Ministerio de Cultura y Educación
Universidad Nacional de San Luis
Facultad de Ciencias Físico Matemáticas y Naturales
Departamento: Informatica
Area: Area V: Automatas y Lenguajes

(Programa del año 2019)
(Programa en trámite de aprobación)
(Presentado el 24/08/2019 10:49:56)

I - Oferta Académica

Materia	Carrera	Plan	Año	Período
DISEÑO Y CONSTRUCCION DE COMPILADORES	LIC.CS.COMP.	32/12	2019	2° cuatrimestre

II - Equipo Docente

Docente	Función	Cargo	Dedicación
ARAGON, VICTORIA SOLEDAD	Prof. Responsable	P.Adj Exc	40 Hs
FERRETTI, EDGARDO	Responsable de Práctico	JTP Exc	40 Hs
GARCIARENA UCELAY, MARIA JOSE	Auxiliar de Práctico	E.Beca Fac	5 Hs

III - Características del Curso

Credito Horario Semanal				
Teórico/Práctico	**Teóricas**	**Prácticas de Aula**	**Práct. de lab/ camp/ Resid/ PIP, etc.**	**Total**
7 Hs	0 Hs	0 Hs	2 Hs	9 Hs

Tipificación	Periodo
B - Teoria con prácticas de aula y laboratorio	2º Cuatrimestre

Duración			
Desde	**Hasta**	**Cantidad de Semanas**	**Cantidad de Horas**
05/08/2019	15/11/2019	15	135

IV - Fundamentación

Los conceptos involucrados en el diseño y construcción de compiladores deben formar parte del conocimiento general de un Licenciado en Ciencias de la Computación, con el perfil requerido por el correspondiente plan de estudios.
Esta materia profundiza los conocimientos dados en la materia Análisis Comparativo de Lenguajes y aplica conceptos ya adquiridos en Autómatas y Lenguajes. Además las técnicas involucradas en la construcción de un compilador pueden aplicarse en el ámbito del desarrollo del software de base como en el desarrollo de software de aplicación. Los contenidos del curso se corresponden con las propuestas curriculares recomendadas por la ACM/IEEE Computer Society Joint Currículum Task force en 2012, para el área de compiladores.

V - Objetivos / Resultados de Aprendizaje

Al finalizar el curso se espera que el estudiante sea capaz de diseñar e implementar un compilador para un subconjunto del lenguaje de Programación.

VI - Contenidos

BOLILLA 1: INTRODUCCIÓN
Concepto y tipos de traductores. Compiladores de una o más pasadas. Estructura Clásica de un Compilador: Modelo de Análisis y Síntesis de un compilador. Sintaxis y Semántica.

BOLILLA 2: ANALIZADOR LEXICOGRÁFICO O SCANNER
Breve recapitulación de los conceptos fundamentales vistos previamente en la materia Autómatas y Lenguajes.

BOLILLA 3: ANÁLISIS SINTÁCTICO TOP DOWN o DESCENDENTE DETERMINÍSTICO
Definiciones de gramáticas LL(k) y LL(k) fuerte. Condición de LL(k) fuerte. Definición de los conjuntos FIRSTk, FOLLOWk. Teoremas Fundamentales. Parser descendente del tipo “table driven”. Construcción de tablas de parsing para gramáticas LL(k) con k = 1. Análisis sintáctico descendente por el método descendente recursivo . Generalidades. Algoritmos de implementación de un parser descendente recursivo. Errores en la etapa sintáctica. Esquema de recuperación de errores en entornos de descendentes recursivos, pánico y antipánico.

BOLILLA 4: TRADUCCIÓN DIRIGIDA POR LA SINTAXIS
Definiciones dirigidas por la sintaxis: atributos, atributos sintetizados y heredados, forma general de una definición dirigida por la sintaxis, grafo de dependencia de atributos. Definiciones S-atribuidas y L-atribuidas. Esquemas de traducción dirigidos por la sintaxis. Restricciones para el cálculo de los valores de los atributos. Implementación de esquemas de traducción L-atribuidos en un parser descendente recursivo.

BOLILLA 5: TABLA DE SÍMBOLOS
Finalidad. Entradas de la tabla de símbolos. Descripción de objetos en la tabla de símbolos. Tratamiento de nombres de longitud fija y variable. Revisión de posibles implementaciones: lista, hash, árbol, otras. Representación de la información sobre tipos. Manipulaciones básicas. Control de bloques y de amplitud.

BOLILLA 6: ANÁLISIS SEMÁNTICO
Introducción. Expresiones de tipo. Sistema de tipos. Salvaguarda de información del tipo asociado a los identificadores. Chequeo de tipo de expresiones y sentencias. Equivalencia de expresiones de tipo estructural. Errores semánticos. Esquema de recuperación de errores semánticos.

BOLILLA 7: AMBIENTES DE TIEMPO DE EJECUCIÓN
Revisión de las características básicas de los lenguajes de programación. Representación de tiempo de ejecución de objetos computacionales estáticos, semi-dinámicos y dinámicos. Organización de la memoria en tiempo de ejecución. Administración dinámica de la memoria como stack: activación de procedimientos, referenciación local, registros de activación, referenciación global, vector display. Administración dinámica de la memoria como heap: organización, administración del heap con elementos de tamaño fijo y variable, asignación inicial y re-uso.

BOLILLA 8: SISTEMA DE EJECUCIÓN BÁSICO Y EXTENDIDO
Representaciones intermedias de los programas: Notación polaca, de n-uplas, de árboles de sintaxis abstracta. Código de máquina abstracta. Arquitectura de la máquina virtual. Representación interna de los objetos en tiempo de ejecución: lógicos, caracteres, enteros, reales, arreglos. Análisis y traducción de expresiones aritméticas y lógicas, asignaciones, estructuras de control de repetición y selección, secuencia de llamado y retorno, pasaje de parámetros por dirección y por valor. Referenciación global y local. Variables sub-indicadas: acceso, pasaje de arreglos y elementos de arreglos como parámetros.

BOLILLA 9: GENERACIÓN DE CÓDIGO INTERMEDIO
Ubicación del generador de código dentro del proceso de compilación. Esquema de traducción dirigido por la sintaxis para la generación de código intermedio para expresiones, asignaciones, estructuras de control e invocación y retorno de procedimientos. Generación de código intermedio embebido en un parser descendente recursivo.

VII - Plan de Trabajos Prácticos

La metodología que se utilizará consta del dictado de clases teóricas con prácticas de aula presenciales y de laboratorio no presenciales, estas últimas guiadas por una programación didáctica desarrollada para abordar las diferentes temáticas.

El programa está compuesto por 9 bolillas y los temas se agruparon en bloques:

-Bloque 1: Bolilla 1

-Bloque 2: Bolillas 2 y 3
-Bloque 3: Bolilla 4
-Bloque 4: Bolillas 5 y 6
-Bloque 5: Bolillas 7 a 9

Las metas de comprensión que se planean alcanzar a través de la resolución de actividades en las prácticas de aula, por parte de los estudiantes, para cada bloque son:

-Bloque 1
- ¿Cuántas son y cómo interactúan las fases del compilador?¿Cuáles son los conceptos involucrados en la construcción de un compilador que ya poseo?

-Bloque 2
- ¿Cuándo una gramática es apta para ser analizada por un parser descendente o top-down? ¿Cómo transformar una gramática no apta en una apta?
- ¿Cómo se aplican las técnicas de parser descendente recursivo y tablas LL(k) fuertes? ¿Es posible recuperarse de un error sintáctico y continuar analizando la cadena de entrada? ¿Cómo?
- ¿Cómo se reflejan las estructuras y acciones de la tabla de parsing LL(K) fuertes en el parser descendente recursivo?

-Bloque 3
- Conocer los mecanismos de traducción dirigida por la sintaxis: definiciones dirigidas por la sintaxis (DDS) y esquemas de traducción dirigidas por la sintaxis (ETDS).
- Identificar cuándo un atributo es heredado y cuándo sintetizado.
- Reconocer las diferencias entre DDS y ETDS, y los requerimientos de implementación de cada uno de ellos.
- ¿Cómo se integran el esquema de traducción dirigido por la sintaxis y el parser descendente recursivo?

-Bloque 4
- En el contexto de un compilador ¿qué significa analizar semánticamente una cadena de entrada? y ¿cuál es la información que debe contener una expresión de tipo de acuerdo con el sistema de tipos?
- En lenguajes fuertemente tipeados, es decir, lenguajes en los cuales su compilador puede garantizar que los programas que acepte se ejecutarán sin errores de tipo, ¿cuándo y cómo se emplean estas expresiones de tipo? ¿Dónde se resguardan estas expresiones de tipo para luego poder ser recuperadas?
- ¿Qué información almacena una entrada en tabla de símbolos para un identificador en relación con su clase y para qué lo hace?
- ¿Cómo puede organizarse la tabla de símbolos en función del tipo de lenguaje a ser compilado?
- ¿Cómo se realiza el manejo de tabla de símbolos en un lenguaje fuertemente tipeado?

-Bloque 5
- ¿Cómo se relaciona la entrada en Tabla de Símbolos de un objeto con su representación en tiempo de ejecución?
- ¿Cómo diseño el descriptor para un objeto de tipo estructurado?
- ¿Cómo impactan en la arquitectura de la Máquina Abstracta para C las características del lenguaje fuente?
- ¿De dónde proviene la información para determinar cuáles son las instrucciones Máquina Abstracta para C que debo diseñar?

Prácticas de laboratorio no presenciales: consistirá en la programación del diseño de algunos módulos del compilador, cada uno de los cuales tendrá una entrega y aprobación parcial, en las fechas predeterminadas en el cronograma de la materia. Por lo que el objetivo general del laboratorio es diseñar e implementar un compilador para un subconjunto del lenguaje de Programación.

PRÁCTICO DE LABORATORIO 1: Parser Descendente Recursivo
Dado que la materia se desarrolla en un cuatrimestre y la programación del parser no presenta mayores dificultades, se optó por entregar el código del parser descendente, requiriéndoles a los estudiantes su interpretación y la inclusión de un esquema de recuperación de errores antipánico al mismo.

PRÁCTICO DE LABORATORIO 2: Tabla de Símbolos y Análisis Semántico
Dado que la materia se desarrolla en un cuatrimestre y los estudiantes traen como conceptos estudiados en materias anteriores conceptos de estructuras de datos y sus manipulaciones, se optó por entregar el código de la Administración de la Tabla de Símbolos, requiriéndoles a los estudiantes su interpretación y que completen algunas de las funciones de la misma. Los estudiantes deben diseñar un chequeador de tipos, usando un esquema de traducción dirigido por la sintaxis para el lenguaje del proyecto e incluirlo en el parser descendente recursivo.

PRÁCTICO DE LABORATORIO 3: Generación de Código Intermedio
Los estudiantes usando un esquema de traducción dirigido por la sintaxis y la máquina virtual definida para el lenguaje del proyecto, deben incorporar al parser la generación de código intermedio.

VIII - Regimen de Aprobación

F.1. Régimen para estudiantes regulares

Para regularizar la materia los estudiantes deberán aprobar:

1. Los prácticos de laboratorio (fuente y ejecutable) y/o su correspondiente recuperación en las fechas estipuladas a tal efecto. La no aprobación de un trabajo de máquina y/o su recuperación implicará la pérdida automática de la regularidad de la materia.
2. Un examen parcial práctico o alguna de sus dos recuperaciones, con un porcentaje de ejercicios correctos del 70%.
3. La presentación oral y escrita de un informe que releve las características y decisiones de diseño tomadas para la implementación del compilador junto con el rol que desempeñó cada integrante del grupo y la dinámica grupal con la cual trabajaron, entre otros.

F.2. Régimen de aprobación para estudiantes regulares

Los estudiantes que han regularizado la materia para aprobarla deberán rendir un examen final que podrá ser escrito u oral.

F.3. Régimen de estudiantes libres

En estos casos, el estudiante tendrá una evaluación dividida en partes. En una se pedirá un Trabajo Especial, el cual es un compilador desarrollado bajo las pautas que se dan en el curso de la asignatura. En otra parte se tomará un examen escrito de carácter práctico. Finalmente, una parte oral y/o escrita de teoría. Para su aprobación, se requiere la aprobación de las tres partes.

F.4. En la asignatura no se admite el régimen por promoción.

IX - Bibliografía Básica

[1] Aho, A. y Ullman, J.D. : "The Theory of parsing. Translation and Compiling", Vol. I y II, Prentice Hall.
[2] Aho, A.V y Ullman, J.D : "Principles of Compiler Design", Addison Wesley.
[3] Aho A.V, Sethi R.y Ullman J.D : "Compilers. Principles, Techniques and Tools", First Edition,Addison Wesley, 1986.
[4] Aho, A.V.,Lam M.S, Sethi R. and Ullman J.D.: "Compilers. Principles, Techniques, & Tools", Second Edition, 2007, Pearson/Addison Wesley, 2007.

X - Bibliografia Complementaria

[1] Gries, Davis: "Compilers Construction", John Wiley, 1975.
[2] Backhouse, R. C : "Syntax of Programming Language", Prentice Hall.
[3] Davie, A. et al.: "Recursive Descendent Compiling", John Wiley.
[4] Wilhelm R., Maurer D. : Compiler Design, Addison Wesley, 1995
[5] Bauer et al.: "An advanced course on Compilers", Springer Verlag.
[6] Waite W.M, Carter L. R. : " An Introduction to Compiler Construction ", Harper Collins College, Publishers, 1993.
[7] Bornat Richard: "Understanding and Writing Compilers", The Macmillan Press LTD, 1982.
[8] Tremblay y Sorenson : "the Theory and Paractice of Compiler Writing", Mc Graw Hill, Computer Science Series, 1985

[9] Appel A. W.: "Modern Compiler Implementation in Java", Second Edition, Cambridge University Press, 2006.

XI - Resumen de Objetivos

Desarrollar en el estudiante la capacidad de diseñar e implementar un compilador para un subconjunto de Lenguaje de Programación.

XII - Resumen del Programa

Descripción de los módulos de un compilador. Análisis Lexicográfico. Análisis Sintáctico. Recuperación de errores. Tabla de Símbolos. Análisis Semántico. Traducción dirigida por la sintaxis. Chequeo de tipos. Generación de código.

XIII - Imprevistos

XIV - Otros

ELEVACIÓN y APROBACIÓN DE ESTE PROGRAMA	
	Profesor Responsable
Firma:	
Aclaración:	
Fecha:	