I075	**Compiler construction**	L	P	S	ECTS
		2	1	1	6

Course objectives. Introducing the concepts of how to design and implement compilers. Students will learn concepts that will enable them to construct translators for a wide variety of languages and machines. They will understand syntax-directed translations, intermediate-code generation and run-time environments. As a final task, students will be required to implement their own simple version of compiler.

Prerequisites. Bachelor in Computer Science.

Course content.

1. Introduction. The structure of a compiler. Evolution of programming languages. Simple syntax-directed translator. Parsing. Translator for simple expressions.
2. Lexical analysis. Symbol tables. Tokens, patterns and lexemes. Finite automata. From regular expressions to automata.
3. Syntax analysis. Context-free grammars. Lexical versus syntactic analysis. Top-down parsing. Bottom-up parsing. LR. Parser generators (Yacc).
4. Syntax-directed translations. Postfix. SDT.
5. Intermediate-code generation. Variants of syntax trees. Three-address code. Types and declarations. Type checking. Control flow. Backpatching.
6. Run-time environments. Storage organization: static vs. dynamic storage allocation. Stack allocation. Access to nonlocal data on the stack. Heap management. Introduction to garbage collector. Introduction to trace-based collection.
7. Code generation. Issues in the design of a code generator. Target language. Addresses in the target code. Basic blocks and flow graphs. Optimization of basic blocks. Simple code generator. Peephole optimization. Register allocation and assignment. Optimal code generation for expressions. Dynamic programming code-generation.

LEARNING OUTCOMES

No.	LEARNING OUTCOMES
1.	Understanding of a simple syntax-directed translator.
2.	Differentiating between different approaches in parsing the program code.
3.	Explaining the lexical code analysis. Being able to use Lex.
4.	Understanding the syntax code analysis. Being able to implement different approaches in parsing.
5.	Understanding notions related to syntax-directed translations.
6.	Complete understanding of intermediate-code generation.
7.	Complete understanding of run-time environments.
8.	Understanding of code generation process. Being able to implement a simplified compiler.

RELATING THE LEARNING OUTCOMES, ORGANIZATION OF THE EDUCATIONAL PROCESS AND ASSESSMENT OF THE LEARNING OUTCOMES

TEACHING ACTIVITY	ECTS	LEARNING OUTCOME **	STUDENT ACTIVITY*	EVALUATION METHOD	POINTS	
					min	max

Attending lectures and exercises	1	1-8	Lecture attendance, discussion, teams work, independent work on given tasks and short written exams	Attendance lists, tracking activities, closed form exercises	0	4
Homework assignments	1	1-8	Independent work on given problems	Evaluation	0	4
Written exam (Mid-terms)	2	1-8	Preparing for written exam	Evaluation	25	46
Final exam	2	1-8	Revision	Oral exam	25	46
TOTAL	6				50	100

Teaching methods and student assessment. Lectures and exercises are obligatory. The exam consists of a written and an oral part. Upon completion of the course, students can take the exam. Successful midterm exam scores replace the written exam. Exercises are both auditory and laboratory. Laboratory exercises include the usage of computers. Students can improve their grades by writing homework assignments and seminars.

Can the course be taught in English: Yes

Basic literature:

1. A.V. Aho, M.S. Lam, R. Sethi, J.D. Ullman, Compilers: Principles, Techniques, and Tools (2nd Edition), Pearson Education Limited, 2014.

Recommended literature:

1. K. Cooper, L. Torczon, Engineering: A Compiler (2nd edition), Elsevier 2012.
2. H. Seidl, R. Wilhelm, S. Hack, Compiler Design – Analysis and Transformation, Springer Verlag, 2012.