1.- DATOS DE LA ASIGNATURA

Nombre de la asignatura:	**Lenguajes y Autómatas II.**
Carrera:	**Ingeniería en Sistemas Computacionales**
Clave de la asignatura:	SCD-1016
(Créditos) (SATCA[44])	2 – 3 – 5

2.- PRESENTACIÓN

Caracterización de la asignatura

En esta asignatura se debe desarrollar el análisis semántico, la generación de código, la optimización y la generación del código objeto para obtener el funcionamiento de un compilador.

Esta asignatura busca proveer al estudiante de herramientas, conocimientos y habilidades necesarias para desarrollar un compilador con base en los conocimientos previos de la asignatura lenguajes y autómatas I. La aportación de esta materia es relevante en el ámbito del desarrollo de software de sistemas.

Es indispensable distinguir que la carrera de Ingeniería en Sistemas Computacionales se basa no sólo en el desarrollo de software comercial y administrativo, sino también en el desarrollo de software científico y para el desarrollo tecnológico. Esta materia se ubica en la segunda categoría y es indispensable desarrollar software en estos campos para preparar a los egresados y tengan la posibilidad de cursar posgrados de alto nivel.

La asignatura trata de concretar un traductor iniciado en la materia previa para que el estudiante comprenda que es capaz, mediante técnicas bien definidas, de crear su propio lenguaje de programación.

La aportación de la asignatura al perfil del egresado será específicamente la siguiente:

- Desarrollar, implementar y administrar software de sistemas o de aplicación que cumpla con los estándares de calidad buscando como finalidad apoyar la productividad y competitividad de las organizaciones.
- Integrar soluciones computacionales con diferentes tecnologías, plataformas o dispositivos.
- Diseñar e implementar interfaces hombre – máquina y maquina – máquina para la automatización de sistemas.
- Identificar y comprender las tecnologías de hardware para proponer, desarrollar y mantener aplicaciones eficientes.

Intención didáctica.

La materia consta de cuatro bloques estructurados y definidos que abarcan la última etapa de la fase de análisis y síntesis. Al término del semestre se debe obtener un compilador o traductor completo, funcionando de acuerdo a ciertas restricciones y requisitos.

La primera unidad se centra totalmente en el analizador semántico, por lo que el analizador sintáctico debió ser concluido en la materia de lenguajes y autómatas I, ya que servirá de base en esta unidad.

En la segunda unidad se analizan las técnicas para generar código intermedio, para incluirse en su proyecto.

La tercera unidad se centra en la optimización del código. Es importante hacer notar que de esta fase depende la buena y eficiente ejecución del código objeto.

En el último bloque se aborda el tema de la generación de código objeto. Como paso final, es importante que el código resultante sea eficiente y pueda correr directamente sobre la computadora en lenguaje ensamblador o basándose en microinstrucciones.

3.- COMPETENCIAS A DESARROLLAR

Competencias específicas: Desarrollar software de base: traductor, intérprete o compilador.	**Competencias Genéricas:** **Competencias instrumentales** • Capacidad de análisis y síntesis • Conocimientos básicos de la carrera • Comunicación oral y escrita • Conocimiento de una segunda lengua • Conocimiento generales básicos del lenguaje ensamblador. • Habilidad para buscar y analizar información proveniente de fuentes diversa. • Habilidad lógica para solucionar problemas • Habilidades del manejo de la computadora **Competencias interpersonales** • Capacidad crítica y autocrítica • Trabajo en equipo interdisciplinario • Habilidades interpersonales **Competencias sistémicas** • Capacidad de aplicar los conocimientos en la práctica • Habilidades de investigación • Estándares de desarrollo para la implementación de soluciones • Capacidad de aprender • Capacidad de generar nuevas ideas (creatividad) • Habilidad para trabajar en forma autónoma • Capacidad para diseñar y gestionar proyectos • Búsqueda del logro

4.- HISTORIA DEL PROGRAMA

Lugar y fecha de elaboración o revisión	**Participantes**	**Observaciones (Cambios y justificación**

Instituto Tecnológico de Fecha	Representantes de los Institutos Tecnológicos de:	Reunión nacional de Diseño e innovación curricular de la carrera de Ingeniería en
Institutos Tecnológicos Superiores de: Coatzacoalcos, Occidente del Estado de Hidalgo, Teziutlán y Lerdo y I.T. de Toluca. Fecha 12 de Octubre 2009 al 19 de Febrero de 2010	Representante de la Academia de Sistemas Computacionales	Análisis, enriquecimiento y elaboración del programa de estudio propuesto en la Reunión Nacional de Diseño Curricular de la carrera de
Instituto Tecnológico de fecha	Representantes de los Institutos Tecnológicos participantes en el diseño de la carrera de Ingeniería	Reunión nacional de consolidación de la carrea de ingeniería en

5.- OBJETIVO(S) GENERALE(S) DEL CURSO (Competencia específica a desarrollar en el curso)

Desarrollar software de base: traductor, intérprete o compilador.

6.- COMPETENCIAS PREVIAS

Definir, diseñar, construir y programar las fases del analizador léxico y sintáctico de un traductor o compilador.

7.- TEMARIO

Unidad	Temas	Subtemas
1	Análisis semántico	1.1. Arboles de expresiones. 1.2. Acciones semánticas de un analizador sintáctico. 1.3. Comprobaciones de tipos en expresiones . 1.4. Pila semántica en un analizador sintáctico. 1.5. Esquema de traducción. 1.6. Generación de la tabla de símbolo y de direcciones. 1.7. Manejo de errores semánticos.
2	Generación de código intermedio.	2.1 Notaciones 2.1.1 Prefija 2.1.2 Infija

		2.2.3 Postfija 2.2 Representaciones de código Intermedio. 2.2.1 Notación Polaca 2.2.2 Código P 2.2.3 Triplos 2.2.4 Cuádruplos. 2.3 Esquema de generación. 2.3.1 Variables y constantes. 2.3.2 Expresiones. 2.3.3 Instrucción de asignación. 2.3.4 Instrucciones de control. 2.3.5 Funciones 2.3.6 Estructuras
3	Optimización	3.1 Tipos de optimización. 3.1.1 Locales. 3.1.2 Ciclos. 3.1.3 Globales. 3.1.4 De mirilla. 3.2 Costos. 3.2.1 Costo de ejecución. (memoria, registros, pilas) 3.2.2 Criterios para mejorar el código. 3.2.3 Herramientas para el análisis del flujo de datos.
4	Generación de código objeto.	4.1 Registros. 4.2 Lenguaje ensamblador. 4.3 Lenguaje maquina. 4.4 Administración de memoria.

8.- SUGERENCIAS DIDACTICAS (Desarrollo de competencias genéricas)

El profesor debe

Ser conocedor de la disciplina que está bajo su responsabilidad, desarrollar la capacidad para coordinar el trabajo en grupo, orientar el trabajo del estudiante y potenciar en él la autonomía, el trabajo cooperativo y la toma de decisiones. Tomar en cuenta el conocimiento de los estudiantes como punto de partida y como obstáculo para la construcción de nuevos conocimientos.

- Propiciar actividades de búsqueda, selección y análisis de información en distintas fuentes.
- Facilitar el aprendizaje de manera que el estudiante sea capaz de detectar y recuperar errores semánticos, conocer las notaciones para la conversión de expresiones, conocer como se representa el código intermedio, generar notaciones para la conversión de expresiones, representar el código intermedio utilizando un lenguaje propuesto y utilizar un diagrama de sintaxis para representar acciones.
- Propiciar el desarrollo de actividades intelectuales de inducción-deducción y análisis-síntesis, que encaminen hacia la investigación.
- Propiciar actividades de metacognición. Ante la ejecución de una actividad, señalar o identificar el tipo de proceso intelectual que se realizó: una identificación de patrones, un análisis, una síntesis, la creación de un heurístico, etc. Al principio lo hará el profesor, luego será el estudiante quien lo identifique.
- Desarrollar la capacidad para coordinar y trabajar en equipo; orientar el trabajo del estudiante y potenciar en él la autonomía, el trabajo cooperativo y la toma de decisiones.
- Hacer el seguimiento del proceso formativo y propiciar la interacción entre los estudiantes. Proponer investigaciones en diferentes áreas (ciencias sociales, ingeniería, computación, etc), por grupos de interés
- Para promover el desarrollo de capacidades de expresión oral y escrita en los estudiantes, se les invita a que presenten un proyecto de asignatura que incluya los aspectos relevantes de su proyecto. El proyecto incluye una presentación escrita y una oral. Todos los integrantes de cada grupo de trabajo deben participar para incentivar y promover el desarrollo de estas capacidades.
- Promover la interacción directa que permita al estudiante aprender nuevas estructuras de programación y técnicas que usan los lenguajes para procesar información.
- Facilitar el contacto directo con lenguajes y herramientas, para contribuir a la formación de las competencias para el trabajo.
- Desarrollar actividades de aprendizaje que propicien la aplicación de los conceptos, modelos y metodologías que se van aprendiendo en el desarrollo de la asignatura.
- Proponer problemas que permitan al estudiante la integración de contenidos de la asignatura y entre distintas asignaturas, para su análisis y solución.
- Propiciar el uso de las nuevas tecnologías en el desarrollo de la asignatura.

9.- SUGERENCIAS DE EVALUACION

El proceso de evaluación debe ser continuo y formativo por lo que se debe considerar el desempeño en las siguientes actividades:

- Aplicar evaluaciones diagnósticas.
- Desarrollar proyectos usando un lenguaje de programación, donde se apliquen los temas previamente vistos para la construcción de las fases del analizador semántico, código intermedio, optimización y generación de código objeto a fin de construir un compilador.
- Realizar investigaciones documentales referentes a la asignatura usando los diferentes medios bibliográficos o electrónicos, para desarrollar posteriormente: cuadros comparativos, mapas conceptuales, cuadros sinópticos, resúmenes, ensayos, entre otros.
- Representar, comparar, reflexionar sobre teorías o conceptos.

- Aplicar exámenes teóricos-prácticos para detectar que tanto se ha comprendido del tema analizado.
- Realizar prácticas y ejercicios en los diferentes tópicos de la asignatura.
- Evaluar el desempeño del estudiante en el grupo utilizando instrumentos de autoevaluaciones y coevaluaciones (p.e. rúbricas o listas de cotejo).
- Delimitar las especificaciones de los proyectos.

10.- UNIDADES DE APRENDIZAJE

Unidad 1: Análisis Semántico.

Competencia específica a desarrollar	**Actividades de aprendizaje**
Diseñar mediante el uso de arboles de expresiones dirigida por la sintaxis un analizador semántico para un meta-compilador.	• Detectar y recuperar errores semánticos. • Buscar y seleccionar información sobre la construcción de un Analizador Semántico. • Reconocer el manejo de tipos en las expresiones y el uso de operadores. • Establecer las reglas para la conversión de tipos (casting) en expresiones. • Agregar acciones semánticas a la estructura de la gramática. • Manipular la tabla de conversión de símbolos y de direcciones. • Integrar equipos de trabajo para la Construcción de un analizador Semántico

Unidad 2: Generación de código intermedio.

Competencia específica a desarrollar	**Actividades de aprendizaje**
Aplicar las herramientas para desarrollar una máquina virtual que ejecute código intermedio a partir del código fuente de un lenguaje prototipo.	• Aplicar los tipos de notación para la conversión de expresiones: Infija, prefija y posfija. • Representar expresiones mediante el código intermedio. • Reconocer el manejo de tipos en las expresiones y el uso de operadores. • Desarrollar las acciones que representen la estructura de un lenguaje de programación de alto nivel en un código intermedio. • Aplicar las acciones construidas a la gramática del lenguaje prototipo. • Evaluar el prototipo completo construyendo algunos programas tipo usando la gramática definida.

Unidad 3: Optimización.

Competencia específica a desarrollar	Actividades de aprendizaje
Conocer e Identificar los diferentes tipos de optimización que permita eficientar el código intermedio.	• Aplicar las técnicas para la optimización del código intermedio generado • Tener nociones algebraicas para estimar el número de veces que se realiza una instrucción dentro de un ciclo o ciclos anidadas. • Conocer que recursos se consumen en invocación a funciones y expresiones simples. • Estudiar nuevas técnicas para la optimización de código, sobre todo para aquellos lenguajes que requieren de una máquina virtual para su ejecución sobre multiplataformas. • Escribir un ensayo que establezca las tendencias y técnicas empleadas para este propósito. • Conocer los criterios de tiempo de ejecución o extensión de código generado. • Integrar equipos, para analizar códigos intermedios existentes y proponer algunas mejoras

Unidad 4: Generación del código objeto.

Competencia específica a desarrollar	Actividades de aprendizaje
Utilizar un lenguaje de bajo nivel para traducir el código construido a lenguaje máquina para su ejecución.	• Conocer la arquitectura de los microprocesadores intel y compatibles • Conocer la estructura y funcionamiento del lenguaje ensamblador. • Conocer las características principales del lenguaje maquina a fin de llevar un código intermedio y este pueda ser reconocido por el hardware. • Conocer las técnicas de administración de memoria para el almacenamiento de un programa en momento de ejecución. • Experimentar con simuladores de arquitectura de microprocesadores.

11.- FUENTES DE INFORMACION

1. Aho, Sethi, Ullman. Compiladores Principios, técnicas y herramientasEd. Addison Wesley.

2. Lemone Karen A. , Fundamentos de compiladores Cómo traducir al lenguaje de computadora, Ed. Compañía Editorial Continental.
3. Kenneth C. Louden. Construcción de compiladores Principios y práctica.Ed. Thomson.
4. Martin John, Lenguajes formales y teoría de la computación, ED. Mc Graw Hill
5. Hopcroft John E., Introducción a la Teoría de Autómatas, Lenguajes y Computación, ED. Addison Wesley
6. Guerra Crespo. Hector. Compiladores. Ed. Tecnologica didáctica.
7. Ronald Mak. Writing compilers and interpreters. Ed. Wiley Computer Publishing.
8. Fischer, LeBlanc. Crafting a compiler with C. Ed. Cummings Publishing Company, Inc.
9. Salas Parrilla, Jesús. Sistemas Operativos y Compiladores. McGraw Hill.
10. Beck. Software de Sistemas, Introducción a la programación de Sistemas. Addison-Wesley Iberoamericana.
11. Teufel, Schmidt, Teufel. Compiladores Conceptos Fundamentales. Addison-Wesley Iberoamericana.
12. C. Louden, Kenneth. Lenguajes de programación Principios y práctica. Thomson.
13. Levine Gutiérrez, Guillermo. Computación y programación moderna Perspectiva integral de la informática. Pearson Educación.
14. Abel, Peter. Lenguaje ensamblador y programación para PC IBM y compatibles. Pearson Educación.
15. Mak, Ronald. Writing compilers and interpreters. Wiley Computer Publishing.
16. Pittman, Thomas, Peters, James. The art of compiler design Theory and practice. Prentice Hall.
17. Temblay & Sorenson. Compilers Writing. Mc Graw Hill.
18. R. Levine, John; Mason, Tony, Brown, Doug. Lex y Yacc. O'Reilly & Associates.
19. The Lex & Yacc Page, 3-mar-04, 12:45, http://dinosaur.compilertools.net
20. A compact guide to lex & Yacc, Thomas Niemann, 3-Mar-04, 12:50, http://epaperpress.com/lexandyacc
21. Lex & Yacc HOWTO, Bert Hubert (PowerDNS.COM.BV), 3-Mar-04, 12:55, http://ds9a.nl/lex_yacc
22. Flex, 3-mar-04, 13:02, http://www.gnu.org/software/flex/flex.html
23. Compiler construction using flex and Bison, Anthony Aaby, 3-mar-04, 13:05, http://cs.wwc.edu/aabyan/464/BooK/
24. Flex, version 2.5 A fast scanner generator, Edition 2.5, March 1995, Vern Paxson, 3-mar-04, 13:10, http://www.cs.princelon.edu/appel/modern/c/software/flex/flex_toc.html
25. 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.princelon.edu/appel/modern/c/software/bison/bison_toc.html**, 13/dic/2009
26. Bison. http://3d2f.com/programs/30-170-microprocessor-emulator-and-assembler-download.shtml, 13/dic/2009
27. 2/Ago/2005 ,Microprocessor Emulator and Assembler 3.10-k, http://software.intel.com/en-us/articles/all/1/, 24/feb/2010

28. Intel, 31/dic/2009, Intel® Software Development EmulatorBottom of Form, http://software.intel.com/en-us/articles/intel-software-development-emulator/, 24/feb/2010

12.- PRACTICAS PROPUESTAS (aquí sólo describen brevemente, aqueda pendiente la descripción con detalle).

- Diseñar y construir el generador de código semántico para el lenguaje del caso de estudio.
- Realizar arboles de expresiones en casos de estudio.
- Realizar conversiones de tipos en expresiones.
- Construir la tabla de símbolos y de direcciones para la gramática propuesta
- Detectar errores de semántica en expresiones dadas.
- Modificar la GLC agregando las acciones semánticas correspondientes.
- Convertir expresiones mediante el uso de notaciones prefijas, infijas y postfijas.
- Definir e implementar la notación que más se ajuste a las estructuras de evaluación de expresiones de lenguaje.
- Proponer una estructura de código intermedio en base a las características propias de cada lenguaje.
- Desarrollar esquemas de generación de código intermedio
- Definir y construir el generador de código intermedio para su caso de estudio.
- Agregar acciones de representación intermedia al lenguaje de programación propuesto.
- Saber cuántos recursos y cuánto tiempo consume cada instrucción de código intermedio
- Evaluar el código intermedio generado para los programas escritos en el lenguaje de su caso de estudio y si aplica realizar la optimización correspondiente.
- Poder establecer una equivalencia entre las instrucciones del lenguaje intermedio y las instrucciones en ensamblador.
- Diseñar y construir el generador de código máquina u objeto para el lenguaje del caso de estudio.