STUDY MODULE DESCRIPTION FORM

Name of the module/subject	Code
Formal Languages and Compilers	**1010514361010510193**

Field of study	Profile of study (general academic, practical)	Year /Semester
Computing	**general academic**	**3 / 6**
Elective path/specialty	Subject offered in:	Course (compulsory, elective)
-	**English**	**elective**

Cycle of study:	Form of study (full-time,part-time)
First-cycle studies	**part-time**

No. of hours	No. of credits
Lecture: **18** Classes: **-** Laboratory: **18** Project/seminars: **-**	**5**

Status of the course in the study program (Basic, major, other)	(university-wide, from another field)
major	**from field**

Education areas and fields of science and art	ECTS distribution (number and %)
technical sciences	**5 100%**
Technical sciences	**5 100%**

Responsible for subject / lecturer:

Ph. D. Wojciech Complak
email: Wojciech.Complak@cs.put.poznan.pl
tel. (0-61) 665-2983
Faculty of Computing
Piotrowo 3, 60-965 Poznań

Responsible for subject / lecturer:

Ph. D. Dr. Habil. Jerzy Nawrocki
email: Jerzy.Nawrocki@cs.put.poznan.pl
tel. (0-61) 665-3422
Faculty of Computing
Piotrowo 3, 60-965 Poznań

Prerequisites in terms of knowledge, skills and social competencies:

1	**Knowledge**	Students starting this course should have a basic knowledge of algorithms and programming in imperative languages.
2	**Skills**	Students should be able to solve basic problems in the range of design, checking the correctness and implementing algorithms in the C programming language and the ability to acquire information from the indicated sources.
3	**Social competencies**	Students should also understand the necessity to broaden own competences/be ready to cooperate within the team. In addition, in the field of social competence, the student must present such attitudes as honesty, responsibility, perseverance, cognitive curiosity, creativity, personal culture, respect for other people.

Assumptions and objectives of the course:

1. Teach students basic knowledge of practical aspects of formal languages theory and the construction of translators and run-time environments in the range of principles, techniques and tools used today to build compilers and other automatic text processing tools, such as word processors, information retrieval systems, typesetting systems and program verifiers.

2. Develop students' ability to solve simple problems using general-purpose programming languages as well as using specialized tools for this purpose. Expanding knowledge about previously used programming environments and programming languages as a result of looking at them from the perspective of the designer and the implementer, not just the user

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

Knowledge:

1. the student has an expanded and deep knowledge of mathematics useful for formulating and solving complex IT tasks related to formal specification and verification of software - [K1st_W1]

2. the student has a structured and theoretically founded general knowledge in the field of algorithms, computer systems architecture and programming paradigms - [K1st_W4]

3. the student knows the basic techniques, methods and tools used in the process of solving IT problems in the field of algorithm analysis, computer system architecture, operating systems and implementation of programming languages - [K1st_W7]

Skills:

1. the student can properly plan and carry out functional and non-functional test of software - [K1st_U3]

2. the student can, by formulating and solving IT tasks, apply properly selected analytical and experimental methods - [K1st_U4]

3. the student is able to specify and implement analyzers using known tools - [K1st_U11]

Social competencies:

1. the student is aware of the importance of knowledge in solving engineering problems and knows examples and understands the reasons for malfunctioning IT systems that led to serious financial and social losses or to serious health conditions or even to death - [K1st_K2]

Assessment methods of study outcomes

The learning outcomes presented above are verified in the following way:

Formative assessment:

a) lectures:

- based on answers to questions related to subjects covered during former lectures,

b) laboratory classes:

- based on assessment of progress of implementation of assigned tasks,

Total assessment:

Verification of assumed learning objectives is based on:

- evaluation of preparation for individual laboratory sessions (initial test) and assessment of skills related to solving laboratory exercises,

-continuous assessment, during each class (verbal answers) - rewarding the progress in the ability to use the principles and methods learned,

- assessment of knowledge and skills related to the implementation of project/laboratory tasks through test at the end of the semester, test includes 11 practical tasks regarding specific tools and issues discussed in the course (2 x AWK, 2 x lex, 2 x LLgen, 2 x yacc, 3 x SLR), tasks are both constructional (i.e. write a program) and analytical (i.e. determine the result of a given program).

- assessment of the project implementation report,

Additional elements cover:

- discussing additional aspects of the class topic,

- ability of using the acquired knowledge while solving a given problem

- remarks related to improving teaching materials.

- pointing out perceptual difficulties enabling ongoing improvement of the teaching process

Course description

The first lecture is devoted to presenting the organization of the course (its scope, the environment and tools, literature and rules for passing) and the introduction to the problem of text processing using the AWK language as an example.

The second lecture is dedicated to the analysis-synthesis model of the translator and the decomposition of the translation process into phases. The phase of lexical analysis and the principles of its conducting with the use of the lex lexical analyser generator is presented.

The first laboratory classes are committed to organizational issues: familiarizing with the environment and tools, running scripts for compilation and learning how to use the AWK language for text processing.

The third lecture, opening the series on syntax analysis, discusses the general principles of syntax analysis and concepts related to context-free grammars (such as terminals and non-terminals, productions, arguments, types of recursion, ambiguity and equivalence of grammars) and introduction to the top-down method.

LLgen, a syntax analyser generator using top-down method is presented in the further part of the series devoted to the syntax analysis. During the first lecture dedicated to this generator, general principles of its operation and constructing specifications of syntax analysers are presented.

During laboratory classes, students continue working on the design and implementation of simple text filters using a lex generator.

The next lecture is committed to the idea of syntax-directed translation. The concepts of attributes, syntax directed definitions, translation schemes, and S-attributed and L-attributed definitions are presented. The rules for implementing syntax-directed translation in the LLgen generator are also discussed.

The sixth lecture is devoted to the bottom-up method, principles of construction and operation of analysers utilising this method and the yacc generator. This lecture includes the characteristics of the yacc generator, the syntax rules for analysers specification, the principles of cooperation with the lexical analyser, and the detection and handling of syntax errors.

During the laboratory classes, students, while implementing simple text filters, learn the basics of the LLgen generator and the principles of cooperation of the syntax and lexical analysers.

The seventh lecture is devoted to the principles of implementation of syntax-directed translation in the bottom-up method in the yacc generator (synthesized and inherited attributes, attribute types, multiple actions).

The next lecture in the cycle on syntax analysis is devoted to the use of ambiguous grammars in the bottom-up method in the yacc generator. The advantages and typical practical examples of ambiguous grammars and the principles of using them in the yacc generator are presented.

During the laboratory classes the students independently implement a simple imperative programming language analyser with the use of LLgen and lex generators.

The ninth lecture is dedicated to the semantic analysis phase. Different types of context dependencies control, such as:

control flow verification, uniqueness of name declarations, name repetitions and type control are presented.

The concluding lecture on syntax analysis is devoted to the comparison of the advantages and disadvantages of various methods of creating translators based on the bottom-up method and demonstration of principles of parser code generation.

During laboratory classes, students solve tasks related to the yacc generator.

The next lecture is devoted to the intermediate code generation phase: various types of intermediate languages and virtual machines are discussed, and as an example of a specific implementation, the three-address code is presented in detail.

The twelfth lecture concerns the target code generation and its optimization as well as the issues related to building the run-time environment, such as access to nonlocal data, dynamic storage allocation and passing parameter to subroutines.

During laboratory classes, students using lex and yacc, start implementing a translator between selected, familiar, imperative languages.

The last presented during lectures tool is the integrated ANTLR generator environment.

Students complete the implementation and carry out translator tests during laboratory classes.

The last lecture is devoted to summary of all issues presented during the semester and checking students' knowledge in the form of a final test.

During last laboratory classes students present finished translator projects.

Teaching methods:

1. lecture: multimedia presentation, presentation illustrated with examples shown on the blackboard, solving tasks, programming tools demonstration,

2. laboratory classes: solving tasks, discussion.

Basic bibliography:

1. Compilers: Principles, Techniques, and Tools, 2. Ed., A.V. Aho, M. S. Lam, R. Sethi, J.D. Ullman, Addison-Wesley, 2007

2. Gawk: Effective AWK Programming, https://www.gnu.org/software/gawk/manual/

3. lex & yacc, 2nd Edition, D. Brown, J. Levine, T. Mason, O'Reilly Media, 1992

Additional bibliography:

1. The Definitive ANTLR Reference: Building Domain-Specific Languages, T. Parr, The Pragmatic Bookshelf, 2007

2. Theory and Practice of Compiler Writing, J-P. Sorenson, P.G.Tremblay, McGraw-Hill, 1986

Result of average student's workload

Activity	Time (working hours)
1. participating in laboratory classes / tutorials	15
2. preparing for laboratory classes	30
3. consulting issues related to the subject of the course; especially related to the laboratory classes and projects	2
4. implementing a program / programs, running and verification (beyond the time of the laboratories)	10
5. preparing for tests	10
6. participating in lectures	30
7. reading indicated literature/teaching materials (10 pages of scientific text = 1 hour), 50 pages	5
8. preparation for and participation in the final test	15

Student's workload

Source of workload	hours	ECTS
Total workload	118	5
Contact hours	38	2
Practical activities	28	1