CIENCIA DE LA COMPUTACION

COMPILADORES

04 CRÉDITOS

UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

CE2A

ÍNDICE

UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA
SILABO 2020-2

1. ASIGNATURA

CS3402 - Compiladores

2. DATOS GENERALES

2.1.**Créditos:** cuatro (4) créditos
2.2. Horas de teoría: dos (2) semanales
2.3. Horas de práctica: cuatro (4) semanales
2.4. Duración del período: dieciséis (16) semanas
2.5. Condición:
Obligatorio para Ciencia de la Computación
2.6. Modalidad: Virtual
2.7. Requisitos:
CS2101. Teoría de la Computación. (4° Sem)

3. PROFESORES

3.1. Profesor coordinador del curso

José Antonio Fiestas Iquira (jfiestas@utec.edu.pe)
Horario de atención: Jueves 15:00-16:00

3.2. Profesor(es) instructor(es) del curso

José Antonio Fiestas Iquira (jfiestas@utec.edu.pe)
Horario de atención: Jueves 15:00-16:00

4. INTRODUCCIÓN AL CURSO

Compiladores e intérpretes son parte esencial de un sistema computacional. Sin ellos tendríamos aún que programar en lenguaje de máquina (Assembler). Durante el curso se impartirán los conceptos y principios fundamentales de la teoría de compilación para realizar el diseño y la implementación de un compilador. Esto se hará tanto en forma teórica como práctica, lo que se verá reflejado en la comprensión de conceptos y principios, asi como con los ejercicios aplicativos, y en el desarrollo del proyecto de curso. El estudiante desarrollará una visión global del problema del diseño y construcción de un compilador, siendo capaz de apli-

car el conocimiento adquirido en tópicos relacionados a lenguajes de programación, arquitectura, algoritmos e ingeniería de software.

5. OBJETIVOS

Sesión 1: Explicar cómo programas que procesan otros programas tratan a los otros programas como su entrada de datos. Describir un árbol de sintaxis abstracto para un lenguaje pequeño

Sesión 2: Describir los beneficios de tener representaciones de programas que no sean cadenas de código fuente.

Sesión 3: Escribir un programa para procesar alguna representación de código para algún propósito, tales como un interprete, una expresión optimizada, o un generador de documentación.

Sesión 4: 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.

Sesión 5: Discutir las ventajas, desventajas y dificultades del término (just-in-time) y recompilación automática. Identificar los servicios proporcionados por los sistemas de tiempo de ejecución en lenguajes modernos.

Sesión 6: 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). Distinguir sintaxis y parseo de la semantica y la evaluación.

Sesión 7: Bosqueje una representación de bajo nivel de tiempo de ejecución de construcciones del lenguaje base, tales como objetos o cierres (closures)

Sesión 8: Explicar cómo las implementaciones de los lenguajes de programación tipicamente organizan la memoria en datos globales, texto, heap, y secciones de pila y como las características tales como recursión y administración de memoria son mapeados a este modelo de memoria. Identificar y corregir las pérdidas de memoria y punteros desreferenciados. Discutir los beneficios y limitaciones de la recolección de basura (garbage collection), incluyendo la noción de accesibilidad.

Sesión 9: Usar gramáticas formales para especificar la sintaxis de los lenguajes

Sesión 10: Usar herramientas declarativas para generar parseadores y escáneres.

Sesión 11: Identificar las características clave en las definiciones de sintaxis: ambiguedad, asociatividad, precedencia.

Sesión 12: Implementar analizadores sensibles al contexto y estáticos a nivel de fuente, tales como, verificadores de tipos o resolvedores de identificadores para iden- tificar las ocurrencias de vinculo.

Sesión 13: Describir analizadores semánticos usando una gramática con atributos.

Sesión 14: Identificar todos los pasos esenciales para convertir automáticamente código fuente en código emsamblador o otros lenguajes de bajo nivel. Generar código de bajo nivel para llamadas a funciones en lenguajes modernos.

Sesión 15: Discutir por qué la compilación separada requiere convenciones de llamadas uniformes. Discutir por qué la compilación separada limita la optimización debido a efectos de llamadas desconocidas. 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 (peephole optimization)

6. COMPETENCIAS

a2. Aplicar conocimientos de computación y de matemáticas apropiadas para la disciplina.
El estudiante obtendrá un concepto global adecuado del diseño de un compilador a través de ejercicios de interpretación de gramáticas y expresiones regulares.

a3. Aplicar la base matemática, principios de algoritmos y la teoría de la Computación en el modelamiento y diseño de sistemas.
El estudiante logrará formular el diseño de un compilador basado en los conocimientos adquiridos en Teoría de la Computación, a través de ejercicios aplicativos

b2. Analizar problemas e identificar y definir los requerimientos computacionales apropiados para su solución.
El estudiante logrará aplicar los conocimientos teóricos adquiridos a través de casos prácticos en clase, y en el desarrollo del proyecto del curso.

7. RESULTADOS DE APRENDIZAJE

Al final del curso de Compiladores se espera:

RA1. Que el estudiante sea capaz de conocer las técnicas básicas empleadas durante el proceso de generación intermedio, optimización y generación de código. Esto se logrará paulatinamente durante el curso, en las etapas mencionadas de construcción de un compilador a través de los ejercicios de clase propuestos.

RA2. Que el estudiante sea capaz de implementar pequeños compiladores., que sean cad vez mas complejos. Esto se logrará a través de la implementación por etapas de un compilador. El objetivo es que lo aprendido se consolide con la culminación satisfactoria del proyecto de curso.

8. TEMAS

1. Representación de programas

1.1 Programas que tienen otros programas como entrada tales como interpretes, compiladores, revisores de tipos y generadores de documentación.

1.2 Estructuras de datos que representan código para ejecución, traducción o transmisión.

1.3 Arboles de sintaxis abstracta, para contrastar la sintaxis correcta.

2: Traducción y ejecución de lenguajes

2.1 Interpretación vs. compilación a código nativo vs. compilación de representación portable intermedia.

2.2 Pipeline de traducción de lenguajes: análisis, revisión opcional de tipos, traducción, enlazamiento, ejecución:

2.2.1 Ejecución como código nativo o con una máquina virtual

2.2.2 Alternativas como carga dinámica y codificación dinámica de código (o “just-in-time””)

2.3 Representación en tiempo de ejecución de construcción del lenguaje núcleo tales como objetos (tablas de métodos) y funciones de primera clase (cerradas)

2.4 Ejecución en tiempo real de asignación de memoria: pila de llamadas, montículo, datos estáticos:

– Implementación de bucles, recursividad y llamadas de cola

2.5 Gestión de memoria:

2.5.1. Gestión manual de memoria: asignación, limpieza y reuso de la pila de memoria

2.5.2. Gestión automática de memoria: recolección de datos no utilizados (garbage collection)

3: Análisis de sintaxis

3.1 Exploración (análisis léxico) usando expresiones regulares.

3.2. 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.

3.3. Generación de exploradores (scanners) y analizadores a partir de especificaciones declarativas.

4: Análisis semántico de compiladores

4.1 Representaciones de programas de alto nivel tales como árboles de sintaxis abstractas.

4.2. Alcance y resolución de vínculos.

4.3. Revisión de tipos.
4.4. Especificaciones declarativas tales como gramáticas atribuídas.

5: Generación de código
5.1. Llamadas a procedimientos y métodos en envío.
5.2. Compilación separada; vinculación. Selección de instrucciones. Calendarización de instrucciones.
5.3. Asignación de registros. Optimización por rendija (peephole)

9. PLAN DE TRABAJO

9.1 Metodología

La metodología del curso se enfoca en clases magistrales y proyectos de investigación, en las clases teóricas (1 a la semana) y prácticas (2 a la semana) respectivamente. En las clases teóricas se imparten conocimientos necesarios para el desarrollo de ejercicios prácticos semanales. De esta manera se logrará una evaluación continua del avance del alumno. El conocimiento teórico adquirido se evaluará en los exámenes parcial y final, mientras que el práctico en el proyecto del curso.

9.2. **Sesiones Teóricas:**

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

9.3 **Sesiones de Laboratorio**:

Para verificar que los alumnos hayan alcanzado el logro planteado para cada una de las unidades de aprendizaje, realizarán actividades que les permita aplicar los conocimientos adquiridos durante las sesiones de teoría y se les propondrá retos que permitan evaluar el desempeño de los alumnos.

Del mismo modo, se fomenta la participación en equipo a través de proyectos de investigación que se realizan de manera grupal. Estos se presentarán en forma escrita y oral, para dar la oportunidad al alumno, de exponer sus ideas.

Como parte de la evaluación continua se asignarán tareas semanales en forma grupal o individual, motivándolos con puntos adicionales en las diferentes etapas de la evaluación del curso.

10. SISTEMA DE EVALUACIÓN

EVALUACIÓN *La ponderación de la evaluación se hará si ambas partes están aprobadas	**TEORÍA**	**PRÁCTICA Y/O LABORATORIO**
	1 Nota de Evaluación Contínua (C) (10%) 1 Exámen Parcial (E1) (25 %) 1 Exámen Final (E2)(25 %)	1 Nota de Evaluación Contínua (C1)(5%, primeras 8 semanas) 1 Nota de Evaluación Contínua (C2)(5%, semanas 8-16) 1. Proyecto (P) (30 %)
	40%	60%
	100%	

Se utilizarán las siguientes rúbricas para medir las competencias del curso, y evaluar las actividades más significativas del curso: enlace

11. SESIONES DE APOYO O TUTORÍAS

Este apartado permite formalizar los espacios de apoyo a los estudiantes y que éstos tengan la atención NECESARIA y el tiempo disponible para presentar sus dudas y consultas acerca del curso:

Semana	Fecha/ Hora	Tema a tratar	Objetivos de la sesión
2	08/09/2020 2:00 PM – 2:30 PM	Evaluaciones y rubricas	Resolver dudas sobre exámenes y rubricas , asi como participación en clase
4	22/09/2020 2:00 PM – 2:30 PM	Topicos del curso	Resolver dudas sobre primeras 3 semanas
6	06/10/2020 2:00 PM – 2:30 PM	Proyecto	Consultas sobre proyecto y rubricas
8	20/10/2020 2:00 PM – 2:30 PM	Parcial	Solucionario examen parcial

10	03/11/2020 2:00 PM – 2:30 PM	Topicos del curso	Resolver dudas sobre temas tratados en clase
12	17/11/2020 2:00 PM – 2:30 PM	Proyecto	Consultas sobre presentación de proyecto
14	01/12/2020 2:00 PM – 2:30 PM	Final	Consultas sobre ultimas notas y examen final

12. REFERENCIAS BIBLIOGRÁFICAS

- Alfred Aho et al. (2011) Compilers Principles Techniques And Tools. 2nd. ISBN:10-970-26-1133-4. Pearson
- W. Appel. (2002) Modern compiler implementation in Java. 2.a edición. Cambridge University Press.
- Kenneth C. Louden. (2004) Compiler Construction: Principles and Practice. Thomson.
- Kenneth C. Louden. (2004) Lenguajes de Programacion. Thomson.
- Bernard Teufel and Stephanie Schmidt. (1998) Fundamentos de Compiladores. Addison Wesley Iberoamericana.