



National University of Engineering (UNI)
School of Computer Science
Syllabus 2027-I

1. COURSE

CS342. Compilers (Mandatory)

2. GENERAL INFORMATION

2.1 Course	: CS342. Compilers
2.2 Semester	: 6 th Semester.
2.3 Credits	: 4
2.4 Hours	: 2 HT; 4 HP;
2.5 Duration of the period	: 16 weeks
2.6 Type of course	: Mandatory
2.7 Learning modality	: Face to face
2.8 Prerequisites	: CS211. Theory of Computation. (4 th Sem)

3. PROFESSORS

Meetings after coordination with the professor

4. INTRODUCTION TO THE COURSE

This course provides a deep understanding of how high-level languages are transformed into executable machine code. The study of compilers allows students to understand language semantics, memory management, and code optimization, which are fundamental skills for designing domain-specific languages and high-performance software engineering.

5. GOALS

- Understand the phases of a compiler and construction tools.
- Implement lexical and syntactic analyzers based on grammars.
- Generate and optimize intermediate code for a specific architecture.

6. COMPETENCES

AG-C08) Problem Analysis: Identifies, formulates, and analyzes complex computing problems. (Usage)

AG-C12) Uses systemic approaches to select, develop, apply, integrate, and manage secure computing technologies to achieve user objectives. (Usage)

7. TOPICS

Unit 1: Program Analysis and Analyzers (20 hours)	
Competences Expected: AG-C08,AG-C12	
Topics	Learning Outcomes
• Relevant program representations, such as basic blocks, control-flow graphs, def-use chains, and static single assignment • Undecidability and consequences for program analysis • Flow-insensitive analysis, such as type-checking and scalable pointer and alias analysis • Flow-sensitive analysis, such as forward and backward dataflow analyses • Path-sensitive analysis, such as software model checking and software verification • Tools and frameworks for implementing analyzers • Role of static analysis in program optimization and data dependency analysis during exploitation of concurrency • Role of program analysis in (partial) verification and bug-finding • Parallelization: – Analysis for auto-parallelization – Analysis for detecting concurrency bugs	• Explain the difference between dataflow graph and control flow graph [Explain] • Explain why non-trivial sound program analyses must be approximate [Explain] • Argue why an analysis is correct (sound and terminating) [Argue] • Explain why potential aliasing limits sound program analysis and how alias analysis can help [Explain] • Use the results of a program analysis for program optimization and/or partial program correctness [Use]
Readings : [Aho2006], [Appel2004]	

Unit 2: Language Translation and Execution (24 hours)	
Competences Expected: AG-C08,AG-C12	
Topics	Learning Outcomes
• Execution models for JIT (Just-In-Time), compiler, interpreter • Use of intermediate code, e.g., bytecode • Limitations and benefits of JIT, compiler, and interpreter • Cross compilers/transpilers • BNF and extended BNF representation of context-free grammar • Parse tree using a simple sentence such as arithmetic expression or if-then-else statement • Execution as native code or within a virtual machine • Language translation pipeline: syntax analysis, parsing, optional type-checking, translation/code generation and optimization, linking, loading, execution • Run-time representation of core language constructs such as objects (method tables) and functions that can be passed as parameters to and returned from functions (closures) • Secure compiler development	• Explain and understand the differences between compiled, JIT, and interpreted language implementations, including the benefits and limitations of each [Explain] • Differentiate syntax and parsing from semantics and evaluation [Differentiate] • Use BNF and extended BNF to specify the syntax of simple constructs such as if-then-else, type declaration and iterative constructs for known languages such as C++ or Python [Use] • Illustrate the parse tree using a simple sentence/arithmetic expression [Apply] • Illustrate translation of syntax diagrams to BNF/extended BNF for simple constructs such as if-then-else, type declaration, iterative constructs, etc [Apply] • Illustrate ambiguity in parsing using nested if-then-else/arithmetic expression and show resolution using precedence order [Apply] • Discuss the benefits and limitations of garbage collection, including the notion of reachability [Debate]
Readings : [Aho2006], [Appel2004]	

Unit 3: Run-time Behavior and Systems (20 hours)	
Competences Expected: AG-C08,AG-C12	
Topics	Learning Outcomes
• Process models using stacks and heaps to allocate and deallocate activation records and recovering environments using frame pointers and return addresses during a procedure call including parameter-passing examples • Schematics of code lookup using hash tables for methods in implementations of object-oriented programs • Data layout for objects and activation records • Object allocation in heap • Implementing virtual entities and virtual methods; virtual method tables and their application • Run-time behavior of object-oriented programs • Compare and contrast allocation of memory during information exchange using parameter-passing and non-local variables (using chain of static links). • Dynamic memory management approaches and techniques: malloc/free, garbage collection (mark-sweep, copying, reference counting), regions (also known as arenas or zones) • Just-in-time compilation and dynamic recompilation • Interface to operating system (e.g., for program initialization) • Interoperability between programming languages including parameter-passing mechanisms and data representation: – Big endian, little endian – Data layout of composite data types such as arrays • Other common features of virtual machines, such as class loading, threads, and security checking • Sandboxing	• Discuss benefits and limitations of automatic memory management [Debate] • Explain the use of metadata in run-time representations of objects and activation records, such as class pointers, array lengths, return addresses, and frame pointers [Explain] • Compare and contrast static allocation vs stack-based allocation vs heap-based allocation of data elements [Compare] • Explain why some data elements cannot be automatically deallocated at the end of a procedure/method call (need for garbage collection) [Explain] • Discuss advantages, disadvantages, and difficulties of just-in-time and dynamic recompilation [Debate] • Discuss the use of sandboxing in mobile code [Debate] • Identify the services provided by modern language run-time systems [Analyze]
Readings : [Aho2006], [Cooper2011]	

8. WORKPLAN

8.1 Methodology

Individual and team participation is encouraged to present their ideas, motivating them with additional points in the different stages of the course evaluation.

8.2 Theory Sessions

The theory sessions are held in master classes with activities including active learning and roleplay to allow students to internalize the concepts.

8.3 Practical Sessions

The practical sessions are held in class where a series of exercises and/or practical concepts are developed through problem solving, problem solving, specific exercises and/or in application contexts.

9. EVALUATION SYSTEM

Theory Sessions:

The theory sessions are held in master classes with activities including active learning and roleplay to allow students to internalize the concepts.

Lab Sessions:

In order to verify their competences, several activities including active learning and roleplay will be developed during lab sessions.

Oral Presentations:

Individual and team participation is encouraged to present their ideas, motivating them with additional points in the different stages of the course evaluation.

Reading:

Throughout the course different readings are provided, which are evaluated. The average of the notes in the readings is considered as the mark of a qualified practice. The use of our virtual campus allows each student to access the course information, and interact outside the classroom with the teacher and with the other students.

The final grade NF depends on the following grades:

- PE : Partial Exam.
- FE : Final Exam.
- P : Permanent Evaluation (Average of practical work and graded assignments).

The final grade NF is calculated as follows:

$$NF = \frac{PE + FE + P}{3}$$

10. BASIC BIBLIOGRAPHY