



NATIONAL AUTONOMOUS UNIVERSITY OF MEXICO
SCHOOL OF ENGINEERING



COURSE SYLLABUS

COMPILERS

0443

7

8

Course

Code

Semester

Credits

**ELECTRICAL
ENGINEERING**

COMPUTER ENGINEERING

COMPUTER ENGINEERING

Division

Department

Undergraduate Program

Course:

Hours /week:

Hours / Semester:

Compulsory

Lecture

Lecture

Elective

Practical

Practical

Total

Total

Mode: Lecture-based Course.

Prerequisite course: Formal Languages and Automata

Subsequent course: None

Course Objective(s) The student will design translators as tools for the use and development of software systems; as well as differentiate between existing translators to create efficient software tailored to the type of problem to be solved.

Course Topics

NO.	NAME.	HOURS
1.	General Overview	10.0
2.	Lexical Analysis	8.0
3.	Syntactic Analysis	4.0
4.	Top-Down Syntactic Analysis	8.0
5.	Bottom-Up Syntactic Analysis	10.0
6.	Syntax-Directed Translation	8.0
7.	Memory Organization at Runtime	4.0
8.	Intermediate Code Generation and Semantic Analysis	8.0
9.	Optimization and Code Generation	4.0

64.0

Practical Activities

0.0

Total

64.0

1. General Overview

Objective: The student will identify the role of translators as tools for the use and development of software systems, and will also distinguish the different areas of work of these tools.

Content:

1.1 Review of the Origin and Evolution of Language-Translators

1.1.1 History of Languages and Translators

1.1.2 T Diagrams in Translator Development

1.2 Classification of Translators

1.2.1 Comparative Analysis of Interpreters, Compilers, and Hybrids

1.2.2 Differences Between Programming Languages, Markup Languages, and Scripts

1.2.3 Translation Problems

1.3 Stages in the Compilation Process

1.3.1 General Overview of Each Stage in the Compilation Process

1.3.2 Identification of Errors in Each Stage

1.4 Execution Environments

1.4.1 Machine-Dependent and Machine-Independent Translators

1.4.2 Virtual Machines

2. Lexical Analysis

Objective: The student will build a lexical analyzer from the definition of lexical component classes.

Content:

2.1 Functions of a Lexical Analyzer

2.2 Identification of Lexical Classes

2.3 Structure of Symbol Tables

2.4 Handling Lexical Errors

2.5 Programming a Lexical Analyzer (Scanner)

2.6 Automatic Generation of Lexical Analyzers

3. Syntactic Analysis

Objective: The student will explain in detail the syntactic analysis stage in the compilation process, as well as the grammars suitable for defining the structure of programming languages.

Content:

3.1 Grammars Suitable for Syntactic Analysis

3.2 Representation of Syntax

3.2.1 BNF Notation

3.2.2 Railroad Diagrams

3.3 Classification of Syntactic Analyzers

4. Top-Down Syntactic Analysis

Objective: The student will build a top-down syntactic analyzer based on a grammar suitable for this type of syntactic analysis.

Content:

- 4.1 LL Grammars
- 4.2 Building the Parser Table for Top-Down Analysis
- 4.3 Handling Syntactic Errors
- 4.4 Building a Recursive Top-Down Syntactic Analyzer

5. Bottom-Up Syntactic Analysis

Objective: The student will build a bottom-up syntactic analyzer based on a grammar suitable for this type of syntactic analysis.

Content:

- 5.1 LR Grammars
- 5.2 SLR(1) Analyzer
- 5.3 LR(1) Analyzer
- 5.4 LALR(1) Analyzer
- 5.5 Error Detection and Recovery
- 5.6 LALR(1) Syntax Analyzer Generators (YACC)

6. Syntax-Directed Translation

Objective: The student will modify grammars and perform syntax-directed translation in both top-down and bottom-up analyzers.

Content:

- 6.1 Translation Grammars
- 6.2 Handling the Symbol Table
- 6.3 Syntax-Directed Translation in Top-Down Analyzers
- 6.4 Syntax-Directed Translation in Bottom-Up Analyzers

7. Memory Organization at Runtime

Objective: The student will analyze the structures for managing memory during program execution.

Content:

- 7.1 Translation Grammars
- 7.2 Different Types of Organizations
- 7.3 Stack Organization
- 7.4 Heap Organization
- 7.5 Parameter Passing

8. Intermediate Code Generation and Semantic Analysis

Objective: The student will describe different types of intermediate codes and perform semantic analysis for compiler creation.

Content:

- 8.1 Attributes and Grammars with Attributes
- 8.2 Attribute Handling Algorithms
- 8.3 Intermediate Languages
- 8.4 Type and Declaration Checking
- 8.5 Generation of Intermediate Code for Different Statements

9. Optimization and Code Generation

Objective: The student will apply various techniques to optimize and generate codes.

Content:

9.1 Main Sources for Optimization

9.2 Optimization of Basic Blocks

9.3 Flow Graphs

9.4 Object Machine

9.5 Basic Code Generation Techniques



UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
FACULTAD DE INGENIERÍA



PROGRAMA DE ESTUDIO

COMPILADORES

0434

7

8

Asignatura

Clave

Semestre

Créditos

INGENIERÍA ELÉCTRICA

**INGENIERÍA
EN COMPUTACIÓN**

**INGENIERÍA
EN COMPUTACIÓN**

División

Departamento

Licenciatura

Asignatura:

Obligatoria

Optativa

Horas/semana:

Teóricas

Prácticas

Total

Horas/semestre:

Teóricas

Prácticas

Total

Modalidad: Curso teórico

Seriación obligatoria antecedente: Lenguajes Formales y Automatas

Seriación obligatoria consecuente: Ninguna

Objetivo(s) del curso:

El alumno diseñará los traductores como herramientas de uso y desarrollo de sistemas de software; como también diferenciará traductores existentes para elaborar software eficiente y adecuado al tipo de problema por resolver.

Temario

NÚM.	NOMBRE	HORAS
1.	Panorama general	10.0
2.	Análisis léxico	8.0
3.	Análisis sintáctico	4.0
4.	Análisis sintáctico descendente	8.0
5.	Análisis sintáctico ascendente	10.0
6.	Traducción dirigida por sintaxis	8.0
7.	Organización de memoria en tiempo de corrida	4.0
8.	Generación de código intermedio y análisis semántico	8.0
9.	Optimización y generación de código	4.0
		64.0
	Actividades prácticas	0.0
	Total	64.0

1 Panorama general

Objetivo: El alumno identificará el papel de los traductores como herramientas de uso y desarrollo de sistemas de software, además de distinguir las diferentes áreas de trabajo de estos.

Contenido:

- 1.1 Reseña del origen y evolución de los lenguajes-traductores.
 - 1.1.1 Historia de lenguajes y traductores.
 - 1.1.2 Diagramas T en el desarrollo de traductores.

- 1.2 Clasificación de los traductores.
 - 1.2.1 Análisis comparativo de intérpretes, compiladores e híbridos.
 - 1.2.2 Diferencias entre lenguajes de programación, de marcado y scripts.
 - 1.2.3 Problemas de traducción.

- 1.3 Etapas en el proceso de compilación.
 - 1.3.1 Generalidades de cada etapa en el proceso de compilación.
 - 1.3.2 Identificación de errores en cada etapa.

- 1.4 Ambientes de ejecución.
 - 1.4.1 Traductores dependientes e independientes de la máquina.
 - 1.4.2 Máquinas virtuales.

2 Análisis léxico

Objetivo: El alumno construirá un analizador léxico a partir de la definición de clases de componentes léxicos.

Contenido:

- 2.1 Funciones de un analizador léxico.
- 2.2 Identificación de clases léxicas.
- 2.3 Estructura de las tablas de símbolos.
- 2.4 Manejo de errores léxicos.
- 2.5 Programación de un analizador léxico (scanner).
- 2.6 Generación automática de analizadores léxicos.

3 Análisis sintáctico

Objetivo: El alumno explicará a detalle la etapa del análisis sintáctico en el proceso de compilación, así como las gramáticas idóneas para la definición de la estructura de los lenguajes de programación.

Contenido:

- 3.1 Gramáticas idóneas para análisis sintáctico.
- 3.2 Representación de sintaxis.
 - 3.2.1 Notación BNF.
 - 3.2.2 Diagramas de tren.

- 3.3 Clasificación de los analizadores sintácticos.

4 Análisis sintáctico descendente

Objetivo: El alumno construirá un analizador sintáctico descendente a partir de una gramática adecuada para este tipo de análisis sintáctico.

Contenido:

- 4.1 Gramáticas LL.
- 4.2 Construcción de la tabla de Parser para un análisis descendente.
- 4.3 Manejo de errores sintácticos.
- 4.4 Construcción de un analizador sintáctico descendente recursivo.

5 Análisis sintáctico ascendente

Objetivo: El alumno construirá un analizador sintáctico ascendente a partir de una gramática adecuada para este tipo de análisis sintáctico.

Contenido:

- 5.1 Gramáticas LR.
- 5.2 Analizador SLR(1).
- 5.3 Analizador LR(1).
- 5.4 Analizador LALR(1).
- 5.5 Detección y recuperación de errores.
- 5.6 Generadores de analizadores de sintaxis LALR(1) YACC.

6 Traducción dirigida por sintaxis

Objetivo: El alumno modificará gramáticas y realizará la traducción dirigida por la sintaxis en analizadores descendentes y ascendentes.

Contenido:

- 6.1 Gramáticas de traducción.
- 6.2 Manejo de la tabla de símbolos.
- 6.3 Traducción dirigida por sintaxis en analizadores descendentes.
- 6.4 Traducción dirigida por sintaxis en analizadores ascendentes.

7 Organización de memoria en tiempo de corrida

Objetivo: El alumno analizará las estructuras para manejar la memoria en el momento de ejecución del programa.

Contenido:

- 7.1 Gramáticas de traducción.
- 7.2 Diferentes tipos de organizaciones.
- 7.3 Organización de pila o stack.
- 7.4 Organización de heap.
- 7.5 Paso de parámetros.

8 Generación de código intermedio y análisis semántico

Objetivo: El alumno describirá los diferentes tipos de código intermedio y el análisis semántico para la creación de compiladores.

Contenido:

- 8.1 Atributos y gramáticas con atributos.
- 8.2 Algoritmos de manejo de atributos.
- 8.3 Lenguajes intermedios.
- 8.4 Revisión de tipos y declaraciones.
- 8.5 Generación de código intermedio de diferentes sentencias.

9 Optimización y generación de código

Objetivo: El alumno aplicará las diferentes técnicas para optimizar y generar código.

Contenido:

- 9.1 Principales fuentes para la optimización.

- 9.2 Optimización de bloques básicos.
- 9.3 Grafos de flujo.
- 9.4 La máquina objeto.
- 9.5 Técnicas básicas de generación de código.

Bibliografía básica
Temas para los que se recomienda:

AHO, Alfredo, SETHI, Ravi, et al. <i>Compiladores. Principios, técnicas y herramientas</i> Addison-Wesley Iberoamericana, 2000	Todos
LOUDEN, Kenneth <i>Compiler Construction. Principles and Practice</i> Thompson Learning, 1997	Todos
PITTMAN, Thomas, PETERS, James <i>The Art of Compiler Design; Theory and Practice</i> Pearson Education, 1991	Todos
TREMBLAY, Jean-paul, SORENSON, Paul <i>The Theory and Practice of Compiler Writing</i> Mc. Graw-Hill, 1985	Todos

Bibliografía complementaria**Temas para los que se recomienda:**

BENNETT, J. P. <i>Introduction to Compiling Techniques. A first Course using</i> <i>Ansi C, LEX and YACC</i> Mc. Graw-Hill - Book Company Europe, 1996	Todos
KAPLAN, Randy <i>Constructing Language Processor for Little Languages</i> Wiley, 1994	Todos
LEVINE, Jhon, MASON, Tony, et al. <i>Lex y Yacc</i> 2nd edition O Reilly, 1992	2, 5
MAK, Ronald <i>Writing Compilers and Interprets</i> 2nd edition Willey, 1996	Todos
PRATT, Terrence, ZELKOWITZ, Marvin <i>Lenguajes de Programación. Diseño e Implementación</i>	1

Prentice Hall, 1998

SCOTT, Michael

Programming Language Pragmatics

Morgan Kaufmann, 2000

Todos

Sugerencias didácticas

Exposición oral	<input checked="" type="checkbox"/>
Exposición audiovisual	<input checked="" type="checkbox"/>
Ejercicios dentro de clase	<input checked="" type="checkbox"/>
Ejercicios fuera del aula	<input checked="" type="checkbox"/>
Seminarios	<input type="checkbox"/>
Uso de software especializado	<input type="checkbox"/>
Uso de plataformas educativas	<input type="checkbox"/>

Lecturas obligatorias	<input checked="" type="checkbox"/>
Trabajos de investigación	<input checked="" type="checkbox"/>
Prácticas de taller o laboratorio	<input checked="" type="checkbox"/>
Prácticas de campo	<input type="checkbox"/>
Búsqueda especializada en internet	<input type="checkbox"/>
Uso de redes sociales con fines académicos	<input type="checkbox"/>

Forma de evaluar

Exámenes parciales	<input checked="" type="checkbox"/>
Exámenes finales	<input checked="" type="checkbox"/>
Trabajos y tareas fuera del aula	<input checked="" type="checkbox"/>

Participación en clase	<input checked="" type="checkbox"/>
Asistencia a prácticas	<input checked="" type="checkbox"/>

Perfil profesiográfico de quienes pueden impartir la asignatura

Licenciatura en Ingeniería en Computación, Ciencias de Computación, Matemáticas Aplicadas o una carrera similar. Deseable haber realizado estudios de posgrado, contar con conocimientos y experiencia en el área de Ciencias de la Computación, contar con experiencia docente o haber participado en cursos o seminario de iniciación en la práctica docente.