SYLLABUS

Formal Languages and Compiler Design

University year 2025-2026

1. Information regarding the programme

1.1. Higher education institution	**Babeş Bolyai University**
1.2. Faculty	**Faculty of Mathematics and Computer Science**
1.3. Department	**Department of Computer Science**
1.4. Field of study	**Computer Science**
1.5. Study cycle	**Bachelor**
1.6. Study programme/Qualification	**Computer Science**
1.7. Form of education	**Full time**

2. Information regarding the discipline

2.1. Name of the discipline	**Formal Languages and Compiler Design**					Discipline code	MLE5023
2.2. Course coordinator	**Prof.PhD. Simona Motogna**						
2.3. Seminar coordinator	**Prof.PhD. Simona Motogna**						
2.4. Year of study	3	2.5. Semester	5	2.6. Type of evaluation	E	2.7. Discipline regime	Mandatory

3. Total estimated time (hours/semester of didactic activities)

3.1. Hours per week	**6**	of which: 3.2 course	**2**	3.3 seminar/laboratory/project	**2 sem + 2 lab**
3.4. Total hours in the curriculum	84	of which: 3.5 course	28	3.6 seminar/laboratory/project	**56**

Time allotment for individual study (ID) and self-study activities (SA)	**hours**
Learning using manual, course support, bibliography, course notes (SA)	15
Additional documentation (in libraries, on electronic platforms, field documentation)	8
Preparation for seminars/labs, homework, papers, portfolios and essays	10
Tutorship	3
Evaluations	5
Other activities:	-
3.7. Total individual study hours	**41**
3.8. Total hours per semester	**125**
3.9. Number of ECTS credits	**5**

4. Prerequisites (if necessary)

4.1. curriculum	Data Structures and Algorithms
4.2. competencies	Average programming skills in a high level programming language

5. Conditions (if necessary)

5.1. for the course	Course room with projector
5.2. for the seminar /lab activities	Laboratory with computers; high level programming language environment (.NET or any Java/Python environment a.s.o.)

6.1. Specific competencies acquired [1]

[1] One can choose either competences or learning outcomes, or both. If only one option is chosen, the row related to the other option will be deleted, and the kept one will be numbered 6.

Professional/essential competencies	• advanced programming skills in high-level programming languages • use of theoretical foundations of computer science as well as of formal models
Transversal competencies	• application of organized and efficient work rules, of responsible attitudes towards the didactic-scientific field, to bring creative value to own potential, with respect for professional ethics principles and norms • use of efficient methods and techniques to learn, inform, research and develop the abilities to bring value to knowledge, to adapt at the requirements of a dynamical society and to communicate efficiently in Romanian language and in an international language

6.2. Learning outcomes

Knowledge	- The graduate has knowledge related to programming, mathematics, engineering and technology and has the skills to use them to create complex information technology systems. - The graduate has the knowledge to select and use appropriate instructional procedures to facilitate the process of knowledge assimilation.
Skills	- The graduate is able to present and explain methods, algorithms, paradigms and techniques used in various branches of computer science. - The graduate is able to identify complex problems and examine related issues to develop solving options and implement solutions. - The graduate is able to combine diverse information to formulate solutions and generate ideas for developing new products and applications.
Responsibility and autonomy:	- The graduate has the ability to apply general rules to specific problems and produce relevant solutions. - The graduate has the ability to choose and use programming paradigms (procedural, object-oriented, functional) to develop software applications appropriate for the specific domain of the application being developed.

7. Objectives of the discipline (outcome of the acquired competencies)

7.1 General objective of the discipline	• Be able to understand compiler design and to implement compiler techniques • Improved programming skills
7.2 Specific objective of the discipline	• Acquire knowledge about back-end of a compiler • Understand and work with formal languages concepts: Chomsky hierarchy; regul automata and the equivalence between them; context-free grammars, push-dow their equivalence • Understand and work with compilers concepts: scanning, parsing

8. Content

8.1 Course	Teaching methods	Remarks

1. General Structure of a compiler. Compiler phases	Exposure: description, explanation, examples, discussion of case studies	
2. Scanning (Lexical Analysis)	Exposure: description, explanation, examples, discussion of case studies	
3. Introductory notions of formal languages. Grammars and Finite Automata	Exposure: description, explanation, examples, debate, dialogue	
4. Regular languages, regular expressions, equivalence between finite automata, regular grammars and regular expressions. Pumping lemma	Exposure: description, explanation, examples, proofs	
5. Context-free grammars, syntax tree	Exposure: description, explanation, examples, discussion of case studies	
6. Parsing: general notions, classification.	Exposure: description, explanation, examples, discussion of case studies	
7. Recursive-descendant parser	Exposure: description, explanation, examples, discussion of case studies	
8. LL(1) parser	Exposure: description, explanation, examples, discussion of case studies	
9. LR(k) parsing method. LR(0) parser	Exposure: description, explanation, examples, discussion of case studies	
10. SLR, LR(1), LALR parser	Exposure: description, explanation, examples, discussion of case studies	
11. Scanner generator (lex); Parser generators (yacc)	Exposure: description, examples, discussion of case studies, live demo	
12. Attribute grammars; generation of intermediary code	Exposure: description, explanation, examples, discussion of case studies	
13. Code optimization and object code generation	Exposure: description, explanation, examples, discussion of case studies	
14. Push-down automata and Turing machines	Exposure: description, explanation, examples, discussion of case studies	

Bibliography

1. A.V. AHO, D.J. ULLMAN - Principles of computer design, Addison-Wesley, 1978.
2. A.V. AHO, D.J. ULLMAN - The theory of parsing, translation and compiling, Prentice-Hall, Engl. Cliffs., N.J., 1972, 1973.
3. D. GRIES - Compiler construction for digital computers,, John Wiley, New York, 1971.
4. MOTOGNA, S. – Metode de proiectare a compilatoarelor, Ed. Albastra, 2006
5. SIPSER, M., Introduction to the theory of computation, PWS Pulb. Co., 1997
6. CSÖRNYEI ZOLTÁN, Bevezetés a fordítóprogramok elméletébe, I, II., ELTE, Budapest, 1996
7. L.D. SERBANATI - Limbaje de programare si compilatoare, Ed. Academiei RSR, 1987.
8. CSÖRNYEI ZOLTÁN, Fordítási algoritmusok, Erdélyi Tankönyvtanács, Kolozsvár, 2000.
9. DEMETROVICS JÁNOS-DENEV, J.-PAVLOV, R., A számítástudomány matematikai alapjai, Nemzeti Tankönyvkiadó, Budapest, 1999
10. GRUNE, DICK - BAL, H. - JACOBS, C. - LANGENDOEN, K.: Modern Compiler Design, John Wiley, 2000

8.2 Seminar / laboratory	Teaching methods	Remarks
1. Specification of a programming language; BNF	Explanation, dialogue, case studies	
2. Grammars; language generated by a grammar; grammar corresponding to a language	Dialogue, debate, case studies, examples, proof	

3. Finite automata: language generated by a FA; FA corresponding to a language	Dialogue, debate, case studies, examples, proof	
4. Transformations: finite automata – regular grammars	Dialogue, debate, case studies, examples, proof	
5. Transformations: regular expressions – finite automata	Dialogue, debate, case studies, examples, proof	
6. Transformations: regular expressions – regular grammars	Dialogue, debate, case studies, examples, proof	
7. Context free grammars; descendent recursive parser	Dialogue, debate, case studies, examples, proof	
8. LL(1) parser	Dialogue, debate, case studies, examples, proof	
9. LR(0) parsers	Dialogue, debate, case studies, examples, proof	
10. SLR parser	Dialogue, debate, case studies, examples, proof	
11. LR(1) parser	Dialogue, debate, case studies, examples, proof	
12. Attribute grammars	Dialogue, debate, case studies, examples, proof	
13. Intermediary code	Dialogue, debate, case studies, examples, proof	
14. Push down automata	Dialogue, debate, case studies, examples, proof	
Laboratory		
Task 1: Specify a mini-language and implement scanner 1.1: Mini language specification (BNF notation)	Explanation, dialogue, case studies	
Task 1: Specify a mini-language and implement scanner 1.2: implement main functions in scanning	Explanation, dialogue, case studies	
Task 1: Specify a mini-language and implement scanner 1.3: Symbol Table organization	Testing data discussion, evaluation	
Task 2: regular grammars + finite automata + transformations 2.1: Define data structures for RG and FA; implement transformations	Explanation, dialogue, case studies	
Task 2: regular grammars + finite automata + transformations 2.2: Main program, testing + delivery	Testing data discussion, evaluation	
Task 3: context free grammars + equivalent transformations of cfg 3.1: extend task 2 for cfg; implement transformations	Explanation, dialogue, case studies	
Task 3: context free grammars + equivalent transformations of cfg 3.2: Main program, testing + delivery	Testing data discussion, evaluation	
Task 4: Parser implementations 4.1: define data structures and architecture of application	Explanation, dialogue, case studies	One of: descendant recursive, LL(1), LR(0), SLR
Task 4: Parser implementations 4.2: implement main functions in parsing	Explanation, dialogue, case studies	Task 4 is developed in teams of 2 students (teamwork)
Task 4: Parser implementations 4.3: main program and module integration	Explanation, dialogue, case studies	
Task 4: Parser implementations 4.4: testing on small formal grammars	Testing data discussion, evaluation	
Task 4: Parser implementations 4.5: testing on mini-language; delivery	Testing data discussion, evaluation	

Task 5: use tools for lexer and parser generator: lex, yacc – implementation + delivery	Explanation, dialogue, case studies	

Bibliography:
1. A.V. AHO, D.J. ULLMAN - Principles of computer design, Addison-Wesley, 1978.
2. A.V. AHO, D.J. ULLMAN - The theory of parsing, translation and compiling, Prentice-Hall, Engl. Cliffs., N.J., 1972, 1973.
3. MOTOGNA, S. – Metode de proiectare a compilatoarelor, Ed. Albastra, 2006
4. G. MOLDOVAN, V. CIOBAN, M. LUPEA - Limbaje formale si automate. Culegere de probleme, Univ. Babes-Bolyai, Cluj-Napoca, 1996.

9. Corroborating the content of the discipline with the expectations of the epistemic community, professional associations and representative employers within the field of the program

- The course respects the IEEE and ACM Curriculla Recommendations for Computer Science studies;
- The course exists in the studying program of all major universities in Romania and abroad;
- The content of the course is considered the software companies as important for average programming skills

10. Evaluation

Activity type	10.1 Evaluation criteria	10.2 Evaluation methods	10.3 Percentage of final grade
10.4 Course	- know the basic principle of the domain; - apply the course concepts - problem solving	Written exam	60%
10.5 Seminar/laboratory	- be able to apply algorithms, understand examples - problem solving	problems solved - homeworks delivered - continuous observations during semester	10%
	- be able to implement course concepts and algorithms - apply techniques for different classes of programming languages	-Practical examination duri documentation -portofolio -continous observations	30%
10.6Minimum standard of performance			
• Attend 75% of seminar activities during semester AND attend 90% of lab activities during semester • At least grade 5 (from a scale of 1 to 10) at both written exam and laboratory work. • Understand basic concepts of formal languages: grammars, finite automata; be able to apply scanning and parsing algorithms			

11. Labels ODD (Sustainable Development Goals)[2]

Not applicable.

Date: 12.04.2025

Signature of course coordinator: Prof.dr. Simona Motogna

Signature of seminar coordinator: Prof.dr. Simona Motogna

Date of approval: ...

Signature of the head of department: Assoc.prof.phd. Adrian STERCA

[2] Keep only the labels that, according to the *Procedure for applying ODD labels in the academic process*, suit the discipline and delete the others, including the general one for *Sustainable Development* – if not applicable. If no label describes the discipline, delete them all and write „*Not applicable*.".