Universitat Autònoma de Barcelona

2019/2020

Compilers

Code: 102782
ECTS Credits: 6

Degree	Type	Year	Semester
2502441 Computer Engineering	OB	3	2
2502441 Computer Engineering	OT	4	2

Contact

Name: Francisco Javier Sánchez Pujadas
Email: Javier.Sanchez.Pujadas@uab.cat

Use of Languages

Principal working language: catalan (cat)
Some groups entirely in English: No
Some groups entirely in Catalan: Yes
Some groups entirely in Spanish: No

Teachers

Jorge Bernal del Nozal

Prerequisites

It is advisable to have studied the subjects:

- Fundamentals of Computing
- Methodology of the Programming
- Programming Laboratory

Knowledge that may be useful for this subject is:

- Knowledge about imperative and object-oriented programming languages such as C / C ++ will facilitate the understanding of the practical examples given in theory.
- For the generating part of the code the knowledge about the assembler will be useful.
- The knowledge about recursivity of the Algorithm Analysis and Design course will help to understand the functioning of the syntactic analysis.

Objectives and Contextualisation

Knowledges:

- Introduce the fundamental concepts and structures of a compiler of imperative languages.
- Improve the use and understanding of compilers (efficiency, language limits, etc.), and programming languages.
- Practical implementation of simple parsers.
  Recognizers of configuration files with complex parameters.
  Flexibilization of dialogues with the user.
- Solve problems using syntactic and semantic techniques.
  Definition and implementation of grammars to solve problems.

Decision between implementing an entire compiler or using a tool to generate compilers based on the type of problem to be solved.

- Implement complex algorithms.
- Test strategies of complex programs.
- Implementation of recursive algorithms.
- Use of tools that generate code.
- Modularization in subproblems strongly coupled.
- Adaptation and use of a program already made.

Abilities:

- Improve the quality of the programs created thanks to the greater knowledge of the operation of the compilers.
- Correct use of the structures of the programming languages to obtain more optimal programs.
- Resolving complexity problems by applying the principles of modularity and recursion.
- Use of automatic programming tools.

Competences

Computer Engineering

- Acquire personal work habits.
- Acquire thinking habits.
- Have the capacity for the basic theory of programming languages and the associated techniques of lexical, syntactic and semantic processing, and know how to apply them to the creation, design and processing of languages.
- Have the capacity to conceive, develop and maintain computer systems, services and applications employing the methods of software engineering as an instrument to ensure quality.

Learning Outcomes

1. Adapt knowledge of lexical analysis , syntactic and semantic code generation in basic compiler .
2. Apply knowledge of lexical, syntactic and semantic analysis to the generation of code in a basic compiler.
3. Develop a mode of thought and critical reasoning.
4. Know and understand the processes of lexical, syntactic and semantic analysis of programming languages and analyse the different ways of using each of them.
5. Make ones own decisions.
6. Manage time and resources available. Work in an organized manner .
7. Prevent and solve problems.

Content

I. Introduction. Basics

1. Introduction to compilers.
2. Compiler construction strategies.
3. Definition of compiler and its parts.
4. Presentation of the following subjects of the course.

II. Lexicographical analysis: SCANNER.

- Definition of lexicographical analysis
- Regular expressions
- Implementation of a lexicographical analyzer (Scanner)
- Lexicographical analysis.

III. Grammar and syntactic analysis: PARSER.

- Definition of context-free grammars.
- Notations to define grammars.
- Characteristics of grammar and its associated Programming Languages.
- Grams LL (1).
- Parsers LL (1).
- Recovery of errors in Parsers LL (1).
- Parsers LR, SLR, LALR.
- Practical considerations

IV. Semantic analysis

- Syntax-directed translation.
- Table of Symbols.
- Semantic analysis of statements.
- Semantic analysis of expressions.

V. Organization of the memory at run time.

- Execution environment and static memory allocation.
- Allocation of the stack.
- Heap

VI. Code generation

- Abstract machine
- Generation of expressions code.
- Process of references to data structures.
- Generation of functions code.
- Generation of control structures code.

Methodology

This subject has a very strong link between what the theory is, an explanation of how compilers work and can be created, and practice or problems, as a compiler is actually created. During theory classes, concepts and techniques that are necessary to solve the practice are explained. The concepts of practice and problems are related, since the student will design a solution to solve a problem and implement it in practice. In this way, the student's learning process will be based on these three types of activities: theory classes, seminars or explanations of how to apply theoretical concepts to practice and practical classes or problems in which students will present and discuss their solutions

Unified class sessions

we do not differentiate between theory classes, problems and practices. In this way the session begins by explaining the theoretical concepts as in a theory class. Then these concepts are applied to problems or practices. Thus the session ends up as a class of problems or practices.

Theory classes

The theory classes will serve to introduce the concepts that are detailed in the syllabus of the subject. They will be based on the teacher's explanations, which will be assisted by complementary material in the form of photocopies or transparencies. The student must complete the theory classes with the hours of study, which should serve to finish understanding the contents of the subject and be able to prepare correctly the practice sessions. It must be borne in mind that the theory classes present a continuity throughout the course, so in order to correctly follow a class one must have assimilated what was explained in the previous classes.

Classes of problems

The classes of problems will be devoted basically to the detailed explanation of how to apply the theoretical concepts in a practical way, and will serve as a bridge between the theoretical classes and the practices.

Teachers will raise problems and indicate how they are resolved by preparing students to solve the practices. It is very important to have assimilated the previous theoretical concepts to be able to correctly follow the kinds of problems.

Before each session of problems the teacher will propose a list of exercises that the students should try to solve previously to be able to present doubts at the time of class. In some sessions, common problems arising in the practices will be prepared or discussed.

Practice

The practice will consist of to make modifications on the compiler explained to theory. The practice will be done in groups of 1, 2 or 3 people. The sessions will be divided into 3 blocks: Syntactic Analysis, Semantic Analysis and Code Generation. Each block will have one or more deliveries. The student will be able to test each delivery with a self-test consisting of the tests that the practice must pass to pass. After the teacher will pass a more complete test that will be the basis for obtaining the note of group practice. The practice can be tested with a web checker as many times as the student needs before the delivery date. In addition, it will be possible to evaluate the delivery, and the knowledge obtained by each student of the group with a small oral test or written by the individual note.

Cross-curricular competences: The cross-curricular competences will be worked out and evaluated at various moments throughout the course. Some examples of how they work are the following:

- T01.01 - Develop a way of thinking and critical reasoning:
  Theoretical classes explain examples of existing languages and compilers ofwhich the pros, cons and design errors are valued.
  In problemclasses, students must create an abstract model of the problem to be able to solve it.
  In practice, students must deduce what semantic restrictions the compiler must check.
- T02.03- Manage time andresources available. Work in an organized way:
  In practicum, the students must decide in which order to solve the various problems of the practice and what is their strategy to face them.
- T02.04 - Preventing and solving problems:
  In practice there are strong dependencies between the different stages of the compilation. Students must make decisions about what they are doing in the first stages to ensure that the following are not too complicated.
- T02.05 - Take own decisions:
  As for problems and practices, students must make decisions about how to solve the various problems that appear

Activities

Title	Hours	ECTS	Learning Outcomes
Type: Directed			
Seamless Ccasses (theory, problems and practices)	46	1.84	1, 2, 4, 3, 6, 5, 7
Type: Autonomous			
Preparation and implementation of the practice	47	1.88	1, 2, 4, 3, 6, 5, 7
Preparation and resolution of problems	12	0.48	1, 2, 4, 3, 6, 5, 7
Study of the subject taught in theoretical classes	36	1.44	1, 2, 4, 3

Assessment

Evaluation criteria and indicators:

- Understanding the theoretical concepts of the subject.
- Correct use of the main tools used to build a compiler.
- Application of the general principles of design in the resolution of complex problems.
- Try complex problems.

Activities and evaluation instruments:

Practical notes: In this section there is a group note and an individual:
Group note: Corresponds to the grade obtained for group deliveries of the practice.
Individual note: This is the score obtained in the practice exam.
Note Theory: corresponds to the final exam or partial exam / s of theory on the subject. It has been approved for the partial examinations in order to eliminate the final exam.
The final grade of the subject is obtained by combining the evaluation of these two activities in the following way:

Final Note = (0.5 * Practical Notes) + (0.5 * Note Theory)

Practical Notes = (0.2 * Individual Note) + (0.8 * Note Group)
Individual Note = Examination of practices.
Note Group = (0.2 * Group A. Syntactic) + (0.3 * Semantic Group A.) + (0.5 * Generation Group Code)

The group notes for syntactic, semantic, and code generation can correspond to a weighted average of several deliveries.

Validation of internships:

Practices of previous years are not validated.

Recovery of practices:

In the case of having suspended a group delivery, it will be possible to recover in the following deliveries of the practice. The note will be 0.75 * (max (note recovery, delivery note suspended) -note delivery suspended) + note delivery suspended. In the case of suspending the practice exam, the student must submit to a practice recovery exam on the day of the final exam.

Conditions to pass the subject:

Final Note> = 5
Practical note> = 5 and have approved each of the individual and group deliveries.
Theory note> = 5 and have approved separately the parts that have the theory exam.
Conditions for the non-evaluable:

Do not have any part of the subject suspended.

Conditions for subject failure:

Do not reach an average grade higher than or equal to 5.
failing to pass some of the evaluation activities of the subject although the average exceeds 5. In this case, the note will be the minimum score obtained from one of the parties (exams or practices).

Conditions for honor grade:

The matrícula of honor can be obtained with a note average superior or equal to 9.0.
Due to the fact that there is a limited number of matriculations of honor that can be given by group, they will be awarded in order of note from major to minor.

Practices, papers or exams copied:

Notwithstanding other disciplinary measures deemed appropriate, and in accordance with the current academic regulations, irregularities committed by a student that can lead to a variation of the qualification will be classified by zero (0). Assessment activities qualified in this way and by this procedure will not be

recoverable. If it is necessary to pass any of these assessment activities to pass the subject, this subject will be suspended directly, without opportunity to recover it in the same course. These irregularities include, among others:
- the total or partial copy of a practice, report, or any other evaluation activity;
- let copy;
- present a group work not done entirely by the members of the group;
- present as own materials prepared by a third party, although they are translations or adaptations, and generally works with non-original and exclusive elements of the student;
- Have communication devices (such as mobile phones, smart watches, etc.) accessible during theoretical-practical tests (individual exams).

In case the student has committed irregularities in an evaluation act, the numerical note of the file will be the lowest value between 3.0 and the corresponding grade according to the method of evaluation of the subject (and therefore not the approved one will be possible for compensation).

In summary: copying, copying, or plagiarizing in any of the assessment activities is equivalent to a SUSPENS with a score of less than 3.0.

Publication notes, dates of exams, etc.:

The dates of continuous evaluation and delivery of works will be published on the virtual campus and may be subject to changes of programming for reasons of adaptation to possible incidents. The virtual campus will always be informed about these changes as it is understood to be the usual mechanism for exchanging information between teacher and students.

Procedure for the review of qualifications

For each evaluation activity, a place, date and revision will be indicated in which the student can review the activity with the teacher. In this context, claims can be made about the activity note, which will be evaluated by the teachers responsible for the subject. If the student does not submit to this review, this activity will not be reviewed later.

Assessment Activities

Title	Weighting	Hours	ECTS	Learning Outcomes
Final theory exam	See activities and evaluation tools	3	0.12	1, 2, 4, 3
Generation of practice code	See activities and evaluation tools	1	0.04	2, 3, 6, 5, 7
Itermediate theory exam/s	See activities and evaluation tools	2	0.08	1, 2, 4, 3
Practicum exam	See activities and evaluation tools	1	0.04	1, 2, 3
Semantic analysis of the practice	See activities and evaluation tools	1	0.04	1, 4, 3, 6, 5, 7
Syntactic analysis of the practice	See activities and evaluation tools	1	0.04	1, 4, 3, 6, 5, 7

Bibliography

Basic bibliography

- *Charles N. Fischer & Richard J. Leblanc jr.*: Crafting a Compiler with C. The Benjamin / Cummings Publishing Company inc., 1991.
- *Jean-Paul Tremblay & Paul G. Sorenson*: The Theory and Practice of Compiler Writing. MCGRAW-HILL, 1985.
- *David A. Watt*: Programming Language Concepts and Paradigms. Prentice Hall, 1990.
- Alicia Garrido & all: Diseño de compiladores. Publicaciones Universidad de Alicante, 2002.

Reference bibliography

- *David A. Watt.*: Programming Language Processors. Prentice Hall, 1993.
- *Anthony J. Field & Peter G. Harrison*: Functional Programming. Addison-Wesley, 1988.
- *Terrence W. Pratt*: Programming Languages: Design and Implementation. Prentice Hall International Editions, 1984.
- *Edited by G. Goos and J. Hartmanis*: Compiler Construction. An Advanced Course. Springer-Verlag, 1974
- *Jean-Paul Tremblay & Paul G. Sorenson*: An Implementation Guide to Compiler Writing. MCGRAW-HILL, 1982.
- *Ronald Mak*: Writing Compilers & Interpreters: An Applied Approach. Wiley, 1991.
- *J. P. Brown*: Writing Interactive Compilers and Interpreters. Wiley, 1979.
- *Allen Y. Holub*: Compiler Design in C. Prentice Hall, 1990.