SPC

SOCIEDAD
PERUANA DE
COMPUTACIÓN

Peruvian Computing Society (SPC)
School of Computer Science
Sillabus 2021-I

1. COURSE

CS342. Compilers (Mandatory)

2. GENERAL INFORMATION

2.1 Credits	:	4
2.2 Theory Hours	:	2 (Weekly)
2.3 Practice Hours	:	2 (Weekly)
2.4 Duration of the period	:	16 weeks
2.5 Type of course	:	Mandatory
2.6 Modality	:	Face to face
2.7 Prerrequisites	:	CS211. Theory of Computation. (4^{th} Sem)

3. PROFESSORS

Meetings after coordination with the professor

4. INTRODUCTION TO THE COURSE

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

5. GOALS

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

6. COMPETENCES

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. SPECIFIC COMPETENCES

a2) Use logical propositions in an orderly manner. ()

a4) Apply efficient techniques for solving computer problems. ()

a6) Apply finite-state machine and automaton techniques in the resolution of computer problems. ()

a7) Apply techniques and knowledge of computer architecture for the generation and optimization of code. ()

b1) Identify and efficiently apply various algorithmic strategies and data structures for the solution of a problem given certain space and time constraints. ()

j2) Apply graph and tree theory for optimization and problem solving ()

8. TOPICS

Unit 1: Program Representation (5)	
Competences Expected: a,b	
Topics	**Learning Outcomes**
• 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 : [Lou04b]	

<table>
<tr><td colspan="2">Unit 2: Language Translation and Execution (10)</td></tr>
<tr><td colspan="2">Competences Expected: a,b,j</td></tr>
<tr><td>Topics</td><td>Learning Outcomes</td></tr>
<tr><td>

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, deallocating, and reusing heap memory
Automated memory management: garbage collection as an automated technique using the notion of reachability

</td><td>

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]

</td></tr>
<tr><td colspan="2">Readings : [Aho+11], [Lou04a], [App02], [TS98]</td></tr>
</table>

<table>
<tr><td colspan="2">Unit 3: Syntax Analysis (10)</td></tr>
<tr><td colspan="2">Competences Expected: a,b,j</td></tr>
<tr><td>Topics</td><td>Learning Outcomes</td></tr>
<tr><td>

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

</td><td>

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]

</td></tr>
<tr><td colspan="2">Readings : [Aho+11], [Lou04a], [App02], [TS98]</td></tr>
</table>

Unit 4: Compiler Semantic Analysis (15)	
Competences Expected: a,b,j	
Topics	**Learning Outcomes**
• 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+11], [Lou04a], [App02], [TS98]	

Unit 5: Code Generation (20)	
Competences Expected: a,b,j	
Topics	**Learning Outcomes**
• 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+11], [Lou04a], [App02], [TS98]	

9. WORKPLAN

9.1 Methodology

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

9.2 Theory Sessions

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

9.3 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.

10. EVALUATION SYSTEM

********* EVALUATION MISSING ********

11. BASIC BIBLIOGRAPHY

[Aho+11] Alfred Aho et al. *Compilers Principles Techniques And Tools*. 2nd. ISBN:10-970-26-1133-4. Pearson, 2011.

[App02] A. W. Appel. *Modern compiler implementation in Java*. 2.a edición. Cambridge University Press, 2002.

[Lou04a] Kenneth C. Louden. *Compiler Construction: Principles and Practice*. Thomson, 2004.

[Lou04b] Kenneth C. Louden. *Lenguajes de Programacion*. Thomson, 2004.

[TS98] Bernard Teufel and Stephanie Schmidt. *Fundamentos de Compiladores*. Addison Wesley Iberoamericana, 1998.