SYLLABUS

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**

2. Information regarding the discipline

2.1 Name of the discipline	**Advanced Compiler Design**						
2.2 Course coordinator	**Assoc.Prof.PhD. Simona Motogna**						
2.3 Seminar coordinator	**Assoc.Prof.PhD. Simona Motogna**						
2.4. Year of study	**3**	2.5 Semester	**6**	2.6. Type of evaluation	**C**	2.7 Type of discipline	**Optional**

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

3.1 Hours per week	3	Of which: 3.2 course	2	3.3 seminar/laboratory	1
3.4 Total hours in the curriculum	42	Of which: 3.5 course	28	3.6 seminar/laboratory	14

Time allotment:	Hours
Learning using manual, course support, bibliography, course notes	8
Additional documentation (in libraries, on electronic platforms, field documentation)	7
Preparation for seminars/labs, homework, papers, portfolios and essays	8
Tutorship	2
Evaluations	8
Other activities:	-

3.7 Total individual study hours	33
3.8 Total hours per semester	75
3.9 Number of ECTS credits	5

4. Prerequisites (if necessary)

4.1. curriculum	• Formal Languages and Compiler Design course
4.2. competencies	• Basic knowledge of front-end of a compiler • Medium programming skills

5. Conditions (if necessary)

5.1. for the course	•
5.2. for the seminar /lab activities	• Laboratory: computers and use of a programming language environment

6. Specific competencies acquired

Professional competencies	• Knowledge, understanding and use of basic concepts of theoretical Computer Science • Ability to work independently and/or in a team in order to solve problems in defined professional contexts. • Good programming skills in high-level languages

Transversal competencies	• Ability to apply compiler techniques to different real life problems • Ability to model phenomena using formal languages • Improved programming abilities: debugging and correcting compilers errors

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 • Be able to understand compiler optimizations • Improved programming skills
7.2 Specific objective of the discipline	• Acquire knowledge about back-end of a compiler • Understand concepts: virtual machine, JIT compilation, compiler optimizations, machine code generation

8. Content

8.1 Course	Teaching methods	Remarks
1. Review compiler phases. Semantic analysis. Define attribute grammar. [1,4]	Conversation: debate, dialogue; exposuse: description, explanation, examples	
2. Attribute grammar evaluators. L-attributed grammars, S-attributed grammars [2,4]	exposure: description, explanation, examples, discussion of case studies	
3. Manual methods [2,4]: Control flow graph, Symbolic interpretation, Data flow equations	exposure: description, explanation, example	
4. Intermediary code generation [1,2,4]. Three-address code: quadruples, triples	exposure: description, explanation, example; dialogue, case studies	
5. Intermediary code optimization [1,4]	exposure: description, explanation, example, dialogue, debate	
6. Object code generation. Optimizations of the object code [1,2,4]	exposure: description, explanation, example, discussion of case studies	
7. Compiler design for imperative and object-oriented languages (I): Identification, Type checking, Type table, Source Language Data Representation & Handling [2]	exposure: description, explanation, example, dialogue, debate, case studies	
8. Compiler design for imperative and object-oriented languages (II):, Functions- activation records, Object Type, Inheritance, Polymorphism [2,3]	exposure: description, explanation, example, case studies, dialogue, debate	
9. Compiler design for functional languages [2,3]	exposure: description, explanation, example, case studies, dialogue, debate	
10. Compiler design for logical languages [2,3]	exposure: description, explanation, example, case studies, dialogue, debate	
11. Memory management: Garbage Collection	exposure: description,	

mechanism [2,3,5]	explanation, example, case studies, dialogue, debate	
12. Java Language Design [3,5]	exposure: description, explanation, example, case studies, dialogue, debate	
13. .NET Language Design [4,5,6]	exposure: description, explanation, example, case studies, dialogue, debate	
14. Final written exam	evaluation	

Bibliography

1. GRUNE, DICK - BAL, H. - JACOBS, C. - LANGENDOEN, K.: Modern Compiler Design, John Wiley, 2000
2. MITCHELL, JOHN: Foundations for Programming Languages, MIT Press, 1996
3. MOTOGNA, SIMONA: Metode de proiectare a compilatoarelor, Ed. Albastra, 2006
4. RICHTER, J.: Applied Microsoft .NET Framework Programming, Microsoft Press, 2002
5. LIDIN, SERGE: Inside .NET IL Assembler, Microsoft Press International, 2002
6. STUTZ, DAVID - NEWARD, TED - SHILLING, GEOFF: Shared Source CLI Essentials, O'Reilly UK, 2003
7. Sun Java Systems, [http://docs.sun.com/db/prod/java.sys], 01.09.2004

8.2 Seminar / laboratory	Teaching methods	Remarks
1. Task 1: Create an attribute grammar and write a program for attribute evaluation 1.1 define attribute grammar	Explation, dialogue, case studies	Professor will assigned a specific statement to be modelled with attribute grammars
2. Task 1: Create an attribute grammar and write a program for attribute evaluation 1.2 refine attribute grammar to satisfy evaluator restrictions	Explation, dialogue, case studies	
3. Task 1: Create an attribute grammar and write a program for attribute evaluation 1.3 program for attribute evaluation	Explation, dialogue, case studies	
4. Task 1: Create an attribute grammar and write a program for attribute evaluation 1.4 testing of the evaluator and deliver the program	Evaluation	
5. Task 2: Intermediary code generation 2.1: form of intermediary code; data staructure for intermediary code	Explation, dialogue, case studies	Professor will assigned a specific statement to be transformed to intermediary code
6. Task 2: Intermediary code generation 2.2: program for intermediary code generation	Explation, dialogue, case studies	
7. Task 2: Intermediary code generation 2.3:testing and delivery of the program	Evaluation	
8. Task 3: Apply optimization technique to a fragment of 3 address code 3.1 case study: chosen optimization technique	Explation, dialogue, case studies	Optimization will be applied for the result of task 2
9. Task 3: Apply optimization technique to a fragment of 3 address code 3. 2 implement optimization	Explation, dialogue, case studies	

10. Task 3: Apply optimization technique to a fragment of 3 address code 3.3 testing and delivery	Evaluation	
11. Task 4: Object code generation. Transform it to object code, using a minimum number of registers, determined based on the number of live variables. 4.1 Algorithm for determining the number of live variables and minimal number of registers	Explation, dialogue, case studies	Object code will be generated for output of task 3
12. Task 4: Object code generation 4.2 Implement object code generation	Explation, dialogue, case studies	
13. Task 4: Object code generation 4.3 testing and delivery	Evaluation	
14. Final laboratory: final presentation of tasks	Evaluation	
Bibliography Same as course & course notes		

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 advanced programming skills

10. Evaluation

Type of activity	10.1 Evaluation criteria	10.2 Evaluation methods	10.3 Share in the grade (%)
10.4 Course	- know the basic principle of the domain; - apply the course concepts - understand advanced topics in the field	Written exam	50%
10.5 Seminar/lab activities	- be able to implement course concepts and algorithms - apply techniques for different classes of programming languages	-Practical examination -documentation -portofolio -continous observations	50%
10.6 Minimum performance standards			
➢ At least grade 5 (from a scale of 1 to 10) at both written exam and laboratory work.			

Date ..... Signature of course coordinator Assoc.Prof.PhD. Simona MOTOGNA

Signature of seminar coordinator Assoc.Prof.PhD. Simona MOTOGNA

Date of approval .....

Signature of the head of department .....