

## Compilers (E018520)

**Course size** *(nominal values; actual values may depend on programme)*

**Credits 6.0**

**Study time 180 h**

### Course offerings and teaching methods in academic year 2026-2027

|                |         |      |                  |      |
|----------------|---------|------|------------------|------|
| A (semester 2) | English | Gent | independent work | 0.0h |
|                |         |      | group work       | 0.0h |
|                |         |      | seminar          |      |
|                |         |      | lecture          |      |

### Lecturers in academic year 2026-2027

De Sutter, Bjorn

TW06

lecturer-in-charge

### Offered in the following programmes in 2026-2027

|                                                                                                        | crdts | offering |
|--------------------------------------------------------------------------------------------------------|-------|----------|
| <a href="#">Master of Science in Teaching in Science and Technology(main subject Computer Science)</a> | 6     | A        |
| <a href="#">Master of Science in Computer Science</a>                                                  | 6     | A        |
| <a href="#">Master of Science in Computer Science Engineering</a>                                      | 6     | A        |
| <a href="#">Master of Science in Information Engineering Technology</a>                                | 6     | A        |
| <a href="#">Exchange Programme in Computer Science (master's level)</a>                                | 6     | A        |
| <a href="#">Exchange Programme Information Engineering Technology</a>                                  | 6     | A        |

### Teaching languages

English

### Keywords

context free grammars, derivation trees, parsers, abstract syntax trees, semantic analysis, basic blocks, instruction selection, register allocation, data flow analysis, control flow analysis, code generation, optimization, support for managed languages, garbage collection, pipelining, scheduling, high-level language techniques, just-in-time compilation

### Position of the course

A computer uses a machine language whereas programmes are written in a higher programming language. Compilers form the connection between the programmer and the computer. Beside computer architecture, the compiler is responsible for the performance of the machine. This happens in two steps. In the front end, lexical and parsing techniques are used to convert the source code to an intermediate representation. In the back end, optimization techniques and code generation techniques are deployed to map the representation optimally to the targeted architecture. In this course, theoretical basic concepts are coupled to the implementation of a compiler used world-wide, and to a number of recent developments in the domain.

### Contents

- Introduction: Data structures for compilers, Overview of the different phases in a compilation
- Lexical analysis: Tokens and regular expressions, Finite automata
- Parsing: Context free grammar, Predictive and LR parsers
- Abstract syntax: Semantic actions, Abstract syntax tree
- Semantic analysis: Symbol tables, Type checking
- Activation records: Stack window and local variables, Passing arguments
- Intermediary representation: Derivation tree, Translation of intermediary code
- Basic blocks and traces: Canonical trees, Conditional jumps

- Instruction selection: Algorithms for instruction selection
- Liveness analysis and register allocation: Liveness equations, Graph coloring and register allocation
- Data flow analysis: Data structures for data flow representation, Data flow algorithms
- Loop optimizations: Reducible loops, Loop optimizations, Control flow analysis
- Code generation: pipelining and scheduling
- Memory optimisation: prefetching, loop transformations
- Support for high-level languages: garbage collection, polymorphism, class hierarchies
- Just-in-time compilation techniques
- Capita selecta state-of-the-art compiler techniques

#### **Initial competences**

- Knowledge of programming languages, programming in C
- Basic knowledge of computer architecture (software and hardware)

#### **Final competences**

- 1 Understand and explain the structure of a compiler.
- 2 Understand and explain the construction and operation of automata for lexical analysis.
- 3 Understand, explain, and apply the construction and operation of various parsing techniques and their complexity.
- 4 Understand and explain typechecking on abstract syntax trees.
- 5 Understand, explain, and construct various representations of code, data, dependencies, and attributes.
- 6 Understand, explain, and execute dependency analysis for liveness analysis and code optimization.
- 7 Understand and apply instruction selection algorithms.
- 8 Understand, explain, and apply graph coloring algorithms for register allocation.
- 9 Understand, explain, and deploy intermediate representations for the generation of machine-independent code.
- 10 Understand, explain, and deploy various loop analyses and transformations for code optimization.
- 11 Understand, explain, and deploy various data flow analyses and related code optimizations.
- 12 Understand, explain, and deploy static code scheduling and loop pipelining techniques.
- 13 Understand, explain, and deploy the implementation of support high-level language features such as garbage collection, polymorphism, and class inheritance.
- 14 Programming basic compiler functionality (lexer, parser, semantic analysis, code generation, etc.) in a lab environment.
- 15 Program basic compiler functionality in a world-wide used, complex compiler, using its internal interfaces.

#### **Conditions for credit contract**

Access to this course unit via a credit contract is determined after successful competences assessment

#### **Conditions for exam contract**

This course unit cannot be taken via an exam contract

#### **Teaching methods**

Group work, Seminar, Lecture, Independent work

#### **Extra information on the teaching methods**

In the lectures, the theoretical foundations and internal operation of compilers are discussed, and the operation of basic algorithms is practiced.

For the "lab" sessions, programming assignments have to be completed during and outside sessions under the supervision of assistants. The resulting software, sometimes with a small report, has to be submitted. After their submissions, students can be invited for an individual oral evaluation in which they are questioned about the software they submitted and in which we check whether they obtained the targeted competences. Some sessions concern the LLVM compiler infrastructure, one of the most used compilers world-wide.

In the exercise sessions, the application of basic compiler algorithms on small

examples is practiced.

## Study material

Type: Handbook

Name: Modern Compiler Implementation in C

Indicative price: € 70

Optional: yes

Language : English

Author : Andrew W. Appel

ISBN : 978-0-52160-765-0

Number of Pages : 556

Alternative : Slides and notes taken during lectures and exercise sessions.

Oldest Usable Edition : 2004

Online Available : No

Available in the Library : Yes

Available through Student Association : Yes

Usability and Lifetime within the Course Unit : intensive

Usability and Lifetime within the Study Programme : one-time

Usability and Lifetime after the Study Programme : not

Additional information: The theory and exercises in this course closely match this book. The available lecture slides and the student's notes from the lectures and exercise sessions suffice to prepare for the exam. This book provides written out explanations of the course content for students that prefer such additional study material. On two occasions in the course, students are asked to study a short section of the book themselves. For this, borrowing a copy from a library or a colleague typically will suffice.

Type: Slides

Name: Compilers - Theory Lectures

Indicative price: Free or paid by faculty

Optional: no

Language : English

Number of Slides : 455

Oldest Usable Edition : 2025-2026

Available on Ufora : Yes

Online Available : Yes

Available in the Library : No

Available through Student Association : No

Additional information: -

## References

- Modern compiler implementation in C, Andrew W. Appel with Maia Ginsburg, Cambridge University Press, ISBN 052158390X, LCCN QA76.73.C15.A63
- LLVM Manual, <http://llvm.org/docs/>

## Course content-related study coaching

### Assessment moments

end-of-term and continuous assessment

### Examination methods in case of periodic assessment during the first examination period

Written assessment

### Examination methods in case of periodic assessment during the second examination period

Written assessment

### Examination methods in case of permanent assessment

Oral assessment, Assignment

### Possibilities of retake in case of permanent assessment

examination during the second examination period is possible in modified form

## Extra information on the examination methods

The permanent evaluation consists of seven lab software projects, some of which with small reports, spread over the duration of the semester. The report and software of each lab project has to be submitted before the start of the next assignment or on a given date. Shortly after submission of each report, an oral

evaluation can follow in which the student is questioned regarding the used methodology and the achieved results.

For the lab assignments, students potentially collaborate in groups of two. Based on a peer assessment regarding their contributions to the solution and report, and possibly after a conversation in case of diverging assessments, it is decided whether or not both students get the same score.

If an oral evaluation takes place, this is done individually. Each student in a group is responsible for the whole submitted solution, and can be questioned about all of it.

Reports/solutions that are not submitted on time without acceptable reason and notification (such as a sick note) get a zero score for that part of the continuous assessment.

No code generated with AI-tools is allowed to be submitted.

When code is submitted that is a derivative work of public documentation (e.g., tutorial), this has to be acknowledged by the student submitting the code. The evaluation then focus on the derivation (i.e., made adaptations, their correctness and completeness) than on the whole of the submitted code.

The permanent assessment is similar in the second exam period. The student is then evaluated on the basis of lab assignments with similar work load as those in the first period. In this case, the student needs to make them individually. The reports then have to be submitted a week before the exam. Right after that written exam, an oral evaluation about the reports and solutions to the assignments takes place.

The periodical evaluation is written, with closed book.

#### **Calculation of the examination mark**

Evaluation 3/5 for the examination and 2/5 for the permanent assessment.

Special condition: a score of 10/20 or more on the laboratory exercises, as well as on the exam is required to pass this course. If the score is lower for either part, while the total, weighted score is 10/20 or more, the final total score becomes 9/20.

If a student passes for one of both parts, they can opt to carry those points over to the second chance exam period.

#### **Facilities for Working Students**

Option to be freed from presence in labs with alternative assignment after consultation with the responsible teacher. Option for oral exam with written preparation at another time in the academic year. Option for feedback by appointment during and after business hours.