

## 1.- DATOS DE LA ASIGNATURA

Nombre de la asignatura: <b>Programación de sistemas</b>
Carrera: <b>Ingeniería en Sistemas Computacionales</b>
Clave de la asignatura: <b>SCC - 0426</b>
Horas teoría-horas práctica-créditos <b>4-2-10</b>

## 2.- HISTORIA DEL PROGRAMA

<b>Lugar y fecha de elaboración o revisión</b>	<b>Participantes</b>	<b>Observaciones (cambios y justificación)</b>
Instituto Tecnológico de Toluca del 18 al 22 agosto 2003.	Representantes de la academia de sistemas y computación de los Institutos Tecnológicos.	Reunión nacional de evaluación curricular de la carrera de Ingeniería en Sistemas Computacionales.
Instituto Tecnológico de: Acapulco, Cd. Juárez Minatitlán. 23 agosto al 7 de noviembre 2003.	Academia de sistemas y computación.	Análisis y enriquecimiento de las propuestas de los programas diseñados en la reunión nacional de evaluación.
Instituto Tecnológico de León 1 al 5 de marzo 2004.	Comité de consolidación de la carrera de Ingeniería en Sistemas Computacionales.	Definición de los programas de estudio de la carrera de Ingeniería en Sistemas Computacionales.

### 3.- UBICACIÓN DE LA ASIGNATURA

#### a). Relación con otras asignaturas del plan de estudio

Anteriores		Posteriores	
Asignaturas	Temas	Asignaturas	Temas
Teoría de la Computación	Lenguajes libre de contexto  Autómatas finitos		

#### b). Aportación de la asignatura al perfil del egresado

Desarrolla software de base como: traductores, cargadores, ligadores, herramientas, utilerías, DBMS, generadores de código, etc.

### 4.- OBJETIVO(S) GENERAL(ES) DEL CURSO

El estudiante desarrollará software de base, tales como compiladores o interpretes.

## 5.- TEMARIO

Unidad	Temas	Subtemas
1	Introducción a la programación de sistemas	1.1 ¿Qué es y qué estudia la programación de sistemas? 1.2 Herramientas desarrolladas con la teoría de programación de sistemas. 1.3 Lenguajes. 1.3.1 Lenguajes naturales. 1.3.2 Lenguajes artificiales. 1.3.3 Proceso de la comunicación. 1.4 Traductor y su estructura. 1.4.1 Ensambladores. 1.4.2 Compiladores. 1.4.3 Interpretes. 1.5 Generadores de código para compiladores (compilador de compilador).
2	Introducción al diseño de los lenguajes de programación	2.1 Visión del problema. 2.2 Consideraciones Premilinares. 2.3 Objetivos y filosofías del diseño de los lenguajes de programación. 2.4 Diseño detallado. 2.5 Caso de estudio.
3	Análisis Léxico.	3.1 Introducción a los Autómatas finitos y expresiones regulares. 3.2 Analizador de léxico. 3.3 Manejo de localidades temporales de memoria (buffers). 3.4 Creación de tablas de símbolos. 3.5 Manejo de errores léxicos. 3.6 Generadores de código léxico: Lex y Flex.
4	Análisis sintáctico.	4.1 Introducción a las Gramáticas libres de contexto y árboles de derivación. 4.2 Diagramas de sintaxis. 4.3 Precedencia de operadores. 4.4 Analizador sintáctico. 4.4.1 Analizador descendente (LL). 4.4.2 Analizador ascendente(LR, LALR.

## 5.- TEMARIO (Continuación)

5	Análisis semántico	4.5 Administración de tablas de símbolos. 4.6 Manejo de errores sintácticos y su recuperación. 4.7 Generadores de código para analizadores sintácticos: Yacc, Bison  5.1 Analizador semántico 5.2 Verificación de tipos en expresiones. 5.3 Conversión de tipos. 5.4 Acciones agregadas en un analizador sintáctico descendente (top-down). 5.5 Pila semántica en un analizador sintáctico ascendente (bottom-up). 5.6 Administración de la tabla de símbolos. 5.7 Manejo de errores semánticos.
6	Generación de código intermedio.	6.1 Lenguajes intermedios. 6.2 Notaciones. 6.2.1 Infija. 6.2.2 Postfija. 6.2.3 Prefija. 6.3 Representación de código intermedio. 6.3.1 Notación Polaca. 6.3.2 Código P. 6.3.3 Triplos. 6.3.4 Cuádruplos. 6.4 Esquemas de generación. 6.4.1 Expresiones. 6.4.2 Declaración de variables, constantes 6.4.3 Estatuto de asignación. 6.4.4 Estatuto condicional. 6.4.5 Estatuto de ciclos 6.4.6 Arreglos. 6.4.7 Funciones.

## 5.- TEMARIO (Continuación)

7	Optimización.	7.1 Tipos de optimización. 7.1.1 Locales. 7.1.2 Bucles. 7.1.3 Globales. 7.1.4 De mirilla. 7.2 Costos. 7.2.1 Costo de ejecución. 7.2.2 Criterios para mejorar el código. 7.2.3 Herramientas para el análisis del flujo de datos.
8	Generación de código objeto.	8.1 Lenguaje máquina. 8.1.1 Características. 8.1.2 Direccionamiento. 8.2 Lenguaje ensamblador. 8.2.1 Características. 8.2.2 Almacenamiento. 8.3 Registros. 8.3.1 Distribución. 8.3.2 Asignación. 8.4 Administración de memoria.

## 6.- APRENDIZAJES REQUERIDOS

- Conocer la arquitectura de una computadora.
- Dominar algún lenguaje de programación de alto nivel.
- Utilizar algún lenguaje de programación bajo nivel.
- Dominar la teoría e implementación de autómatas.
- Dominar la teoría de lenguajes libres de contexto.

## 7.- SUGERENCIAS DIDÁCTICAS

- Realizar Investigación en diversas fuentes de información sobre los conceptos de la asignatura, por equipos analizarlos y discutirlos en clase.
- Elaborar de manera conjunta ejercicios y prácticas coordinadas por el profesor.
- Llevar a cabo dinámicas grupales que permitan analizar la teoría con casos prácticos.
- Manejar herramientas de programación de sistemas formales como los metalenguajes.
- Desarrollar proyectos relacionados con el proceso de traducción o cualquier herramienta que se encuentre dentro del área del software de base.
- Presentar los resultados del desarrollo del proyecto final.

## 8.- SUGERENCIAS DE EVALUACIÓN

- Examen escrito.
- Investigaciones documentales.
- Elaboración de ensayos.
- Desarrollo de un proyecto final en donde se aplique la teoría de compiladores.
- Desarrollo de un proyecto final en donde se aporte el desarrollo de software en alguna de las áreas de la programación de sistemas.

## 9.- UNIDADES DE APRENDIZAJE

**UNIDAD 1.-** Introducción a la programación de sistemas.

<b>Objetivo Educativo</b>	<b>Actividades de Aprendizaje</b>	<b>Fuentes de Información</b>
El estudiante ubicará la importancia de la Programación de Sistemas	1.1 Realizar una investigación acerca de las áreas de aplicación de la programación de sistemas, analizarla y discutir en el grupo. 1.2 Buscar a través de diferentes medios, herramientas utilizadas en el entorno industrial, empresarial, gubernamental o académico, que estén clasificadas dentro de la programación de sistemas. 1.3 Desarrollar un ensayo acerca de los traductores que se utilizan con mayor frecuencia en nuestro tiempo. 1.4 Buscar generadores de código que sirvan para desarrollar compiladores.	1, 2, 4, 9, 12, 13

**UNIDAD 2.-** Introducción al diseño de los lenguajes de programación.

<b>Objetivo Educativo</b>	<b>Actividades de Aprendizaje</b>	<b>Fuentes de Información</b>
Conocerá las bases que deben tomar en cuenta para el buen diseño de un lenguaje de programación	<p>2.1 Definir aspectos a considerar en el diseño de lenguajes de programación como: la comunicación humana, prevención y detección de errores, usabilidad, programación efectiva, eficiencia, compilabilidad, independencia de la máquina, simplicidad, uniformidad, ortogonalidad, generalización y especialización.</p> <p>2.2 Discutir en grupo las condiciones que determinan la funcionalidad del lenguaje tales como: Microestructuras, estructura de las expresiones, estructuras de datos, estructuras de control, estructuras de compilador, estructuras para entradas y salidas.</p> <p>2.3 Analizar un lenguaje computacional como caso tipo y desarrollar un ensayo acerca de las características que lo define, basándose en los puntos vistos como parte del diseño de un lenguaje de programación.</p> <p>2.4 Proponer y diseñar un lenguaje prototipo.</p>	13

**UNIDAD 3.-** Análisis Léxico.

<b>Objetivo Educativo</b>	<b>Actividades de Aprendizaje</b>	<b>Fuentes de Información</b>
Construirá un analizador léxico a partir de un generador de código compilador de compilador	<p>3.1 Considerar como base de esta unidad lo visto en Teoría de la Computación acerca de autómatas finitos determinísticos y expresiones regulares aplicándolo en ejercicios tipos.</p> <p>3.2 Buscar y seleccionar información sobre la construcción de un Analizador Léxico.</p> <p>3.3 Construir un Analizador de Léxico,</p>	1, 2, 3, 5, 6, 7, 8, 9, 10, 11, 13, 14. [1], [2], [3], [5], [6], [7]

	<p>considerando el reconocimiento de tokens, la administración de un espacio temporal de memoria (buffers), la construcción de la tabla de símbolos, así como el manejo de errores.</p> <p>3.4 Desarrollar un analizador de léxico, aplicado a un lenguaje prototipo o comercial, utilizando un generador de código para esta etapa del proceso como puede ser Lex o Flex, entre otros.</p>	
--	---	--

#### UNIDAD 4.- Análisis sintáctico.

<b>Objetivo Educativo</b>	<b>Actividades de Aprendizaje</b>	<b>Fuentes de Información</b>
Implementará un analizador sintáctico a partir de un generador de código compilador de compilador	<p>4.1 Considerar como base de esta unidad lo visto en Teoría de la Computación acerca de lenguajes libres de contexto y los árboles de derivación a través de ejercicios tipo.</p> <p>4.2 Construir diagramas de sintaxis a partir de gramáticas.</p> <p>4.3 Agregar dentro de la gramática del lenguaje la precedencia de operadores.</p> <p>4.4 Construir un analizador sintáctico bajo la metodología TOP-DOWN y BOTTOM-UP.</p> <p>4.5 Aplicar técnicas para la detección y recuperación de errores en la etapa de sintaxis.</p> <p>4.6 Integrar la etapa del léxico dentro del desarrollo del compilador.</p> <p>4.7 Desarrollar un analizador de sintáctico, aplicado a un lenguaje prototipo o comercial, utilizando un generador de código para esta etapa del proceso como puede ser Yacc o Bison, entre otros.</p>	<p>1, 2, 3, 5, 6, 7, 8, 9, 10, 11, 13, 14. [r], [2], [3], [4], [6], [8]</p>

**UNIDAD 5.- Análisis semántico.**

<b>Objetivo Educativo</b>	<b>Actividades de Aprendizaje</b>	<b>Fuentes de Información</b>
Implementará un analizador sintáctico a partir de un generador de código compilador de compilador	5.1 Reconocer el manejo de tipos en las expresiones y el uso de operadores. 5.2 Establecer las reglas para la conversión de tipos (casting) en expresiones. 5.3 Agregar acciones semánticas a la estructura de la gramática, tanto en metodología TOP-DOWN como BOTTOM-UP. 5.4 Manejar la tabla de símbolos a fin de administrar el almacenamiento de información con su tipo. 5.5 Detectar los errores semánticos y la recuperación de los mismos.	1, 2, 3, 5, 6, 7, 8, 9, 10, 11, 13, 14.

**UNIDAD 6.- Generación de código intermedio.**

<b>Objetivo Educativo</b>	<b>Actividades de Aprendizaje</b>	<b>Fuentes de Información</b>
Desarrollará una máquina virtual que ejecute un código intermedio a partir del código fuente del lenguaje prototipo.	6.1 Reconocer el manejo de tipos en las expresiones y el uso de operadores. 6.2 Realizar ejercicios con los diferentes tipos de notaciones para la conversión de expresiones: Infija, prefija y posfija. 6.3 Realizar ejercicios de técnicas básicas para la representación de un código intermedio: Código P, Triplos y Cuádruplos. 6.4 Desarrollar las acciones que representen la estructura de un lenguaje de programación de alto nivel en un código intermedio. 6.5 Aplicar las acciones construidas a la gramática del lenguaje prototipo.	1, 2, 3, 5, 6, 7, 8, 9, 10, 11, 13, 14.

## UNIDAD 7.- Optimización.

<b>Objetivo Educativo</b>	<b>Actividades de Aprendizaje</b>	<b>Fuentes de Información</b>
Aplicará las técnicas de optimización de código.	<p>7.1 Aplicar las técnicas para la optimización de código sobre el código intermedio generado, evaluando los criterios de tiempo de ejecución o extensión de código generado.</p> <p>7.2 Buscar nuevas técnicas para la optimización de código, sobre todo para aquellos lenguajes que requieren de una máquina virtual para su ejecución sobre multiplataformas.</p> <p>7.3 Escribir un ensayo que establezca las tendencias y técnicas empleadas para este propósito.</p>	1, 2, 3, 5, 6, 7, 8, 9, 10, 11, 13, 14.

## UNIDAD 8.- Generación de código objeto.

<b>Objetivo Educativo</b>	<b>Actividades de Aprendizaje</b>	<b>Fuentes de Información</b>
Utilizará el lenguaje de bajo nivel para llevar el código construido en un lenguaje de alto nivel a un lenguaje entendible por la máquina para su ejecución	<p>8.1 Proporcionar y propiciar la investigación bibliográfica del lenguaje máquina.</p> <p>8.2 Conocer las características de un lenguaje ensamblador.</p> <p>8.3 Llevar a cabo dinámicas grupales para recordar las características principales del lenguaje ensamblador, a fin de llevar un código intermedio, independiente de la máquina a un código reconocido por el hardware.</p> <p>8.4 Conocer las técnicas de administración de memoria para el almacenamiento de un programa en momento de ejecución.</p>	1, 2, 3, 5, 6, 7, 8, 9, 10, 11, 13, 14.

## 10. FUENTES DE INFORMACIÓN

1. Aho, Sethi, Ullman.  
Compiladores Principios, técnicas y herramientas  
Ed. Addison Wesley.
2. Karen A. Lemone.  
Fundamentos de compiladores Cómo traducir al lenguaje de  
computadora.  
Ed. Compañía Editorial Continental.
3. Jesús Salas Parrilla.  
Sistemas Operativos y Compiladores.  
Ed. McGraw Hill.
4. Beck,.  
Software de Sistemas, Introducción a la programación de Sistemas  
Ed. Addison-Wesley Iberoamericana.
5. Teufel, Schmidt, Teufel.  
Compiladores Conceptos Fundamentales.  
Ed. Addison-Wesley Iberoamericana.
6. Kenneth C. Loudon.  
Construcción de compiladores Principios y práctica.  
Ed. Thomson.
7. Kenneth C. . Loudon  
Lenguajes de programación Principios y práctica.  
Ed. Thomson.
8. Guillermo Levine Gutiérrez.  
Computación y programación moderna Perspectiva integral de la  
informática.  
Ed. Pearson Educación.
9. Ronald Mak.  
Writing compilers and interpreters.  
Ed. Wiley Computer Publishing.
10. Fischer, LeBlanc.  
Crafting a compiler with C.  
Ed. Cummings Publishing Company, Inc.
11. Thomas Pittman, James Peters.  
The art of compiler design Theory and practice  
Ed. Prentice Hall.

12. Peter Abel.  
Lenguaje ensamblador y programación para PC IBM y compatibles.  
Ed. Pearson Educación.
13. Temblay & Sorenson.  
Compilers Writing.  
Ed. Mc Graw Hill.
14. John R. Levine, Tony Mason, Doug Brown.  
Lex y Yacc.  
Ed. O'Reilly & Associates.

### Referencias en Internet

- [1] The Lex & Yacc Page, 3-mar-04, 12:45  
<http://dinosaur.compilertools.net>
- [2] A compact guide to lex & Yacc, Thomas Niemann, 3-Mar-04, 12:50  
<http://epaperpress.com/lexandyacc>
- [3] Lex & Yacc HOWTO, Bert Hubert (PowerDNS.COM.BV), 3-Mar-04, 12:55  
[http://ds9a.nl/lex\\_yacc](http://ds9a.nl/lex_yacc)
- [4] Bison, 3-Mar-04, 13:00  
<http://www.gnu.org/software/bison/bison.html>
- [5] Flex, 3-Mar-04, 13:02  
<http://www.gnu.org/software/flex/flex.html>
- [6] Compiler Construction using Flex and Bison, Anthony Aaby, 3-mar-04, 13:05  
<http://cs.wvc.edu/aabyan/464/Book/>
- [7] Flex, versión 2.5 A fast scanner generator, Edition 2.5, March 1995, Vern Paxson, 3-Mar-04,13:10  
[http://www.cs.princeton.edu/appel/modern/c/software/flex/flex\\_toc.html](http://www.cs.princeton.edu/appel/modern/c/software/flex/flex_toc.html)
- [8] Bison, The Yacc-compatible Parser Generator, November 1995, Bison Version 1.5, Charles Donnelly and Richard Stallman, 3-Mar-04,13:10  
[http://www.cs.princeton.edu/appel/modern/c/software/bison/bison\\_toc.html](http://www.cs.princeton.edu/appel/modern/c/software/bison/bison_toc.html)

## 11. PRÁCTICAS

### Unidad Práctica

- 1 Desarrollar un compilador para un lenguaje prototipo o lenguaje comercial, para ser concluido en, máximo un semestre
- 2 Desarrollar herramientas de software de base como editores, procesadores de texto, hojas de cálculo.
- 3 Desarrollar un manejador de bases de datos, con una estructura básica.
- 4 Desarrollar interpretadores de comandos para herramientas gráficas.