UNIVERSIDAD NACIONAL DE CÓRDOBA
Facultad de Ciencias Exactas, Físicas y Naturales
República Argentina

Programa de:

Práctica y Construcción de Compiladores

Carrera: *Ingeniería en Computación* Escuela: *Escuela de Ingeniería en Computación* Departamento: *Computación*.	Plan: *281-05* Carga Horaria: ***72*** Semestre: *Décimo* Carácter: *Optativa* Bloque: *Tec. Aplicadas*	Puntos: ***3*** Hs. Semanales: ***4,5*** Año: *Quinto*

Objetivos: Al terminar el curso el alumno:

- *Comprenderá el impacto de la estructura y funcionamiento de un compilador sobre los programas que compila.*
- *Comprenderá técnicas de reconocimiento de patrones.*
- *Diseñará un compilador simple utilizando las técnicas aprendidas en esta materia y a lo largo de su carrera.*
- *Desarrollará habilidades para resolver problemas de procesamiento de información.*

Programa Sintético:

1. *Introducción a los compiladores.*
2. *Análisis Léxico.*
3. *Análisis Sintáctico.*
4. *Análisis Semántico.*
5. *Generación de Código Intermedio.*
6. *Generación y Optimización de Código.*

Programa Analítico: de foja *2* a foja 7.

Programa Combinado de Examen (si corresponde): de foja a foja .

Bibliografía: de foja *7* a foja 7.

Correlativas Obligatorias: Algoritmos y Estructuras de Datos

Correlativas Aconsejadas: Informática Avanzada

Rige: *2014*

Aprobado HCD, Res.: Fecha:	Modificado / Anulado / Sust. HCD Res.: Fecha:

El Secretario Académico de la Facultad de Ciencias Exactas, Físicas y Naturales (UNC) certifica que el programa está aprobado por el (los) número(s) y fecha(s) que anteceden. Córdoba, / / .

Carece de validez sin la certificación de la Secretaría Académica:

PROGRAMA ANALITICO

LINEAMIENTOS GENERALES

En los comienzos de la computación, la programación se realizaba directamente sobre el hardware utilizando codificación binaria o hexadecimal. Como este camino no era viable para desarrollar y mantener software, a principios de los años 50 comienzan los primeros esfuerzos para generar lenguajes de programación. El primer lenguaje de programación es el Ensamblador, que simplificaba un poco la forma de expresar las instrucciones del microprocesador. En la segunda mitad de la década del 50 aparecen los primeros lenguajes con cierto nivel de abstracción, A-0 System (Arithmetic Language versión 0), FORTRAN (FORmula TRANslation), COBOL (Common Bussiness Oriented Language) y LISP (LISt Processing). Estos lenguajes requirieron varios años de trabajo para ser funcionales, pero el éxito de los tres últimos fue tan grande que todavía siguen vigentes.

La evolución del software y los paradigmas de programación utilizados siempre estuvieron acompañados por la aparición de diferentes lenguajes de programación y sus respectivos compiladores. Entre los lenguajes importantes que aparecieron a lo largo de la historia se pueden mencionar Algol-58, SmallTalk, PROLOG, C, C++, Java, ML, Haskell.

Dentro del proceso evolutivo de la computación, aparecen los lenguajes intrerpretados. Este tipo de lenguajes de programación no requieren un proceso de generación de código ejecutable ya que corren sobre un intérprete. Este tipo de lenguajes ofrece una dinámica de trabajo diferente, muy útil para ciertas situaciones. Entre los lenguajes interpretados se pueden destacar LISP, BASIC, LOGO, Shell Scripting, SQL, Python, HTML.

El conocimiento relacionado a las técnicas de compilación reviste un importante rol para los profesionales involucrados en el diseño y construcción de software debido a que le permiten principalmente conocer como obtener mayor provecho de los compiladores que utiliza y conocer sus limitaciones. Además, el estudio de la teoría utilizada para la generación de construcciones léxicas y gramaticales le permiten mejorar la comprensión de los lenguajes de programación que utiliza y lograr un mejor uso.

El estado de arte actual de todo lo referente a la compilación de código fuente ofrece una amplia gama de herramientas de software para la generación de compiladores desde diversos lenguajes de programación. El aprendizaje de algunas de estas herramientas abren un abanico de posibilidades para el desarrollo de todo lo referente a traducción de código, intrepretación de comandos, rastreo de archivos, etc.

METODOLOGIA DE ENSEÑANZA

Las etapas de construcción y elaboración de conocimientos son sustentadas mediante la exposición dialogada como estrategia didáctica y el empleo de proyección de diapositivas, filminas, pizarrón y proyector multimedia como materiales didácticos. Todos los materiales de estudio, incluyendo sistema de consultas, preguntas frecuentes, e-mail, evaluaciones, etc., se disponen en el sistema informático de aprendizaje del Departamento de Computación (Laboratorio de Enseñanza Virtual – LEV. http://lev.efn.uncor.edu)

La fase de ejercitación y aplicación de los contenidos de la asignatura, se fundamenta tanto en el desarrollo teórico como en el práctico del presente curso. Se realizan dos tipos diferenciados de actividades en coordinación con el desarrollo de la autonomía de aprendizaje, consistentes en la solución de problemas acotados y en la elaboración de un proyecto informático integrador realizado en equipo. En estas instancias el trabajo individual y grupal, permite la conformación de ideas y el establecimiento de relaciones entre el conocimiento adquirido y situaciones nuevas planteadas desde otras problemáticas de la misma disciplina.

El dictado se realizará en 16 clases de 3hs (reloj) consistentes en la presentación teórica de los temas por parte del docente, las que no podrán superar 1:30min en cada sesión.

La presentación de temas prácticos se realizará preferentemente, en el caso de disponer de equipos de computación, en el marco de tareas de laboratorio, previamente asignadas por el docente coincidentes con el tema teórico previo, asumiendo el docente el rol de tutor y mediante evaluaciones formativas en cada clase.

En el caso de no disponer del laboratorio se realizarán ejercitaciones y simulaciones de escritorio que permitan poner de manifiesto los objetivos de la asignatura.

El proceso de elaboración del proyecto integrador será seguido mediante entregas parciales pautadas en el LEV, así como la devolución de las evaluaciones.

Programación de actividades y bibliografía recomendada

Clase	Tema	Fecha
1	Unidad 1	
2	Unidad 2	
3	Unidad 2	
4	Unidad 2	
5	Unidad 3	
6	Unidad 3 – TP1	
7	Unidad 3	
8	Unidad 3	
9	Unidad 4	
10	Unidad 4	
11	Unidad 4 – TP2	
12	Unidad 5	
13	Unidad 5	
14	Unidad 5	
15	Unidad 6	
16	Unidad 6 – Consulta Proyectos	

EVALUACION

Trabajos Prácticos de Acreditacion

Se realizarán dos Trabajos Prácticos correspondientes a la Unidad 1 y 2 el primero y a la Unidad 3 el segundo. Consistirán en la generación de un software que responda a las consignas indicadas y a las metas de las Unidades abordadas. Cada Trabajo Práctico se calificará de 0 a 50 y se considerará aprobado con un mínimo de 30 puntos.

Evaluación Proyecto Final Integrador

Consistirá en la presentación escrita y posterior exposición y defensa del proyecto final integrador basado en el desarrollo de un Compilador utilizando las técnicas aprendidas. Los trabajos se presentarán en el examen final en un tiempo máximo asignado de 30min.
Se evaluarán los trabajos con notas de 0 a 10.

Condición de regularidad

Para alcanzar la condición de ALUMNO REGULAR se deberán cumplir los siguientes requisitos excluyentes:

- Asistir al 80% de las clases teóricas y de laboratorio.
- Aprobar ambos Trabajos Prácticos.

Régimen de promoción

Aprobación de la materia:

Para lograr la promoción se deberán alcanzar los siguientes objetivos excluyentes:

- Alcanzar la condición de ALUMNO REGULAR.
- Aprobar el Proyecto Final Integrador con nota cuatro (4) o superior.

Calificación final:

La calificación es el promedio ponderado de las diferentes evaluaciones y su valor numérico se establece como:

Nota Final = ((TP1 + TP2) /10)* 0.40 + Proyecto Final Integrador * 0.60

Este valor se redondeará al entero más próximo.

CONTENIDOS TEMATICOS

Unidad 1: INTRODUCCIÓN A LOS COMPILADORES
Procesadores de Lenguaje. Estructura de un compilador. Evolución de los lenguajes de programación y las técnicas de compilación. Definiciones sintácticas.

Unidad 2: ANÁLISIS LÉXICO
El rol del analizador léxico. Buffer de entrada. Especificación de tokens. Reconocimiento de tokens.

Unidad 3: ANÁLISIS SINTÁCTICO
El rol del analizador sintáctico. Gramáticas libres de contexto. Escritura de una gramática. Top-down y Bottom-up parsing.

Unidad 4: ANÁLISIS SEMÁNTICO
Definiciones dirigidas por la sintaxis. Órdenes de evaluación para definiciones dirigidas por la sintaxis. Aplicaciones de traducciones dirigidas por la sintaxis.

Unidad 5: GENERACIÓN DE CÓDIGO INTERMEDIO
Variantes de árboles sintácticos. Código de tres direcciones. Tipos y declaraciones. Verificación de tipos. Control de Flujo.

Unidad 6: GENERACIÓN Y OPTIMIZACIÓN DE CÓDIGO
Consideraciones para el diseño de un generador de código. Bloques básicos y grafos de flujo. Principios de optimización de código. Optimización de bloques básicos.

1. LISTADO DE ACTIVIDADES PRACTICAS Y/O DE LABORATORIO

Actividades Prácticas

1.- Actividades de Laboratorio

El alumno realizará actividades de programación en el Laboratorio de Computación que se corresponden con los ejercicios propuestos como actividades de práctica.

1. Generación de rastreadores
2. Manejo de Expresiones Regulares
3. Generación de Analizadores Sintácticos
4. Manejo de Gramáticas
5. Generación de Traductores entre lenguajes
6. Generación de Código Intermedio
7. Optimización Independiente de la arquitectura destino

2.- Actividades de Proyecto y Diseño

Tiene por objeto acreditar que el alumno ha adquirido las siguientes habilidades y técnicas, relacionadas preferentemente a la totalidad de los contenidos de la asignatura:

- Aplicar las técnicas objeto de la materia en la generación de un software capaz de interpretar un código fuente.
- Adquirir la habilidad para generar reglas léxicas y gramaticales para un lenguaje dado.
- Experimentar con diferentes criterios de diseño.
- Capacidad para el trabajo en equipo en la planificación y ejecución de un proyecto informático.

Características generales:

- El proyecto consistirá en el desarrollo de los algoritmos matemáticos y/o de información que den solución a un problema de ciencias o ingeniería.

- Se implementará la solución en el lenguaje definido y se probarán diferentes criterios de diseño y se presentarán todas las versiones de los archivos de código fuente.
- La aplicación resultante deberá poderse ejecutar en un sistema operativo sin errores sintácticos ni lógicos.
- Se documentará la presentación mediante una monografía sobre el tema, los criterios adoptados al respecto del diseño.
- Los grupos estarán constituidos por 2 alumnos como máximo.
- La presentación se realizará durante las clases de laboratorio correspondientes al último mes de clase.
- La calificación será de 0 a 10 y el peso relativo del 60% del total.

2. DISTRIBUCION DE LA CARGA HORARIA

ACTIVIDAD	**HORAS**
TEÓRICA	16
FORMACIÓN PRACTICA:	
o FORMACIÓN EXPERIMENTAL	23
o RESOLUCIÓN DE PROBLEMAS	23
o ACTIVIDADES DE PROYECTO Y DISEÑO	10
o PPS	
	72

DEDICADAS POR EL ALUMNO FUERA DE CLASE

ACTIVIDAD		**HORAS**
PREPARACION TEÓRICA		16
PREPARACION PRACTICA		
	o EXPERIMENTAL DE LABORATORIO	0
	o EXPERIMENTAL DE CAMPO	0
	o RESOLUCIÓN DE PROBLEMAS	20
	o PROYECTO Y DISEÑO	36
	TOTAL DE LA CARGA HORARIA	72

3. BIBLIOGRAFIA

Básica

- Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman. Compilers: Principles, Techniques, and Tools. Segunda Edición. Ed.Addison-Wesley. Estados Unidos. (2007) - Vol. 1014 - Isbn: 0-321-48681-1
- Kenneth J. Louden. Diseño e Implementación de Compiladores. Ed.Thomson. Mexico. (2006)

Recomendada

- Steven S. Muchnick. Advanced Compiler Design and Implementation. Ed.Morgan Kaufmann. San Francisco, Estados Unidos. (1997) - Vol. 856 – Isbn: 978-1-55860-320-2