COURSE DESCRIPTION CARD - SYLLABUS

Course name

Formal Languages and Compilers

Course

Field of study
Computing

Year/Semester
2 / 4

Area of study (specialization)

Profile of study
general academic

Level of study
First-cycle studies

Course offered in
English

Form of study
full-time

Requirements
elective

Number of hours

Lecture	Laboratory classes	Other (e.g. online)
16	16	0
Tutorials	**Projects/seminars**	
0	0	

Number of credit points

3

Lecturers

Responsible for the course/lecturer:
Ph. D. Wojciech Complak
email: Wojciech.Complak@cs.put.poznan.pl
tel. (0-61) 665-2983
Faculty of Computing and Telecommunications
Piotrowo 3, 60-965 Poznań

Responsible for the course/lecturer:
Ph. D. Dr. Habil. Jerzy Nawrocki
email: Jerzy.Nawrocki@cs.put.poznan.pl
tel. (0-61) 665-3422
Faculty of Computing and Telecommunications
Piotrowo 3, 60-965 Poznań

Prerequisites

Students starting this course should have a basic knowledge of algorithms and programming in imperative languages.
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.
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.

Course objective

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 specialised 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

Course-related learning outcomes

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

Skills

1. the student can properly plan and carry out functional and non-functional test of software
2. the student can, by formulating and solving IT tasks, apply properly selected analytical and experimental methods
3. the student is able to specify and implement analysers using known tools

Social competences

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

Methods for verifying learning outcomes and assessment criteria

Learning outcomes presented above are verified as follows:
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,
- continual 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 practical tasks regarding specific tools and issues discussed in the course (AWK, lex, yacc, SLR); tasks are both constructional (i.e. write a program) and analytical (i.e. determine the result of a given program).
- assessment of the student's 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

Programme content

The first lecture is devoted to presenting the organisation of the course (its scope, the environment and tools, literature and grading rules) 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 conducting it using the lex lexical analyser generator are presented.
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, derivations, types of recursion, ambiguity and equivalence of grammars) and introduction to the bottom-up method. 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 next lecture, the idea of syntax-directed translation is presented. The concepts of attributes, syntax-directed definitions, translation schemes, and S-attribute and L-attribute definitions are presented. The rules of implementing syntax-directed translation in the bottom-up method in the yacc generator (synthesized and inherited attributes, attribute types, multiple actions) are also discussed.
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.
The sixth 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.
The last 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. This lecture also concerns the target code generation and its

optimisation 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.

The first laboratory classes are committed to organisational issues: familiarising with the environment and tools, running scripts for compilation and learning how to use the AWK language for text processing. During the following classes, students start working on the design and implementation of simple text filters using the lex generator.
The remaining laboratory classes are devoted to the bottom-up method and lex and yacc generators: implementation of simple text filters, rules of combining syntax and lexical analysers, implementation of a simple imperative programming language analyser and a translator of the code between selected, known imperative languages.
During the last laboratory meeting students present finished translator projects and write the final test.

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

Bibliography

Basic

1. Kompilatory. Reguły, metody i narzędzia, A. V. Aho, R. Sethi, J. D. Ullman, WNT, Warszawa, 2002
2. Automatyczne przetwarzanie tekstów, J. Cybulka, B. Jankowska, J. Nawrocki, Nakom, Poznań, 2002
3. Wprowadzenie do przetwarzania tekstów w języku AWK, J. Nawrocki, W. Complak, Nakom (Pro Dialog), Poznań, 1994
4. Wprowadzenie do generatora Lex, J. Nawrocki, A. Czajka, Nakom (Pro Dialog), Poznań, 1998

Additional

1. lex & yacc, 2nd Edition, D. Brown, J. Levine, T. Mason, O'Reilly Media, 1992
2. The Definitive ANTLR Reference: Building Domain-Specific Languages, T. Parr, The Pragmatic Bookshelf, 2007
3. Compilers: Principles, Techniques, and Tools, 2. Ed., A.V. Aho, M. S. Lam, R. Sethi, J.D. Ullman, Addison-Wesley, 2007

Breakdown of average student's workload

	Hours	ECTS
Total workload	75	3,0
Classes requiring direct contact with the teacher	32	1,5
Student's own work (literature studies, preparation for laboratory classes, preparation for tests/exam, project preparation) [1]	43	1,5

[1] delete or add other activities as appropriate