Universidad Nacional de Ingeniería (UNI)
Programa Profesional de
Inteligencia Artificial
Sílabo 2024-I

1. CURSO

CS342. Compiladores (Obligatorio)

2. INFORMACIÓN GENERAL

2.1 Curso	:	CS342. Compiladores
2.2 Semestre	:	5^{to} 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	:	Presencial
2.8 Prerrequisitos	:	CS211. Teoría de la Computación. (4^{to} 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) Analizar un problema computacional complejo y aplicar los principios computacionales y otras disciplinas relevantes para identificar soluciones. (**Evaluar**)

6) 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 horas)	
Resultados esperados:	
Temas	**Objetivos de Aprendizaje**
• Programas que tienen otros programas como entrada tales como interpretes, compiladores, revisores de tipos y generadores de documentación. • Arboles de sintaxis abstracta, para contrastar la sintaxis correcta. • Estructuras de datos que representan código para ejecución, traducción o transmisión. • Compilación en tiempo just-in time y re-compilación dinámica. • Otras características comunes de las máquinas virtuales, tales como carga de clases, hilos y seguridad.	• Explicar como programas que procesan otros programas tratan a los otros programas como su entrada de datos [Familiarizarse] • Describir un árbol de sintaxis abstracto para un lenguaje pequeño [Familiarizarse] • Describir los beneficios de tener representaciones de programas que no sean cadenas de código fuente [Familiarizarse] • 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] • 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 *frame* [Familiarizarse] • Discutir las ventajas, desventajas y dificultades del término (*just-in-time*) y recompilación automática [Familiarizarse] • Identificar los servicios proporcionados por los sistemas de tiempo de ejecución en lenguajes modernos [Familiarizarse]
Lecturas : [Lou04b]	

Unidad 2: Traducción y ejecución de lenguajes (10 horas)	
Resultados esperados:	
Temas	**Objetivos de Aprendizaje**
• Interpretación vs. compilación a código nativo vs. compilación de representación portable intermedia. • Pipeline de traducción de lenguajes: análisis, revisión opcional de tipos, traducción, enlazamiento, ejecución: – Ejecución como código nativo o con una máquina virtual – Alternativas como carga dinámica y codificación dinámica de código (o "just-in-time"") • Representación en tiempo de ejecución de construcción del lenguaje núcleo tales como objetos (tablas de métodos) y funciones de primera clase (cerradas) • Ejecución en tiempo real de asignación de memoria: pila de llamdas, montículo, datos estáticos: – Implementación de bucles, recursividad y llamadas de cola • Gestión de memoria: – Gestión manual de memoria: asignación, limpieza y reuso de la pila de memoria – Gestión automática de memoria: recolección de datos no utilizados (*garbage colletion*) como una técnica automática usando la noción de accesibilidad	• 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] • Distinguir sintaxis y parseo de la semantica y la evaluación [Evaluar] • Bosqueje una representación de bajo nivel de tiempo de ejecución de construcciones del lenguaje base, tales como objetos o cierres (*closures*) [Evaluar] • Explicar cómo las implementaciones de los lenguajes de programación tipicamente organizan la memoria en datos globales, texto, *heap*, y secciones de pila y cómo las características tales como recursión y administración de memoria son mapeados a esté modelo de memoria [Evaluar] • Identificar y corregir las pérdidas de memoria y punteros desreferenciados [Evaluar] • Discutir los beneficios y limitaciones de la recolección de basura (*garbage collection*), incluyendo la noción de accesibilidad [Evaluar]
Lecturas : [Aho+11], [Lou04a], [TS98], [App02]	

Unidad 3: Análisis de sintaxis (10 horas)	
Resultados esperados:	
Temas	**Objetivos de Aprendizaje**
• Exploración (análisis léxico) usando expresiones regulares. • Estratégias 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. • Generación de exploradores (scanners) y analizadores a partir de especificaciones declarativas.	• Usar gramáticas formales para especificar la sintaxis de los lenguajes [Evaluar] • Usar herramientas declarativas para generar parseadores y escáneres [Evaluar] • Identificar las características clave en las definiciones de sintaxis: ambiguedad, asociatividad, precedencia [Evaluar]
Lecturas : [Aho+11], [Lou04a], [TS98], [App02]	

Unidad 4: Análisis semántico de compiladores (15 horas)	
Resultados esperados:	
Temas	**Objetivos de Aprendizaje**
• Representaciones de programas de alto nivel tales como árboles de sintaxis abstractas. • Alcance y resolución de vínculos. • Revisión de tipos. • Especificaciones declarativas tales como gramáticas atribuídas.	• 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] • Describir analizadores semanticos usando una gramatica con atributos [Evaluar]
Lecturas : [Aho+11], [Lou04a], [TS98], [App02]	

Unidad 5: Generación de código (20 horas)	
Resultados esperados:	
Temas	**Objetivos de Aprendizaje**
• Llamadas a procedimientos y métodos en envío. • Compilación separada; vinculación. • Selección de instrucciones. • Calendarización de instrucciones. • Asignación de registros. • Optimización por rendija (peephole)	• Identificar todos los pasos esenciales para convertir automáticamente código fuente en código emsamblador o otros lenguajes de bajo nivel [Evaluar] • Generar código de bajo nivel para llamadas a funciones en lenguajes modernos [Evaluar] • Discutir por qué la compilación separada requiere convenciones de llamadas uniformes [Evaluar] • Discutir por qué la compilación separada limita la optimización debido a efectos de llamadas desconocidas [Evaluar] • Discutir oportunidades para optimización introducida por la traducción y enfoques para alcanzar la optimización, tales como la selección de la instrucción, planificación de instruccion, asignación de registros y optimización de tipo mirilla (*peephole optimization*) [Evaluar]
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.