



## Universidad Nacional de Ingeniería (UNI)

Programa Profesional de  
Ciencia de la Computación  
Sílabo 2023-I

### 1. CURSO

CS342. Compiladores (Obligatorio)

### 2. INFORMACIÓN GENERAL

2.1 Curso	:	CS342. Compiladores
2.2 Semestre	:	5 <sup>to</sup> Semestre.
2.3 Créditos	:	4
2.4 horas	:	2 HT; 4 HP;
2.5 Duración del periodo	:	16 semanas
2.6 Condición	:	Obligatorio
2.7 Modalidad de aprendizaje	:	Híbrido
2.8 Prerrequisitos	:	CS211. Teoría de la Computación. (4 <sup>to</sup> Sem) CS211. Teoría de la Computación. (4 <sup>to</sup> Sem)

### 3. PROFESORES

Atención previa coordinación con el profesor

### 4. INTRODUCCIÓN AL CURSO

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

### 5. OBJETIVOS

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

### 6. RESULTADOS DEL ESTUDIANTE

- 1) S.O. Analizar un problema computacional complejo y aplicar los principios computacionales y otras disciplinas relevantes para identificar soluciones. (**Evaluar**)
- 6) S.O. Aplicar la teoría de la computación y los fundamentos del desarrollo de software para producir soluciones basadas en computación. (**Evaluar**)

### 7. TEMAS

Unidad 1: Representación de programas (5)	
Resultados esperados:	
Temas	Objetivos de Aprendizaje
<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>
Lecturas : [Lou04b]	

Unidad 2: Traducción y ejecución de lenguajes (10)	
Resultados esperados:	
Temas	Objetivos de Aprendizaje
<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+11], [Lou04a], [TS98], [App02]	

Unidad 3: Análisis de sintaxis (10)	
Resultados esperados:	
Temas	Objetivos de Aprendizaje
<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+11], [Lou04a], [TS98], [App02]	

Unidad 4: Análisis semántico de compiladores (15)	
Resultados esperados:	
Temas	Objetivos de Aprendizaje
<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>
Lecturas : [Aho+11], [Lou04a], [TS98], [App02]	

Unidad 5: Generación de código (20)	
Resultados esperados:	
Temas	Objetivos de Aprendizaje
<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>
Lecturas : [Aho+11], [Lou04a], [TS98], [App02]	

## 8. PLAN DE TRABAJO

### 8.1 Metodología

Se fomenta la participación individual y en equipo para exponer sus ideas, motivándolos con puntos adicionales en las diferentes etapas de la evaluación del curso.

### 8.2 Sesiones Teóricas

Las sesiones de teoría se llevan a cabo en clases magistrales donde se realizarán actividades que propicien un aprendizaje activo, con dinámicas que permitan a los estudiantes interiorizar los conceptos.

### 8.3 Sesiones Prácticas

Las sesiones prácticas se llevan en clase donde se desarrollan una serie de ejercicios y/o conceptos prácticos mediante planteamiento de problemas, la resolución de problemas, ejercicios puntuales y/o en contextos aplicativos.

## 9. SISTEMA DE EVALUACIÓN

\*\*\*\*\* EVALUATION MISSING \*\*\*\*\*

## 10. BIBLIOGRAFÍA BÁSICA

- [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.