

Programa del curso IC-5701

Compiladores e Intérpretes

Escuela de Computación

Carrera de Ingeniería de Computación, Plan 410.

I Parte. Aspectos relativos al plan de estudios

1 Datos generales

Nombre del curso:	Compiladores e Intérpretes
Código:	IC-5701
Tipo de curso:	Teórico - Práctico
Nº de créditos:	4
Nº horas de clase por semana:	4
Nº horas extraclase por semana:	8
Ubicación en el plan de estudios:	Curso del 5to semestre de la carrera de Ingeniería en Computación
Requisitos:	IC4700 Lenguajes de Programación
Correquisitos:	Ninguno
El curso es requisito de:	IC6600 Principios de Sistemas Operativos
Asistencia:	Obligatoria
Suficiencia:	No
Posibilidad de reconocimiento:	Si
Vigencia del programa:	II semestre 2013.

2 Descripción General

Este curso estudia principios y técnicas necesarios para la construcción de procesadores de lenguajes de programación, con énfasis en compiladores e intérpretes. El aprendizaje permite aplicar modelos abstractos (lenguajes formales y autómatas), enfrentar problemas de manipulación de información simbólica, realizar programación modular avanzada, analizar sistemas complejos de software y hacer modificaciones sistemáticas, y ampliar el repertorio de métodos para resolución de problemas informáticos.

3 Objetivos

Objetivo General

Aplicar principios, modelos y técnicas para el diseño y la construcción de procesadores de lenguajes de programación de alto nivel, con énfasis en compiladores e intérpretes.

Objetivos Específicos

- Identificar los principales problemas relacionados con la implementación de lenguajes de programación.
- Comprender principios y métodos para implementar lenguajes de programación.
- Conocer las tareas que deben realizarse para traducir un lenguaje de programación a un lenguaje de máquina.
- Comprender modelos abstractos de lenguajes y máquinas formales (gramáticas, expresiones regulares y autómatas) y su aplicabilidad para la implementación de lenguajes de programación.
- Entender técnicas para construir compiladores e intérpretes que procesen lenguajes procedimentales.
- Diseñar y construir, de manera sistemática, un compilador o/y un intérprete para un lenguaje imperativo con estructura de bloques.

4 Contenidos

Introducción a la Compilación (1.5 semanas)

- Especificación de Lenguajes
- Procesadores de Lenguajes
- Conceptos Generales de la compilación
- Máquinas Reales y Abstractas
- Estructura del Proceso de Compilación
- Bootstrapping

Análisis léxico (2.5 semanas)

- Generalidades del análisis léxico
- Proceso del análisis léxico
- Expresiones regulares
- Automatas (DFA, NFA)
- De Expresiones Regulares a Autómatas y de Autómatas a Expresiones Regulares
- Alternativas de implementación de un analizador léxico
- Tratamiento de errores léxicos
- Construcción automática de analizadores léxicos

Análisis sintáctico (3 semanas)

- Proceso de Análisis Sintáctico
- Gramáticas libres de contexto
- Construcción de árboles sintácticos
- Ambigüedad y recursividad en gramáticas
- Autómatas de pila
- Reconocimiento descendente: descenso recursivo, LL(1), First, Follow
- Reconocimiento ascendente: generación de DFA, LR(0), SLR(1)
- Tratamientos de errores sintácticos
- Construcción automática de parsers

Análisis contextual o semántico (1.5 semanas)

- Manejo de alcances de variables y tablas de símbolos
- Validación de tipos.
- Diseño de analizadores de contexto.
- Tratamiento de errores contextuales.

Estructuras en tiempo de ejecución (1.5 semanas)

- Pila Semántica y Registros de activación
- Representación de datos
- Evaluación de expresiones
- Organización de memoria durante la ejecución de un programa
- Acceso a objetos no locales
- Rutinas: paso de parámetros, ligas estáticas, argumentos, recursión

Generación de código (3 semanas)

- Estructuras de Datos para generación de código
- Técnicas básicas de generación de código
- Generación de código para Estructuras de Datos
- Generación de código para Expresiones
- Generación de código para Estructuras de control
- Generación de código para Procedimientos y Funciones

Interpretación (1.5 semanas)

- Interpretación recursiva directa.
- Interpretación de un lenguaje imperativo.
- Interpretación de un lenguaje funcional.
- Interpretación iterativa indirecta (de una máquina abstracta).

Temas avanzados (1.5 semanas)

- Optimización de código.
- Compilación para arquitecturas paralelas y lenguajes para cómputo de alto rendimiento.
- Compilación para sistemas distribuidos.
- Inferencia de tipos
- Utilización de las técnicas de compilación en otras aplicaciones

II Parte. Aspectos operativos

5 Metodología de enseñanza y aprendizaje

El profesor desarrollará de manera magistral los aspectos teóricos y prácticos más relevantes de los diferentes temas. Se complementará con otras lecturas, código ejemplo y extractos de libros y manuales relevantes. Se espera una alta participación de los estudiantes durante las lecciones. Se cubrirá en clases mucho del material clásico de compiladores, con lecturas adicionales asignadas a los estudiantes. Se realizarán frecuentes exámenes cortos, que cubren el material teórico y práctico reciente. Se asignarán tareas cortas individuales. La presentación de las tareas debe ser de calidad profesional.

Un componente esencial del aprendizaje y de la evaluación de este curso son los proyectos programados; el profesor indicará en cuáles lenguajes se podrá realizar la programación y si se usarán programas de base o herramientas. Se puede trabajar en grupos de 2 personas. Los grupos de trabajo requieren la aprobación del profesor. Habrá un mínimo de 3 proyectos.

El profesor sugerirá un libro principal de referencia. Se presupone que el alumno profundiza los temas abordados en clase en ese libro y otras lecturas recomendadas por el profesor.

6 Evaluación

- Asistencia y participación en clase.
- Asignaciones.
- Exámenes cortos.
- Proyectos programados (grupales).
- Exámenes (individuales).

Los estudiantes desarrollarán ejercicios de práctica, pruebas cortas, tareas, proyectos programados, lecturas de estudio y participarán en clase.

Los exámenes deben versar sobre temas relacionados con el desarrollo de cada proyecto.

Los proyectos programados pueden hacerse agrupando algunas etapas de un compilador, por ejemplo:

- Análisis léxico y sintáctico.
- Análisis contextual (o semántico).

- Generación e interpretación de código.

	%
Proyectos programados	60
Exámenes	20
Tareas y pruebas cortas	20
	100

7 Bibliografía

Referencias principales

- Aho, A.; Sethi, R.; Ullman, J. "Compilers: Principles, techniques and tools". Addison-Wesley, 1986.
- Aho, A.; Sethi, R.; Ullman, J. "Compiladores: Principios, técnicas y herramientas". Addison-Wesley Iberoamericana, 1998.
- Aho, A.; Lam, M; Sethi, R.; Ullman, J. "Compilers: Principles, techniques and tools", 2nd. Ed . Addison-Wesley, 2006.
- Fischer; Le Blanc. "Crafting a Compiler with C". Benjamin Cummings, 1991.
- Hopcroft, J.; Motwani,R.; Ullman, J. "Introduction to Automata Theory, Languages and Computation", 3rd. Ed. Addison-Wesley, 2006.
- Loudon, K. "Compiler construction: Principles and practice". PWS Publishing, 1997.
- Watt, D. "Programming language processors". Prentice-Hall, 1993.
- Watt, D.; Brown, D. "Programming language processors in Java". Prentice-Hall, 2000.

Referencias complementarias

- Appel, A.; Ginsburg, M. "Modern Compiler implementation in C". Cambridge University Press, 1998.
- Backhouse, R. "Syntax of programming languages". Prentice-Hall, 1979.
- Barrett et al. "Compiler construction". 2nd. ed. Science Research Associates, 1986.
- Calingaert, P. "Assemblers, compilers and program translation". Computer Science Press, 1979.

- Leroy, X.; Weis, P. "Manuel de référence du langage Caml". InterEditions, 1993.
- Leroy. "The Caml Light system, release 0.6, Documentation and user's manual". INRIA, 1993.
- Levine; Mason; Brown. "Lex & Yacc". O'Reilly & Associates, Inc, 1992.
- Mauny. "Functional programming with Caml Light". INRIA, 1991.
- Paulson. "ML for the working programmer", 2nd. Ed. Cambridge University Press, 1996.
- Pratt. "Lenguajes de Programación". Prentice-Hall, 1996.
- Reade. "Elements of functional programming". Addison-Wesley, 1989.
- Teufel; Schmidt; Teufel. "Compiladores: conceptos fundamentales". Addison-Wesley Iberoamericana, 1995.
- Waite; Goos. "Compiler construction". Springer-Verlag, 1984.
- Weis; Leroy. "Le langage Caml". InterEditions, 1993.
- Welsh, McKeag. "Structured system programming". Prentice-Hall, 1981
- Wirth. "Algorithms + Data Structures = Programs". Prentice-Hall, 1976.

8 Profesor

Ing. Diego Munguía

Email: dmunguia@itcr.ac.cr

Página: <http://www.ic-itcr.ac.cr/~dmunguia/pages/ic-5701.html>

Twitter:

Horario y lugar de consulta: Martes y Jueves de 11:00 - 13:00

Diego Munguía Molina tiene estudios de Ingeniería en Computación (ITCR) y es egresado de la maestría en Ciencias Cognoscitivas (UCR), actualmente se encuentra desarrollando su tesis de graduación. Profesionalmente se ha desarrollado como arquitecto de software, acumulando experiencia en ingeniería de software, computación de alto rendimiento e integración de sistemas. Ha laborado como docente para la institución desde el 2012 en la Escuela de Computación impartiendo los cursos de Introducción a la Programación, Taller de Programación, Lenguajes de Programación, Compiladores e Intérpretes, Programación Orientada a Objetos y Diseño de Software.