National University of Engineering (UNI)
School of Cybersecurity
Syllabus 2024-II

1. COURSE

CS342. Compilers (Mandatory)

2. GENERAL INFORMATION

2.1 Course	:	CS342. Compilers
2.2 Semester	:	5^{th} Semester.
2.3 Credits	:	4
2.4 Horas	:	2 HT; 4 HP;
2.5 Duration of the period	:	16 weeks
2.6 Type of course	:	Mandatory
2.7 Learning modality	:	Face to face
2.8 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

1) Analyze a complex computing problem and apply principles of computing and other relevant disciplines to identify solutions. (Assessment)

6) Apply security principles and practices to maintain operations in the presence of risks and threats. (Assessment)

7. TOPICS

Unit 1: Representación de programas (5 hours)	
Competences Expected:	
Topics	**Learning Outcomes**
• 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]
Readings : [**Lou004LP**]	

Unit 2: Traducción y ejecución de lenguajes (10 hours)

Competences Expected:

Topics	Learning Outcomes
• 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]

Readings : [**Aho2008**], [**Lou004CO**], [**Appe002**], [**Teu98**]

Unit 3: Análisis de sintaxis (10 hours)

Competences Expected:

Topics	Learning Outcomes
• 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]

Readings : [**Aho2008**], [**Lou004CO**], [**Appe002**], [**Teu98**]

Unit 4: Análisis semántico de compiladores (15 hours)	
Competences Expected:	
Topics	**Learning Outcomes**
• 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]
Readings : [**Aho2008**], [**Lou004CO**], [**Appe002**], [**Teu98**]	

Unit 5: Generación de código (20 hours)	
Competences Expected:	
Topics	**Learning Outcomes**
• 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]
Readings : [**Aho2008**], [**Lou004CO**], [**Appe002**], [**Teu98**]	

8. WORKPLAN

8.1 Methodology

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

8.2 Theory Sessions

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

8.3 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.

9. EVALUATION SYSTEM

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

10. BASIC BIBLIOGRAPHY