Universidad Nacional de Ingeniería (UNI)
Escuela Profesional de
Ciencia de la Computación
Sílabo 2026-I

1. CURSO

CS342. Compiladores (Obligatorio)

2. INFORMACIÓN GENERAL

2.1 Curso	:	CS342. Compiladores
2.2 Semestre	:	6^{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

Este curso proporciona una comprensión profunda de cómo los lenguajes de alto nivel se transforman en código máquina ejecutable. El estudio de los compiladores permite a los estudiantes comprender la semántica de los lenguajes, la gestión de memoria y la optimización de código, que son habilidades fundamentales para diseñar lenguajes específicos de dominio y realizar ingeniería de software de alto rendimiento.

5. OBJETIVOS

- Comprender las fases de un compilador y las herramientas de construcción.
- Implementar analizadores léxicos y sintácticos basados en gramáticas.
- Generar y optimizar código intermedio para una arquitectura específica.

6. RESULTADOS DEL ESTUDIANTE

AG-C08) Análisis de Problemas: Identifica, formula y analiza problemas complejos de computación. (Usage)

AG-C12) Usa enfoques sistémicos para seleccionar, desarrollar, aplica, integrar y administrar tecnologías seguras de computación para lograr los objetivos del usuario. (Usage)

7. TEMAS

Unidad 1: Análisis de Programas y Analizadores (20 horas)	
Resultados esperados: AG-C08,AG-C12	
Temas	**Aprendizaje esperado (*Learning Outcomes*)**
• Representaciones de programa relevantes, como bloques básicos (basic blocks), grafos de flujo de control (control-flow graphs), cadenas definición-uso (def-use chains) y asignación única estática (static single assignment) • Indecidibilidad y consecuencias para el análisis de programas • Análisis insensible al flujo (flow-insensitive), como verificación de tipos y análisis de punteros y alias escalables • Análisis sensible al flujo (flow-sensitive), como análisis de flujo de datos (dataflow) hacia adelante y hacia atrás • Análisis sensible a la ruta (path-sensitive), como verificación de modelos de software (software model checking) y verificación de software • Herramientas y frameworks para implementar analizadores • Papel del análisis estático en la optimización de programas y el análisis de dependencia de datos durante la explotación de concurrencia • Papel del análisis de programas en la verificación (parcial) y la búsqueda de errores (bug-finding) • Paralelización: – Análisis para auto-paralelización – Análisis para detectar errores de concurrencia	• Explicar la diferencia entre grafo de flujo de datos (dataflow graph) y grafo de flujo de control [Explicar] • Explicar por qué los análisis de programa no triviales y sólidos (sound) deben ser aproximados [Explicar] • Argumentar por qué un análisis es correcto (sound y terminante) [Argumentar] • Explicar por qué el alias potencial limita el análisis de programa sólido y cómo el análisis de alias puede ayudar [Explicar] • Usar los resultados de un análisis de programa para optimización de programas y/o corrección parcial del programa [Usar]
Lecturas : [Aho+06], [App04]	

Unidad 2: Traducción y Ejecución de Lenguajes (24 horas)	
Resultados esperados: AG-C08,AG-C12	
Temas	**Aprendizaje esperado (*Learning Outcomes*)**
• Modelos de ejecución para JIT (Just-In-Time), compilador, intérprete • Uso de código intermedio, por ejemplo, bytecode • Limitaciones y beneficios de JIT, compilador e intérprete • Compiladores/transpiladores cruzados (cross compilers/transpilers) • Representación BNF y BNF extendida de gramática libre de contexto • árbol de análisis (parse tree) usando una oración simple como expresión aritmética o sentencia if-then-else • Ejecución como código nativo o dentro de una máquina virtual • Canalización de traducción de lenguaje: análisis sintáctico, análisis (parsing), verificación de tipos opcional, generación de código/traducción y optimización, enlace (linking), carga (loading), ejecución • Representación en tiempo de ejecución de construcciones centrales del lenguaje como objetos (tablas de métodos) y funciones que pueden pasarse como parámetros a y devolverse desde funciones (clausuras) • Desarrollo seguro de compiladores	• Explicar y comprender las diferencias entre implementaciones de lenguaje compilado, JIT e interpretado, incluyendo los beneficios y limitaciones de cada una [Explicar] • Diferenciar sintaxis y análisis (parsing) de semántica y evaluación [Diferenciar] • Usar BNF y BNF extendida para especificar la sintaxis de construcciones simples como if-then-else, declaración de tipo y construcciones iterativas para lenguajes conocidos como C++ o Python [Usar] • Ilustrar el árbol de análisis usando una oración/expresión aritmética simple [Aplicar] • Ilustrar traducción de diagramas de sintaxis a BNF/BNF extendida para construcciones simples como if-then-else, declaración de tipo, construcciones iterativas, etc [Aplicar] • Ilustrar ambigüedad en el análisis usando if-then-else anidado/expresión aritmética y mostrar resolución usando orden de precedencia [Aplicar] • Discutir los beneficios y limitaciones de la recolección de basura, incluyendo la noción de alcanzabilidad [Debatir]
Lecturas : [Aho+06], [App04]	

Unidad 3: Comportamiento en Tiempo de Ejecución y Sistemas (20 horas)	
Resultados esperados: AG-C08,AG-C12	
Temas	**Aprendizaje esperado (*Learning Outcomes*)**
• Modelos de proceso usando pilas y montones para asignar y desasignar registros de activación y recuperar entornos usando punteros de marco (frame pointers) y direcciones de retorno durante una llamada a procedimiento, incluyendo ejemplos de paso de parámetros • Esquemas de búsqueda de código usando tablas hash para métodos en implementaciones de programas orientados a objetos • Distribución de datos para objetos y registros de activación • Asignación de objetos en el montón (heap) • Implementar entidades virtuales y métodos virtuales; tablas de métodos virtuales (virtual method tables) y su aplicación • Comportamiento en tiempo de ejecución de programas orientados a objetos • Comparar y contrastar la asignación de memoria durante el intercambio de información usando paso de parámetros y variables no locales (usando cadena de enlaces estáticos). • Enfoques y técnicas de gestión dinámica de memoria: malloc/free, recolección de basura (mark-sweep, copying, reference counting), regiones (también conocidas como arenas o zonas) • Compilación justo a tiempo (just-in-time) y recompilación dinámica • Interfaz con el sistema operativo (por ejemplo, para inicialización del programa) • Interoperabilidad entre lenguajes de programación incluyendo mecanismos de paso de parámetros y representación de datos: – *Big endian*, *little endian* – Distribución de datos de tipos de datos compuestos como arreglos • Otras características comunes de las máquinas virtuales, como carga de clases (class loading), hilos (threads) y verificación de seguridad • Sandboxing	• Discutir beneficios y limitaciones de la gestión automática de memoria [Debatir] • Explicar el uso de metadatos en representaciones en tiempo de ejecución de objetos y registros de activación, como punteros de clase, longitudes de arreglo, direcciones de retorno y punteros de marco (frame pointers) [Explicar] • Comparar y contrastar asignación estática vs asignación basada en pila vs asignación basada en montón de elementos de datos [Comparar] • Explicar por qué algunos elementos de datos no pueden desasignarse automáticamente al final de una llamada a procedimiento/método (necesidad de recolección de basura) [Explicar] • Discutir ventajas, desventajas y dificultades de la compilación justo a tiempo y la recompilación dinámica [Debatir] • Discutir el uso del sandboxing en código móvil [Debatir] • Identificar los servicios proporcionados por los sistemas en tiempo de ejecución de lenguajes modernos [Analizar]
Lecturas : [Aho+06], [CT11]	

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

[App04] Andrew W. Appel. *Modern Compiler Implementation in Java*. 2nd. Cambridge University Press, 2004.

[Aho+06] Alfred V. Aho et al. *Compilers: Principles, Techniques, and Tools*. 2nd. Pearson, 2006.

[CT11] Keith Cooper and Linda Torczon. *Engineering a Compiler*. 2nd. Morgan Kaufmann, 2011.