

**Universidad Católica San Pablo (UCSP)**  
**Escuela Profesional de**  
**Ciencia de la Computación**  
**SILABO**



**CS342. Compiladores (Obligatorio)**

**1. Información general**

1.1 Escuela	:	Ciencia de la Computación
1.2 Curso	:	CS342. Compiladores
1.3 Semestre	:	8 <sup>vo</sup> Semestre.
1.4 Prerrequisitos	:	CS341. Lenguajes de Programación. (7 <sup>mo</sup> Sem)
1.5 Condición	:	Obligatorio
1.6 Modalidad de aprendizaje	:	Virtual
1.7 horas	:	2 HT; 2 HP; 2 HL;
1.8 Créditos	:	4

**2. Profesores**

**3. Fundamentación del curso**

Que el alumno conozca y comprenda los conceptos y principios fundamentales de la teoría de compilación para realizar la construcción de un compilador

**4. Resumen**

1. Representación de programas 2. Traducción y ejecución de lenguajes 3. Análisis de sintaxis 4. Análisis semántico de compiladores 5. Generación de código

**5. Objetivos Generales**

- Conocer las técnicas básicas empleadas durante el proceso de generación intermedio, optimización y generación de código.
- Aprender a implementar pequeños compiladores.

**6. Contribución a los resultados (*Outcomes*)**

Esta disciplina contribuye al logro de los siguientes resultados de la carrera:

- a) Aplicar conocimientos de computación y de matemáticas apropiadas para la disciplina. (**Evaluar**)
- b) Analizar problemas e identificar y definir los requerimientos computacionales apropiados para su solución. (**Evaluar**)
- j) Aplicar la base matemática, principios de algoritmos y la teoría de la CS en el modelamiento y diseño de sistemas. (**Evaluar**)

**7. Contenido**

UNIDAD 1: Representación de programas (5)	
Competencias: a,b	
Contenido	Objetivos Generales
<ul style="list-style-type: none"> <li>• Programas que tienen otros programas como entrada tales como interpretes, compiladores, revisores de tipos y generadores de documentación.</li> <li>• Árboles de sintaxis abstracta, para contrastar la sintaxis correcta.</li> <li>• Estructuras de datos que representan código para ejecución, traducción o transmisión.</li> <li>• Compilación en tiempo just-in time y re-compilación dinámica.</li> <li>• Otras características comunes de las máquinas virtuales, tales como carga de clases, hilos y seguridad.</li> </ul>	<ul style="list-style-type: none"> <li>• Explicar como programas que procesan otros programas tratan a los otros programas como su entrada de datos [Familiarizarse]</li> <li>• Describir un árbol de sintaxis abstracto para un lenguaje pequeño [Familiarizarse]</li> <li>• Describir los beneficios de tener representaciones de programas que no sean cadenas de código fuente [Familiarizarse]</li> <li>• Escribir un programa para procesar alguna representación de código para algún propósito, tales como un interprete, una expresión optimizada, o un generador de documentación [Familiarizarse]</li> <li>• Explicar el uso de metadatos en las representaciones de tiempo de ejecución de objetos y registros de activación, tales como los punteros de la clase, las longitudes de arreglos, direcciones de retorno, y punteros de <i>frame</i> [Familiarizarse]</li> <li>• Discutir las ventajas, desventajas y dificultades del término (<i>just-in-time</i>) y recompilación automática [Familiarizarse]</li> <li>• Identificar los servicios proporcionados por los sistemas de tiempo de ejecución en lenguajes modernos [Familiarizarse]</li> </ul>
<b>Lecturas:</b> Louden (2004b)	

UNIDAD 2: Traducción y ejecución de lenguajes (10)	
Competencias: a,b,j	
Contenido	Objetivos Generales
<ul style="list-style-type: none"> <li>• Interpretación vs. compilación a código nativo vs. compilación de representación portable intermedia.</li> <li>• Pipeline de traducción de lenguajes: análisis, revisión opcional de tipos, traducción, enlazamiento, ejecución: <ul style="list-style-type: none"> <li>– Ejecución como código nativo o con una máquina virtual</li> <li>– Alternativas como carga dinámica y codificación dinámica de código (o “just-in-time”)</li> </ul> </li> <li>• Representación en tiempo de ejecución de construcción del lenguaje núcleo tales como objetos (tablas de métodos) y funciones de primera clase (cerradas)</li> <li>• Ejecución en tiempo real de asignación de memoria: pila de llamadas, montículo, datos estáticos: <ul style="list-style-type: none"> <li>– Implementación de bucles, recursividad y llamadas de cola</li> </ul> </li> <li>• Gestión de memoria: <ul style="list-style-type: none"> <li>– Gestión manual de memoria: asignación, limpieza y reuso de la pila de memoria</li> <li>– Gestión automática de memoria: recolección de datos no utilizados (<i>garbage collection</i>) como una técnica automática usando la noción de accesibilidad</li> </ul> </li> </ul>	<ul style="list-style-type: none"> <li>• Distinguir una definición de un lenguaje de una implementación particular de un lenguaje (compilador vs interprete, tiempo de ejecución de la representación de los objetos de datos, etc) [Evaluar]</li> <li>• Distinguir sintaxis y parseo de la semántica y la evaluación [Evaluar]</li> <li>• Bosqueje una representación de bajo nivel de tiempo de ejecución de construcciones del lenguaje base, tales como objetos o cierres (<i>closures</i>) [Evaluar]</li> <li>• Explicar cómo las implementaciones de los lenguajes de programación típicamente organizan la memoria en datos globales, texto, <i>heap</i>, y secciones de pila y cómo las características tales como recursión y administración de memoria son mapeados a este modelo de memoria [Evaluar]</li> <li>• Identificar y corregir las pérdidas de memoria y punteros desreferenciados [Evaluar]</li> <li>• Discutir los beneficios y limitaciones de la recolección de basura (<i>garbage collection</i>), incluyendo la noción de accesibilidad [Evaluar]</li> </ul>
<b>Lecturas:</b> Aho et al. (2011), Louden (2004a), Teufel and Schmidt (1998), Appel (2002)	

UNIDAD 3: Análisis de sintaxis (10)	
Competencias: a,b,j	
Contenido	Objetivos Generales
<ul style="list-style-type: none"> <li>• Exploración (análisis léxico) usando expresiones regulares.</li> <li>• Estrategias de análisis incluyendo técnicas de arriba a abajo (top-down) (p.e. descenso recursivo, análisis temprano o LL) y de abajo a arriba (bottom-up) (ej, ‘llamadas hacia atrás - bracktracking, o LR); rol de las gramáticas libres de contexto.</li> <li>• Generación de exploradores (scanners) y analizadores a partir de especificaciones declarativas.</li> </ul>	<ul style="list-style-type: none"> <li>• Usar gramáticas formales para especificar la sintaxis de los lenguajes [Evaluar]</li> <li>• Usar herramientas declarativas para generar parseadores y escáneres [Evaluar]</li> <li>• Identificar las características clave en las definiciones de sintaxis: ambigüedad, asociatividad, precedencia [Evaluar]</li> </ul>
<b>Lecturas:</b> Aho et al. (2011), Louden (2004a), Teufel and Schmidt (1998), Appel (2002)	

UNIDAD 4: Análisis semántico de compiladores (15)	
Competencias: a,b,j	
Contenido	Objetivos Generales
<ul style="list-style-type: none"> <li>• Representaciones de programas de alto nivel tales como árboles de sintaxis abstractas.</li> <li>• Alcance y resolución de vínculos.</li> <li>• Revisión de tipos.</li> <li>• Especificaciones declarativas tales como gramáticas atribuidas.</li> </ul>	<ul style="list-style-type: none"> <li>• Implementar analizadores sensibles al contexto y estáticos a nivel de fuente, tales como, verificadores de tipos o resolvedores de identificadores para identificar las ocurrencias de vinculo [Evaluar]</li> <li>• Describir analizadores semanticos usando una gramatica con atributos [Evaluar]</li> </ul>
<b>Lecturas:</b> Aho et al. (2011), Louden (2004a), Teufel and Schmidt (1998), Appel (2002)	

UNIDAD 5: Generación de código (20)	
Competencias: a,b,j	
Contenido	Objetivos Generales
<ul style="list-style-type: none"> <li>• Llamadas a procedimientos y métodos en envío.</li> <li>• Compilación separada; vinculación.</li> <li>• Selección de instrucciones.</li> <li>• Calendarización de instrucciones.</li> <li>• Asignación de registros.</li> <li>• Optimización por rendija (peephole)</li> </ul>	<ul style="list-style-type: none"> <li>• Identificar todos los pasos esenciales para convertir automáticamente código fuente en código ensamblador o otros lenguajes de bajo nivel [Evaluar]</li> <li>• Generar código de bajo nivel para llamadas a funciones en lenguajes modernos [Evaluar]</li> <li>• Discutir por qué la compilación separada requiere convenciones de llamadas uniformes [Evaluar]</li> <li>• Discutir por qué la compilación separada limita la optimización debido a efectos de llamadas desconocidas [Evaluar]</li> <li>• Discutir oportunidades para optimización introducida por la traducción y enfoques para alcanzar la optimización, tales como la selección de la instrucción, planificación de instrucción, asignación de registros y optimización de tipo mirilla (<i>peephole optimization</i>) [Evaluar]</li> </ul>
<b>Lecturas:</b> Aho et al. (2011), Louden (2004a), Teufel and Schmidt (1998), Appel (2002)	

8. Metodología
<p>El profesor del curso presentará clases teóricas de los temas señalados en el programa propiciando la intervención de los alumnos.</p> <p>El profesor del curso presentará demostraciones para fundamentar clases teóricas.</p> <p>El profesor y los alumnos realizarán prácticas</p> <p>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.</p>

## 9. Evaluar

**Evaluación Continua 1** : 20 %

**Examen parcial** : 30 %

**Evaluación Continua 2** : 20 %

**Examen final** : 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.