San Pablo Catholic University (UCSP)
Undergraduate Program in Computer Science
SILABO

CS342. Compilers (Mandatory)

2020-I

1. General information

1.1 School	:	Ciencia de la Computación
1.2 Course	:	CS342. Compilers
1.3 Semester	:	8^{vo} Semestre.
1.4 Prerrequisites	:	CS341. Programming languages . (7^{th} Sem)
1.5 Type of course	:	Mandatory
1.6 Learning modality	:	Virtual
1.7 Horas	:	2 HT; 2 HP; 2 HL;
1.8 Credits	:	4

2. Professors

Lecturer

- Gina Lucia Muñoz Salas <glmunoz@ucsp.edu.pe>
  - MSc in Ciencia de la Computación, Universidad Católica San Pablo, Perú, 2019.

Practice

- Carlos Eduardo Atencio Torres <ceatencio@ucsp.edu.pe>
  - MSc in Ciencia de la Computación, USP, Brasil, 2014.

3. Course foundation

That the student knows and understands the concepts and fundamental principles of the theory of compilation to realize the construction of a compiler

4. Summary

1. Program Representation 2. Language Translation and Execution 3. Syntax Analysis 4. Compiler Semantic Analysis 5. Code Generation

5. Generales Goals

- Know the basic techniques used during the process of intermediate generation, optimization and code generation.
- Learning to implement small compilers.

6. Contribution to Outcomes

This discipline contributes to the achievement of the following outcomes:

a) An ability to apply knowledge of mathematics, science. (**Assessment**)

b) An ability to design and conduct experiments, as well as to analyze and interpret data. (**Assessment**)

j) Apply the mathematical basis, principles of algorithms and the theory of Computer Science in the modeling and design of computational systems in such a way as to demonstrate understanding of the equilibrium points involved in the chosen option. (**Assessment**)

7. Content

UNIT 1: Program Representation (5)	
Competences: a,b	
Content	**Generales Goals**
• Programs that take (other) programs as input such as interpreters, compilers, type-checkers, documentation generators • Abstract syntax trees; contrast with concrete syntax • Data structures to represent code for execution, translation, or transmission • Just-in-time compilation and dynamic recompilation • Other common features of virtual machines, such as class loading, threads, and security.	• Explain how programs that process other programs treat the other programs as their input data [Familiarity] • Describe an abstract syntax tree for a small language [Familiarity] • Describe the benefits of having program representations other than strings of source code [Familiarity] • Write a program to process some representation of code for some purpose, such as an interpreter, an expression optimizer, or a documentation generator [Familiarity] • 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 [Familiarity] • Discuss advantages, disadvantages, and difficulties of just-in-time and dynamic recompilation [Familiarity] • Identify the services provided by modern language run-time systems [Familiarity]
Readings: Louden (2004b)	

UNIT 2: Language Translation and Execution (10)	
Competences: a,b,j	
Content	**Generales Goals**
• Interpretation vs. compilation to native code vs. compilation to portable intermediate representation • Language translation pipeline: parsing, optional type-checking, translation, linking, execution – Execution as native code or within a virtual machine – Alternatives like dynamic loading and dynamic (or “just-in-time”) code generation • Run-time representation of core language constructs such as objects (method tables) and first-class functions (closures) • Run-time layout of memory: call-stack, heap, static data – Implementing loops, recursion, and tail calls • Memory management – Manual memory management: allocating, de-allocating, and reusing heap memory – Automated memory management: garbage collection as an automated technique using the notion of reachability	• Distinguish a language definition (what constructs mean) from a particular language implementation (compiler vs interpreter, run-time representation of data objects, etc) [Assessment] • Distinguish syntax and parsing from semantics and evaluation [Assessment] • Sketch a low-level run-time representation of core language constructs, such as objects or closures [Assessment] • Explain how programming language implementations typically organize memory into global data, text, heap, and stack sections and how features such as recursion and memory management map to this memory model [Assessment] • Identify and fix memory leaks and dangling-pointer dereferences [Assessment] • Discuss the benefits and limitations of garbage collection, including the notion of reachability [Assessment]
Readings: Aho et al. (2011), Louden (2004a), Appel (2002), Teufel and Schmidt (1998)	

UNIT 3: Syntax Analysis (10)	
Competences: a,b,j	
Content	**Generales Goals**
• Scanning (lexical analysis) using regular expressions • Parsing strategies including top-down (e.g., recursive descent, Earley parsing, or LL) and bottom-up (e.g., backtracking or LR) techniques; role of context-free grammars • Generating scanners and parsers from declarative specifications	• Use formal grammars to specify the syntax of languages [Assessment] • Use declarative tools to generate parsers and scanners [Assessment] • Identify key issues in syntax definitions: ambiguity, associativity, precedence [Assessment]
Readings: Aho et al. (2011), Louden (2004a), Appel (2002), Teufel and Schmidt (1998)	

UNIT 4: Compiler Semantic Analysis (15)	
Competences: a,b,j	
Content	**Generales Goals**
• High-level program representations such as abstract syntax trees • Scope and binding resolution • Type checking • Declarative specifications such as attribute grammars	• Implement context-sensitive, source-level static analyses such as type-checkers or resolving identifiers to identify their binding occurrences [Assessment] • Describe semantic analyses using an attribute grammar [Assessment]
Readings: Aho et al. (2011), Louden (2004a), Appel (2002), Teufel and Schmidt (1998)	

UNIT 5: Code Generation (20)	
Competences: a,b,j	
Content	**Generales Goals**
• Procedure calls and method dispatching • Separate compilation; linking • Instruction selection • Instruction scheduling • Register allocation • Peephole optimization	• Identify all essential steps for automatically converting source code into assembly or other low-level languages [Assessment] • Generate the low-level code for calling functions/methods in modern languages [Assessment] • Discuss why separate compilation requires uniform calling conventions [Assessment] • Discuss why separate compilation limits optimization because of unknown effects of calls [Assessment] • Discuss opportunities for optimization introduced by naive translation and approaches for achieving optimization, such as instruction selection, instruction scheduling, register allocation, and peephole optimization [Assessment]
Readings: Aho et al. (2011), Louden (2004a), Appel (2002), Teufel and Schmidt (1998)	

8. Methodology

El profesor del curso presentará clases teóricas de los temas señalados en el programa propiciando la intervención de los alumnos.

El profesor del curso presentará demostraciones para fundamentar clases teóricas.

El profesor y los alumnos realizarán prácticas

Los alumnos deberán asistir a clase habiendo leído lo que el profesor va a presentar. De esta manera se facilitará la comprensión y los estudiantes estarán en mejores condiciones de hacer consultas en clase.

9. Assessment

Continuous Assessment 1 : 20 %

Partial Exam : 30 %

Continuous Assessment 2 : 20 %

Final exam : 30 %

References

Aho, Alfred et al. (2011). *Compilers Principles Techniques And Tools*. 2nd. ISBN:10-970-26-1133-4. Pearson.

Appel, A. W. (2002). *Modern compiler implementation in Java*. 2.a edición. Cambridge University Press.

Louden, Kenneth C. (2004a). *Compiler Construction: Principles and Practice*. Thomson.

Louden, Kenneth C. (2004b). *Lenguajes de Programacion*. Thomson.

Teufel, Bernard and Stephanie Schmidt (1998). *Fundamentos de Compiladores*. Addison Wesley Iberoamericana.