Ministerio de Cultura y Educación **(Programa del año 2014)**
Universidad Nacional de San Luis
Facultad de Ciencias Físico Matemáticas y Naturales
Departamento: Informatica
Area: Area V: Automatas y Lenguajes

I - Oferta Académica

Materia	Carrera	Plan	Año	Período
DISEÑO Y CONSTRUCCION DE COMPILADORES	LIC.CS.COMP.	32/12	2014	2° cuatrimestre
DISEÑO Y CONSTRUCCION DE COMPILADORES	LIC.CS.COMP.	006/05	2014	2° cuatrimestre

II - Equipo Docente

Docente	Función	Cargo	Dedicación
ESQUIVEL, SUSANA CECILIA	Prof. Responsable	P.Tit. Exc	40 Hs
ARAGON, VICTORIA SOLEDAD	Prof. Colaborador	P.Adj Exc	40 Hs
FERRETTI, EDGARDO	Responsable de Práctico	JTP Exc	40 Hs

III - Características del Curso

Credito Horario Semanal				
Teórico/Práctico	**Teóricas**	**Prácticas de Aula**	**Práct. de lab/ camp/ Resid/ PIP, etc.**	**Total**
10 Hs	3 Hs	2 Hs	5 Hs	10 Hs

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

Duración			
Desde	**Hasta**	**Cantidad de Semanas**	**Cantidad de Horas**
19/08/2014	28/11/2014	15	150

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 alumno sea capaz de:

a) Diseñar e implementar un compilador para un subconjunto del lenguaje de Programación.
b) Diseñar las distintas etapas involucradas en la construcción de un compilador para un lenguaje que defina alguna aplicación.

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: 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 4: PARSING TOP DOWN DETERMINÍSTICO

Definiciones de gramáticas LL(k) y LL(k) fuerte. Condición de LL(k) fuerte. Conjuntos FIRSTk, FOLLOWk, algoritmos de construcción de dichos conjuntos. Teoremas Fundamentales. Análisis sintáctico descendente por medio de tablas, para k=1. Análisis sintáctico descendente por el método recursivo descendente. Generalidades. Algoritmos de implementación de un parser descendente recursivo. Parser descendente del tipo "table driven". Construcción de tablas de parsing para gramáticas LL(k) con k > 1.

BOLILLA 5: DETECCIÓN Y RECUPERACIÓN DE ERRORES

Errores en las etapas lexicográfica, sintáctica, semántica y de ejecución. Esquema de recuperación de errores en entornos de parsing descendente (tablas LL(k) fuertes y parser descendente recursivo).

BOLILLA 6: 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 7: REPRESENTACIONES INTERMEDIAS DE LOS PROGRAMAS FUENTES

Notación polaca, de n-uplas, de árboles de sintaxis abstracta. Código de máquina abstracta. Portabilidad.

BOLILLA 8: SISTEMA DE EJECUCIÓN BASICO Y EXTENDIDO

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 a nivel de sentencia, entrada-salida, estructuras de control a nivel de procedimiento, 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: 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: traductores bottom up. Definiciones L-atribuidas. Esquemas de traducción dirigidos por la sintaxis. Restricciones para el cálculo de los valores de los atributos. Implementación de definiciones L-atribuidas en un entorno de parser top down. Eliminación de recursividad a izquierda de los esquemas de traducción. Parsing descendente recursivo que implementa definiciones L-atribuidas.

BOLILLA 10: CONTROL DE TIPOS

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.

BOLILLA 11: GENERACION 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 a nivel de sentencias y unidades. Generación de código intermedio embebido en un parser descendente recursivo.

BOLILLA 12: GRAMATICAS LR(k) Y PARSER ASCENDENTE

Gramáticas LR(k). Parser ascendente. Diferentes tipos de parser ascendentes: SLR, Canónico y Lookahead. Método de construcción de parser SRL.

VII - Plan de Trabajos Prácticos

I. PLAN DE TRABAJOS PRÁCTICOS DE AULA (Corresponden a ejercicios de lápiz y papel con la finalidad de comprensión, fijación y aplicación de los conceptos asociados)

PRÁCTICO 1: Tabla de símbolos.

Descripción:

En este práctico se le entrega al alumno el código fuente del módulo que implementa la tabla de símbolos más una guía de ejercicios que orienta a los alumnos en la comprensión del tipo de organización de la TS seleccionada y su manipulación. Los alumnos ya tienen conocimientos y experiencias de implementación de estructuras de datos y de archivos y de las operaciones que sobre ellos se efectúan. Por lo anterior, no se consideró necesario que desarrollaran este módulo, sino que pudieran interpretarlo.

Objetivos Específicos: Al finalizar el práctico se espera que el alumno sea capaz de:

a) Identificar posibles representaciones de almacenamiento en la tabla de símbolos para diferentes objetos de datos que puedan aparecer en un programa.
b) Identificar en el código fuente las principales funciones de administración de la tabla de símbolos comprendiendo las tareas que realizan.
c) Explicar la organización de la tabla de símbolos implementada.

PRÁCTICO 2: Gramáticas LL(k).

Descripción:

Esta guía está orientada a que los alumnos fijen y comprendan los conceptos asociados con las gramáticas LL(k) y LL(k) fuertes como asimismo que puedan identificar cuándo una gramática es LL(k) y/o LL(k) fuerte aplicando para ello o bien las respectivas definiciones o los teoremas que correspondan.

Objetivos Específicos: Al finalizar práctico se espera que el alumno sea capaz de:

a) Identificar las diferencias desde el punto de vista del análisis sintáctico entre una gramática LL(k) y LL(k) fuerte para k > 1.
b) Identificar cuándo una gramática no es LL(k) para ningún k.
c) Determinar, usando el método que considere más expeditivo, cuándo una gramática es o no es LL(k)/LL(k) fuerte para un k conocido.
d) Demostrar propiedades de las gramáticas LL(k).
e) Dada una gramática CF, si no es LLk)/LL(k)fuerte realizar las transformaciones necesarias para que lo sea.
f) Construir los conjuntos Firstk y Followk para distintas gramáticas.
g) Construir las tablas de parsing descendente para gramáticas LL(k) fuerte con distintos valores de k.

PRÁCTICO 3: Parsing Descendente Recursivo.

Descripción:

Esta guía está orientada a que los alumnos fijen, comprendan y apliquen los conceptos relacionados con el parser descendente recursivo.

Objetivos Específicos: Al finalizar el práctico se espera que el alumno sea capaz de:

a) Transformar gramáticas expresadas en forma BNF a forma BNFE.
b) Expresar las ventajas/desventajas (si existen) de ambas formas de expresar las gramáticas.
c) Implementar los reconocedores del parser descendente recursivo para diferentes gramáticas expresadas en BNF y/o BNFE.

PRÁCTICO 4: Recuperación de errores.

Descripción:

Esta guía está orientada a que los alumnos fijen, comprendan y apliquen los conceptos relacionados con el esquema de recuperación de errores:

a) Pánico y su implementación en las tablas de parser LL(k) fuertes.

b) Antipánico y su implementación en un parser descendente recursivo.

Objetivos Específicos: Al finalizar el práctico se espera que el alumno sea capaz de:

a) Comprender la función de los distintos componentes de los métodos de recuperación de errores estudiados.
b) Reconocer cuáles son las características de los lenguajes que favorecen el uso de cada método, sus ventajas y desventajas.
c) Consolidar los conceptos previos a través de su aplicación en ejercicios sobre lenguajes académicos.

PRÁCTICO 5: Traducción dirigida por la sintaxis en entorno de parser top descendente.

Descripción:

Esta guía está orientada a que los alumnos fijen, comprendan y apliquen los conceptos relacionados con la traducción de lenguajes guiadas por las gramáticas libres de contexto.

Objetivos Específicos: Al finalizar el práctico se espera que el alumno sea capaz de:

a) Conocer los mecanismos de traducción dirigida por la sintaxis: definiciones dirigidas por la sintaxis y esquemas de traducción dirigidas por la sintaxis.
b) Comprender para qué entornos de parser son útiles las definiciones/esquemas S-atribuidos y L-atribuidos.
c) Reconocer las diferencias entre ambos métodos y los requerimientos de implementación de cada uno de ellos.
d) Implementar los distintos métodos sobre lenguajes académicos y alguna aplicación.

PRÁCTICO 6: Sistema de Ejecución Básico.

Descripción:

Dada la definición del sistema de ejecución requerido para la implementación del compilador a desarrollar, esta guía tiene por finalidad que el estudiante pueda pensar en representaciones de tiempo de ejecución de distintos tipos de objetos computacionales, definiendo si fuere necesario nuevas instrucciones de la máquina virtual para su manipulación en tiempo de ejecución
Objetivos Específicos: Al finalizar el práctico se espera que el alumno sea capaz de:

a) Extender, si fuera necesario, el soporte de tiempo de ejecución definido en clase para soportar distintos objetos computacionales y/o estructuras sintácticas, definiendo formas de representación en tiempo de ejecución y/o instrucciones de la máquina virtual según corresponda.

PRÁCTICO 7: Chequeo de tipos.

Descripción:

Esta guía está orientada a que los alumnos implementen el análisis semántico referido al control de tipos usando esquemas de traducción sobre lenguajes académicos con la finalidad de comprensión y fijación de conceptos.

Objetivos Específicos: Al finalizar el práctico se espera que el alumno sea capaz de:

a) Desarrollar esquemas de traducción para controlar diferentes sistemas de tipos.
b) Incorporar dichos esquemas en parser descendentes recursivos.

PRÁCTICO 8: Generación de Código.

Descripción:

Esta guía está orientada a que los alumnos implementen la generación de código intermedio para distintas construcciones sintácticas usando esquemas de traducción, con la finalidad de comprensión y fijación de conceptos.

Objetivos Específicos: Al finalizar el práctico se espera que el alumno sea capaz de:

a) Desarrollar esquemas de traducción para generar código intermedio para diferentes construcciones sintácticas.
b) Incorporar dichos esquemas en parsers descendentes recursivos.

PRÁCTICO 9: Gramáticas LR(k).
Descripción:
Esta guía está orientada a que los alumnos fijen y comprendan los conceptos asociados con las gramáticas LR(k) y la construcción de tablas SLR.
Objetivos Específicos: Al finalizar práctico se espera que el alumno sea capaz de:
a) Construir la Colección Canónica de Ítems LR(0) para una gramática dada.
b) Construir el autómata que reconoce los prefijos viables para una gramáticas dada.
c) Construir la tabla de parsing SLR para una gramática dada.
d) Realizar el análisis sintáctico de distintas cadenas de entrada mostrando en cada paso el contenido del stack (estado y símbolo), la cadena de entrada y la próxima acción a realizar.
e) Resolver conflictos dados por el/los conjunto(s) de ítems.

PRÁCTICO 10: Desarrollo de una Aplicación de Conceptos de Compiladores a otros Ámbitos

Descripción: Si bien este práctico de aula se describe al final, se desarrolla en paralelo con los prácticos anteriores, como actividad extra aula. La idea es que los estudiantes puedan transferir los conceptos relacionados con la construcción de compiladores a otras áreas aplicativas. La guía describe una aplicación, por ejemplo generación de tickets de avión, de libranzas de sueldos, etc. y los alumnos deben tomar como entrada los datos de un archivo (por ejemplo) y generar usando las técnicas estudiadas la traducción solicitada. Dicho práctico debe obligatoriamente ser entregado.

Objetivo General:

a) Diseñar las distintas etapas involucradas en la construcción de un compilador para un lenguaje que defina alguna aplicación.

II. PLAN DE TRABAJOS PRÁCTICOS DE LABORATORIO (formación práctica: formación experimenta y actividad de diseño y proyecto, 75 horas).

Descripción General: Los prácticos de laboratorio comprenden dos tipos de formación práctica:

Actividades de diseño y proyecto: El alumno debe trabajar sobre el proyecto de diseño e implementación de un compilador para un subconjunto de un lenguaje de programación determinado, para ello trabajarán con una metodología modular, esto es diseñar los distintos módulos del compilador de manera incremental, esto es primero el desarrollo del parser, una vez verificada su corrección lo enriquecerán incorporando un esquema de recuperación de errores, una vez verificada su corrección, le incorporarán el análisis semántico y así siguiendo.

Actividades de formación experimental: Consistirá en la programación del diseño de cada módulo del compilador, cada uno de los cuales tendrá una entrega y aprobación parcial, en las fechas predeterminadas en el cronograma de la materia.

Objetivo General del Laboratorio:

a) Diseñar e implementar un compilador para un subconjunto del lenguaje de Programación.

PRÁCTICO DE LABORATORIO 1: Tabla de Símbolos
Dado que la materia se desarrolla en un cuatrimestre y los alumnos 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 alumnos su interpretación y que completen algunas de las funciones de la misma.

PRÁCTICO DE LABORATORIO 2: 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 alumnos su interpretación y que completen algunas de las funciones reconocedores del mismo.

PRÁCTICO DE LABORATORIO 3: Esquema de Recuperación de Errores
Los alumnos deben diseñar un esquema de recuperación de errores para el lenguaje del proyecto e incluirlo en el parser descendente recursivo.

PRÁCTICO DE LABORATORIO 4: Análisis Semántico
Los alumnos 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 5: Generación de Código Intermedio
Los alumnos 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 par alumnos regulares

Para regularizar la materia los alumnos deberán aprobar:

1. Los prácticos de programación propuestos (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. El trabajo práctico de aula n° 10.

3. Un examen parcial práctico o su respectiva recuperación, con un porcentaje de ejercicios correctos del 70%.
4. Aquellos alumnos que han presentado, en tiempo y forma, el comprobante de trabajo, contarán con una recuperación adicional.

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

F.3. Régimen de alumnos libres

En estos casos, el alumno 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.

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 alumno 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. Tabla de Símbolos. Análisis Sintáctico. Recuperación de errores. Análisis Semántico. Traducción dirigida por la sintaxis. Chequeo de tipos. Generación de código.

XIII - Imprevistos

XIV - Otros