1.28 Compiler Construction

- **Course name:** Compiler Construction
- **Course number:** n/a
- **Subject area:** Programming Languages and Software Engineering

1.28.1 Course characteristics

1.28.1.1 Key concepts of the class

- Overall compilation architecture
- Lexical analysis
- Syntax analysis
- Semantic analysis
- Code generation
- Program optimization
- Virtual machines and JIT technology

1.28.1.2 What is the purpose of this course?

The software development process and the depth of programming cannot be understood without a detailed analysis of the compilation process, from the lexical analysis to the syntactical and semantic analysis up to code generation and optimization, and without understanding both strength and limitation of this process. This course dig deeper into this topic building on the fundamental notions studied in theoretical computer science of which is the natural continuation. The typical compiler pipeline will be studied and a project will allow students to practice with the relevant tools.

1.28.1.3 Course Objectives Based on Bloom's Taxonomy

What should a student remember at the end of the course?

- Understanding in depth the compilation process
- Realizing the limits of the process and of Semantic Analysis
- Read and write grammars for programming language constructs
- Perform lexical analysis and use lexical analyzer generators
- Perform top-down parsing, bottom-up parsing and use parser generators
- Perform semantic analysis

What should a student be able to understand at the end of the course?

- How to design and develop compilers and language-related tools.
- The contents of each phase of the compilation process.
- How integrate compilers into an IDE.
- How to design a virtual machine for a language.

What should a student be able to apply at the end of the course?

- To be able to develop a language compiler.

1.28.1.4 Course evaluation

Table 1.76: Course grade breakdown

Type	Default points	Proposed points
Labs/seminar classes	40	40
Interim performance assessment	30	40
Exams	30	30

If necessary, please indicate freely your course's features in terms of students' performance assessment.

Labs/seminar classes:

In-class participation 1 point for each individual contribution in a class but not more than 1 point a week (i.e. 14 points in total for 14 study weeks), overall course contribution (to accumulate extra-class activities valuable to the course progress, e.g. a short presentation, book review, very active in-class participation, etc.) up to 6 points.

Interim performance assessment:

In-class tests up to 10 points for each test (i.e. up to 40 points in total for 2 theory and 2 practice tests), computational practicum assignment up to 10 points for each task (i.e. up to 30 points for 3 tasks).

Exams:

Mid-term exam up to 30 points, final examination up to 30 points.

Overall score: 100 points (100%).

1.28.1.5 Grades range

Table 1.77: Course grading range

Grade	Default range	Proposed range
A. Excellent	85-100	85-100
B. Good	75-84	75-84
C. Satisfactory	60-75	60-75
D. Poor	0-59	0-59

If necessary, please indicate freely your course's grading features: The semester starts with the default range as proposed, but it may change slightly (usually reduced) depending on how the semester progresses.

1.28.1.6 Resources and reference material

- Alfred V.Aho, Monica S.Lam, Ravi Sethi, Jeffrey D. Ullman. *Compilers. Principles, Techniques, & Tools, Second Edition, Addison-Wesley, 2007, ISBN 0-321-48681-1.*
- *N. Wirth, Compiler Construction, Addison-Wesley, 1996, ISBN 0-201-40353-6*
- *http://www.ethoberon.ethz.ch/WirthPubl/CBEAll.pdf, 2005*

1.28.2 Course Sections

The main sections of the course and approximate hour distribution between them is as follows:

Table 1.78: Course Sections

Section	Section Title	Lecture Hours	Seminars (labs)	Self-study	Knowledge evaluation
1	Introduction to compilers and compiler construction	12	6	12	2
2	Lexical, syntax and semantic analyses	8	4	8	1
3	Code generation, optimization and virtual machines	8	4	8	1
4	Project presentation				2

1.28.2.1 Section 1

Section title:

Introduction to compilers and compiler construction

Topics covered in this section:

- Basic notions: source and target languages, target architecture, compilation phases.
- The history of languages and compiler development. Typical compiler examples.
- Compilation & interpretation. Virtual machines, JIT & AOT technologies. Hybrid modes.

What forms of evaluation were used to test students' performance in this section?

Form	Yes/No
Development of individual parts of software product code	1
Homework and group projects	1
Midterm evaluation	0
Testing (written or computer based)	0
Reports	0
Essays	0
Oral polls	1
Discussions	1

Typical questions for ongoing performance evaluation within this section

1. What is compilation process?
2. What's the difference between compiler and interpreter?

Typical questions for seminar classes (labs) within this section

1. How to compile a program?
2. How to run a program?
3. How to debug a program?

Test questions for final assessment in this section

1. What are significant phases of a compilation process?
2. Why do we need optimization phase?
3. What's the difference between syntax and semantic analyses?

1.28.2.2 Section 2

Section title:

Lexical, syntax and semantic analyses

Topics covered in this section:

- Compilation pipeline & compilation data structures
- Lexical analysis and deterministic state automata
- Bottom-up and top-down parsing
- Principles of semantic analysis

What forms of evaluation were used to test students' performance in this section?

Form	Yes/No
Development of individual parts of software product code	1
Homework and group projects	1
Midterm evaluation	0
Testing (written or computer based)	0
Reports	1
Essays	0
Oral polls	0
Discussions	1

Typical questions for ongoing performance evaluation within this section

1. Abstract syntax tree & symbol tables: what is it for and how create and manage them?
2. How to organize communication between compilation phases?
3. What are basic differences between bottom-up and top-down parsing?

Typical questions for seminar classes (labs) within this section

1. How to implement a hash function for a symbol table?
2. How to write a grammar for an expression using YACC/Bison tool?

Test questions for final assessment in this section

1. Characterize the principles of top-down and bottom-up parsing.
2. Explain how a symbol table can be implemented.
3. AST: is it a tree or a graph? What's about semantic attributes in an AST?

1.28.2.3 Section 3

Section title:

Code generation, optimization and virtual machines

Topics covered in this section:

- Low-level code generation: machine instructions, assembly language
- Virtual machines' architecture and their byte codes.
- The notion of language projections.
- Introduction to optimization techniques.

What forms of evaluation were used to test students' performance in this section?

Form	Yes/No
Development of individual parts of software product code	1
Homework and group projects	1
Midterm evaluation	0
Testing (written or computer based)	0
Reports	0
Essays	0
Oral polls	0
Discussions	0

Typical questions for ongoing performance evaluation within this section

1. What's the difference between assembly and machine instructions?
2. What's the similarities and differences between real target platforms and virtual machines?
3. Explain some of widely used approaches for optimizing program source code? How these approaches can be implemented in a compiler?

Typical questions for seminar classes (labs) within this section

1. Explain the idea behind the notion of control flow graph.
2. What is "basic block" in CFG and what is it for?

Test questions for final assessment in this section

1. Give some simple examples of language-code projections.
2. How the object code for an expression can be optimized?
3. How to avoid tail recursion while optimizing code?