# CIENCIA DE LA COMPUTACIÓN

## COMPILADORES

**4 CRÉDITOS**


UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

CE2A

# ÍNDICE

# UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA
# SILABO 2021-1

## 1. ASIGNATURA

CS3402. Compiladores

## 2. DATOS GENERALES

**2.1 Ciclo:** 5°
**2.2 Créditos:** cuatro (4) créditos
**2.3 Horas de teoría:** dos (2) semanales
**2.4 Horas de práctica:** cuatro (4) semanales
**2.5 Duración del período:** dieciséis (16) semanas
**2.6 Condición:** Obligatorio para Ciencia de la Computación
**2.7 Modalidad:** Virtual
**2.8 Requisitos:** CS2101. Teoría de la Computación. (4$^{to}$ Sem)

## 2. PROFESORES

**3.1. Profesor coordinador del curso**
Rommel Anatoli Quintanilla Cruz ( rquintanilla@utec.edu.pe )
Horario de atención: viernes 7:00-8:00 pm

**3.2. Profesor(es) instructor(es) del curso**
Rommel Anatoli Quintanilla Cruz ( rquintanilla@utec.edu.pe )
Horario de atención: viernes 7:00-8:00 pm

## 3. 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, así como con los ejercicios aplicativos, y en el desarrollo del proyecto de curso. Se desarrollará una visión global del problema del diseño y construcción de un compilador, donde se podrá aplicar el conocimiento adquirido en tópicos relacionados a lenguajes de programación, arquitectura, algoritmos e ingeniería de software.

## 4. 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. Describir los beneficios de tener representaciones de programas que no sean cadenas de código fuente.

- **Sesión 2:** Escribir un programa para procesar alguna representación de código para algún propósito, tales como un intérprete, una expresión optimizada, o un generador de documentación. 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* Discutir las ventajas, desventajas y dificultades del término (*just-in-time*) y re-compilación automática Identificar los servicios proporcionados por los sistemas de tiempo de ejecución en lenguajes modernos

- **Sesión 3:** 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 semántica y la evaluación. Bosqueje una representación de bajo nivel de tiempo de ejecución de construcciones del lenguaje base, tales como objetos o cierres (closures)

- **Sesión 4:** Explicar cómo las implementaciones de los lenguajes de programación típicamente 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 5:** Usar gramáticas formales para especificar la sintaxis de los lenguajes. Usar herramientas declarativas para generar parseadores y escáneres. Identificar las características clave en las definiciones de sintaxis: ambigüedad, asociatividad, precedencia

- **Sesión 6:** 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 vínculo. Describir analizadores semánticos usando una gramática con atributos

- **Sesión 7**: Identificar todos los pasos esenciales para convertir automáticamente código fuente en código ensamblador o otros lenguajes de bajo nivel. Generar código de bajo nivel para llamadas a funciones en lenguajes modernos. Discutir por qué la compilación separada requiere convenciones de llamadas uniformes

- **Sesión 8**: Discutir por qué la compilación separada limita la optimización debido a efectos de llamadas desconocidas. Discutir oportunidades para la 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 Y CRITERIOS DE DESEMPEÑO

Los criterios de desempeño que se van a trabajar en este curso son:

1.3. Aplica conocimientos de computación apropiados para la solución de problemas definidos y sus requerimientos en la disciplina del programa.(nivel 3)

2.4. Resuelve problemas de computación y otras disciplinas relevantes en el dominio. (nivel 3)

3.2. Diseña, implementa y evalúa soluciones a problemas complejos de computación. (nivel 3)

5.2. Lidera eficazmente equipos diversos. (nivel 2)

## 7. RESULTADOS DE APRENDIZAJE

Al final del curso el estudiante de Compiladores se espera que el estudiante sea capaz de:

**RA1. Evaluar** técnicas básicas de generación de código intermedio, optimización y código ensamblador

**RA2**. **Diseñar** soluciones a problemas en cada una de las etapas de construcción de un compilador, sea análisis léxico, sintáctico, semántico, y con métodos de optimización

**RA3. Construir** pequeños compiladores con los elementos básicos de análisis léxico, sintáctico, semántico, y usando métodos de optimización; así como la generación de código intermedio y ensamblador para casos prácticos de traducción

**RA4**. **Liderar** la construcción de soluciones integrales en trabajo colaborativo optimizando la distribución de tareas para construir y optimizar un compilador. (grupal).

## 8. TEMAS

### 1. Representación de programas

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

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

**1.3.** Árboles de sintaxis abstracta, para contrastar la sintaxis correcta.

**1.4.** Compilación en tiempo just-in time y re-compilación dinámica.

**1.5.** Otras características comunes de las máquinas virtuales, tales como carga de clases, hilos y seguridad

## 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 reúso de la pila de memoria.

**2.5.2.** Gestión automática de memoria: recolección de datos no utilizados (garbage collection) como una técnica automática usando la noción de accesibilidad

## 3. Análisis de sintaxis

**3.1.** Exploración (análisis léxico) usando expresiones regulares.

**3.2.** 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 - backtracking, 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 atribuidas.

**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:

El curso está organizado en sesiones virtuales teóricas y prácticas, donde se impulsa la aplicación de los conceptos tratados en clase a través de ejercicios (tanto individuales como grupales). Además, se desarrolla un proyecto como asignación para la casa, que cubre todos los temas relevantes y su aplicación en casos prácticos.

El alumno es evaluado constantemente (evaluación continua) y se hace el seguimiento del avance para un mejor control del desempeño

### 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. Así mismo, se asignan tareas semanales en forma grupal o individual, motivándolos con puntos adicionales en las diferentes etapas de la evaluación del curso.

### 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. De la misma manera, se fomenta la participación en equipo a través de proyectos de investigación. Estos se presentarán en forma escrita y oral, para dar la oportunidad al alumno, de exponer sus ideas,

## 10. SISTEMA DE EVALUACIÓN

| | TEORÍA | PRÁCTICA Y/O LABORATORIO |
|---|---|---|
| **EVALUACIÓN**<br><br>*La ponderación de la evaluación se hará si ambas partes están aprobadas | Evaluación Continua **C1** (10 %)<br>Evaluación Continua **C2** (10 %)<br>Examen Parcia **E1** (25 %)<br>Examen Final **E2** (25 %) | Proyecto **P1** (30 %) |
| | 70% | 30% |
| | **100%** | |

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

## 11. 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 Programación. Thomson.
- Bernard Teufel and Stephanie Schmidt. (1998) Fundamentos de Compiladores. Addison Wesley Iberoamericana.