Poznan University of Technology
Faculty of Electrical Engineering

European **C**redit **T**ransfer **S**ystem

STUDY MODULE DESCRIPTION FORM

Name of the module/subject **Formal languages and compilers**		Code **1010331531010330115**
Field of study **Information Engineering**	Profile of study (general academic, practical) **(brak)**	Year /Semester **2 / 3**
Elective path/specialty **-**	Subject offered in: **Polish**	Course (compulsory, elective) **obligatory**
Cycle of study: **First-cycle studies**	Form of study (full-time,part-time) **full-time**	
No. of hours Lecture: **15** Classes: **15** Laboratory: **15** Project/seminars: **-**		No. of credits **4**
Status of the course in the study program (Basic, major, other) **(brak)**	(university-wide, from another field) **(brak)**	
Education areas and fields of science and art **technical sciences**		ECTS distribution (number and %**)** **4 100%**

Responsible for subject / lecturer:

dr inż. Jolanta Cybulka
email: jolanta.cybulka@put.poznan.pl
tel. 0-61 6653724
Wydział Elektryczny
ul. Piotrowo 3A 60-965 Poznań

Prerequisites in terms of knowledge, skills and social competencies:

1	**Knowledge**	1. Student has the ground knowledge of mathematics, especially algebra, logic, mathematical analysis, statistics and elements of discrete and applied mathematics. 2. Student has grounded and theoretically founded elementary knowledge in algorithmics, abstract data types and their implementation, and also computational theory and practice.
2	**Skills**	1. Student can by herself/himself acquire knowledge from the literature, databases and other sources; can also integrate the acquired knowledge, interpret it, reason, formulate conclusions and justify them. 2. Student can use programming platforms and environments to design, run and debug simple programs written in imperative, object-oriented and declarative programming languages.
3	**Social competencies**	Student knows that she/he is obliged to perform well her/his job and also knows that she/he is obliged to perform well the part of assigned to her/him part of teamwork.

Assumptions and objectives of the course:

Presentation of elements of the theory of formal languages and elements of the theory of translation. Introducing syntax-directed translation methods and tools in order to develop the ability to create the simple formal language processing scripts/systems.

Study outcomes and reference to the educational results for a field of study

Knowledge:

1. . Student has structured and theoretically grounded knowledge of: basic programming constructs, implementation of algorithms, paradigms and styles of programming, methods of verifying the correctness of programs, and formal languages and compilers. - [K_W05]

2. Student has structured and theoretically grounded knowledge of basic algorithms and their analysis, algorithm design techniques, abstract data types and their implementation, and also of computationally complex problems. - [K_W04]

Skills:

1. Student is able to create algorithms using basic algorithmic techniques and also can analyze their computational complexity. - [K_U09]

2. Student is able to assess the usefulness of routine methods and tools to solve simple computer engineering tasks, and is able to select and use appropriate technologies. - [K_U22]

Social competencies:

1. Student is aware of the importance of the accurate completion of the project, using the right notational standards, respecting the linguistic correctness and submitting the work on time. - [K_K07]

Assessment methods of study outcomes

Lectures and classes: writing test (checking the knowledge on the theory of formal languages and the theory of translation), minimal score 50,1%

Laboratory: 2 writing tests which check the skills in programming text transducers, written in one of the three text-processing languages Lex and YACC; minimal score 50,1%.

Course description

Lectures:

The notion of a symbolic formal language. Alphabet, syntax and semantics of a formal language. The generative (combinatorial grammars-like) and the acceptor (abstract machine-driven) approaches to defining language syntax. Noam Chomsky?s classification of formal languages. Regular languages: finite automata, regular expressions. Using Lex system to process regular languages. Context-free languages: pushdown automata, context-free grammars. Context and computational languages and their acceptor automata. The notion of a translation, syntax-directed definition, translation scheme. Deterministic context-free languages (LL and LR) and their acceptor automata. Using YACC to process context-free languages. Preliminaries concerning formal methods of defining the semantics of programming languages (operational, denotational and axiomatic). Translation: interpreting vs compiling. Phases and runs of a compiler. Applying the syntax-directed translation to define the analytic phases of a compiler: lexical, syntactic and context-dependent. Basics of intermediate and final code generation, concept of an intermediate language. Basics of a run-time system: storage allocation, accessing the non-local variables and parameter passing.

Classes:

Solving problems connected with formalizing exemplary languages and specifying their acceptors (transducers) formulated as syntax-directed definitions (Modification in 2017: specifying fragments of a compiler for a simple programming language (SPL)).

1. Regular expressions and defining a scanner for SPL.
2. Finite state automata.
3. Contex-free grammars.
4. Context-free grammars II, pushdown automata (modification 2017: defining a parser for SPL).
5. Translation schemes.
6. (modification 2017) Defining a preprocesor for SPL to some other simple high-level programming language.
7. Test.

Laboratory:

Implementing text transducers by using Lex and YACC systems in the Linux environment.

1. Basics concerning running environment + Lex.
2. Programming general text transducers in Lex.
3. Programming a scanner for SPL (modification 2017) in Lex.
4. Test concerning Lex.
5. Programming parsers in YACC
6. Programming syntax-directed translators in YACC.
7. Test concerning YACC.

Applied methods of education:

a) lectures illustrated by slides and examples of running programs

b) classes: solving problems/excercises by students, discussion over solutions (additionally credited)

c) laboratory: programming text transducers in laboratory in order to prepare to pass the written test.

Basic bibliography:

1. Cybulka J., Jankowska B., Nawrocki J. R.: Automatyczne przetwarzanie tekstów. AWK, Lex i YACC, Wyd. NAKOM, Poznań, 2002

2. Hopcroft J.E., Ullman J.D.: Wprowadzenie do teorii automatów, języków i obliczeń, PWN, Warszawa, 1994.

3. Aho A.V., Sethi R., Ullman J.: Kompilatory. Reguły, metody I narzędzia. WNT, Warszawa 2002

Additional bibliography:

1. Dembiński P., Małuszyński J.: Matematyczne metody definiowania języków programowania, WNT, Warszawa 1981.

2. Kernighan B.W., Ritchie D.M.: Język ANSI C, WNT, 1994.

Result of average student's workload

Activity	Time (working hours)

1. lecture 2. classes 3. laboratory 4. tests and consultations 5. preparation for classes 6. preparation for laboratory 7. preparation to test: lecture+classes 8. preparation for tests: laboratory	15 15 15 5 10 10 15 15

Student’s workload

Source of workload	hours	ECTS
Total workload	100	4
Contact hours	50	2
Practical activities	50	2