cycle*

---

#### **Confirmation**

This course syllabus was confirmed by Department of Computer Science and Engineering on 2017-02-13 and was last revised on 2018-03-26 to be valid from 2018-08-20, autumn semester of 2018.

*Field of education:* Science 100%

*Department:* Department of Computer Science and Engineering

#### **Position in the educational system**

The course is a compulsory course in the Computer Science, Bachelor's Programme. The course is also a single subject course at the University of Gothenburg.

The course can be part of the following programmes: 1) Mathematical Sciences, Master's Programme (N2MAT), 2) Computer Science, Master's Programme (N2COS), 3) Applied Data Science Master's Programme (N2ADS), 4) Bachelor's Programme in Mathematics (N1MAT), 5) Computer Science, Bachelor's Programme (N1COS), 6) Software Engineering Master's Programme (N2SOM) and 7) Software Engineering and Management Master's Programme (N2SOF)

*Main field of studies*

Computer Science

*Specialization*

G1F, First Cycle, has less than 60 credits in first-cycle course/s as entry requirements

#### **Entry requirements**

The student should have successfully completed at least

- 7.5 hec in imperative/object-oriented programming such as DIT012, DIT948 or equivalent,
- an additional course in programming or data structures.

Moreover, the student must also have knowledge in propositional logic, which is acquired by successfully completing courses such as DIT980, DIT725, the part on introductory algebra from MMGD200, or equivalent.

### **Learning outcomes**

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

#### *Knowledge and understanding*

- demonstrate knowledge of the issues and problems that arise in writing correct concurrent programs;
- identify the problems of synchronization typical of concurrent programs, such as race conditions and mutual exclusion;

#### *Competence and skills*

- apply common patterns, such as lockings, semaphores, and message-passing synchronization for solving concurrent program problems;
- apply practical knowledge of the programming constructs and techniques offered by modern concurrent programming languages.;
- implement solutions using common patterns in modern programming languages;

#### *Judgement and approach*

- evaluate the correctness, clarity, and efficiency of different solutions to concurrent programming problems;
- judge whether a program, a library, or a data structure is safe for usage in a concurrent setting;
- pick the right language constructs for solving synchronization and communication problems between computational units.

### **Course content**

Concurrent and parallel programming has become ubiquitous in modern software and systems, where concurrency is leveraged to exploit physical parallelism and speed up computations, to provide interactive multi-tasking, and to handle interaction with asynchronous external events. This course aims to provide an introduction to the principles underlying concurrent systems, as well as to practical programming solutions for modeling and exploiting concurrency in programs. Domains where such principles and practices are relevant include operating systems, distributed systems, real-time systems, and multicore architectures.

The concepts covered in the course include:

- physical vs logical parallelism
- concurrency problems (race conditions, interference, deadlock, fairness, livelock).
- mutual exclusion
- shared memory synchronization (using semaphores or fine grained locking)
- message-passing synchronization (using message queues)

The course illustrates practical solutions to concurrent programming using both imperative and functional programming languages. Thus, the course will also include short introductory tutorials on functional programming in general and on the functional programming language used in the course, providing sufficient background to understand and use the concurrent programming abstractions demonstrated by means of functional languages.

#### *Sub-courses*

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

#### **Form of teaching**

Lectures, exercise classes, and laboratory sessions.

*Language of instruction:* English

#### **Assessment**

The course is examined by an individual written exam carried-out in an examination hall, and laboratory work that is normally carried out in pairs.

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).

The complete course grade is determined by the score on both the laboratory part and the written exam. To be awarded Pass (G) for a full course, the students must pass both the laboratory part and the exam with the grade G. To be awarded Pass with Distinction (VG) for the full course, the student must in addition obtain at least 80% of the total amount of points in the laboratory part and the exam together.

### **Course evaluation**

The course is evaluated through meetings both during and 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**

Knowledge in functional programming (corresponding for example to DIT440 or DIT142) is beneficial but not required.

The course replaces the DIT390 Concurrent programming, 7.5 hec. The course cannot be included in a degree which contains DIT390. Neither can the course be included in a degree which is based on another degree in which the course DIT390 is included.

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.