UCLouvain

linfo2132

2026

Languages and translators

6.00 credits	30.0 h + 30.0 h	Q2

Teacher(s)	Sadre Ramin ;
Language :	English > French-friendly
Place of the course	Louvain-la-Neuve
Prerequisites	Required#: concepts, paradigms, and semantics of programming languages as targeted in course LINFO1104 Required#: advanced notions of algorithms and object-oriented programming as covered in course LEPL1402 Desirable#: principles of computer systems, as targeted in course LINFO1252
Main themes	· Methods to analyze context-free languages, upstream and downstream methods · Generators of lexical analyzers and parsers · Statistical semantics and attributed grammars · Methods to translate a source code in a target code, and generation of target code · Machine virtuelle et byte-code (JVM) · Garbage Collection et gestion mémoire · Domain Specific Languages (DSL)
Learning outcomes	**At the end of this learning unit, the student is able to :** Given the learning outcomes of the "Master in Computer Science and Engineering" program, this course contributes to the development, acquisition and evaluation of the following learning outcomes: · INFO1.1-3 · INFO2.2-4 · INFO5.4, INFO5.5 · INFO6.4 Given the learning outcomes of the "Master [120] in Computer Science" program, this course contributes to the development, acquisition and evaluation of the following learning outcomes: · SINF1.M2 · SINF2.2-4, SINF2.5 · SINF5.4, SINF5.5 · SINF6.4 Given the learning outcomes of the "Master [60] in Computer Science" program, this course contributes to the development, acquisition and evaluation of the following learning outcomes: · 1SINF1.M2 · 1SINF2.2-4, 1SINF2.5 · 1SINF5.4, 1SINF5.5 · 1SINF6.4 Students completing successfully this course will be able to · explain in a practical way the structure of compilers dealing with algorithmic languages; · design and implement a compiler for a practical language which solves a interesting problem; · show the interest of compiling techniques in problem resolving. Students will have developed skills and operational methodology. In particular, they have developed their ability to · treat rigorously a problem, justifying and validating each step of a project to be able to rely on it to implement the following one; · explain in practical terms how a source code (Java) is finally translated into byte-code; · explain the enforcement mechanisms byte code by JVM; · explain memory management during the execution of a program; · explain how garbage collection mechanisms.

Evaluation methods	June session: The evaluation consists of two components: The project (done in groups) accounts for 60% of the course's final grade. A written exam accounts for 40%. August session: If the student did not successfully pass the course in the first session (i.e., they did not obtained at least 10/20 for the final grade), the student is allowed to redo those components (project or exam or both) of the evaluation for which they did not obtain at least 50% of the respective points. They will keep the points for the component that they passed (if any). The same weights as in the June session are applied for the calculation of the final grade. Regarding the use of generative AI: Students are permitted to use generative AI tools to support their project work, e.g., to compare solution approaches, find errors in code, or improve the wording in reports. However, submissions, whether source code, reports, or other forms, must essentially be the students' own work. If generative AI tools have been used, their nature and scope must be described in detail in the submission (see the respective project instructions) and only the original part of the student's work will be evaluated. This also applies when drawing on the work of others, whether from the Internet or from another student. Failure to comply with these guidelines may result in reduced grades or other academic sanctions. The teacher may request a student to go through an additional oral exam as a complement of the exam and/or of the project activities, in cases including, but not limited to, technical issues, or suspicion of irregularities.
Teaching methods	The course consists in a series of pre-recorded video lectures, lecture and Q&A sessions, and lab sessions. A project will take place with several deadlines distributed over the quadrimester.
Content	The course presents the theory and practice of programming language implementation, as well as compiler architecture. We will review the standard components of a compiler, from front-end (parsing, lexical analysis) to back-end (code generation). Among others, we will see: · Scanning/lexing and regular expressions · Parsing and context free grammars · Type checking · Register allocation and code generation During the course, the students will implement a compiler for a new programming language.
Inline resources	Teams and/or Moodle
Other infos	Students should have solid knowledge of programming in a compiled object-oriented language, of algorithms and data structures, and of the basics of computer architectures, as taught for example in the following courses: · LEPL1402 and/or LEPL1509: Programming of larger programs in Java · LINFO1252 and/or LINFO2241: Computer systems/architecture · LINFO1121: Algorithms and data structures In addition, formal notations will be used to describe the properties of programming languages.
Faculty or entity in charge	INFO

Programmes containing this learning unit (UE)

Program title	Acronym	Credits	Prerequisite	Learning outcomes
Master [120] in Computer Science and Engineering	INFO2M	6		
Master [120] in Computer Science	SINF2M	6		
Master [60] in Computer Science	SINF2M1	6		