Ministerio de Cultura y Educación **(Programa del año 2022)**
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.	18/11	2022	2° cuatrimestre
DISEÑO Y CONSTRUCCION DE COMPILADORES	LIC.CS.COMP.	32/12	2022	2° cuatrimestre

II - Equipo Docente

Docente	Función	Cargo	Dedicación
ARAGON, VICTORIA SOLEDAD	Prof. Responsable	P.Adj Exc	40 Hs
FERRETTI, EDGARDO	Responsable de Práctico	P.Adj Exc	40 Hs

III - Características del Curso

Credito Horario Semanal				
Teórico/Práctico	Teóricas	Prácticas de Aula	Práct. de lab/ camp/ Resid/ PIP, etc.	Total
Hs	3 Hs	3 Hs	3 Hs	9 Hs

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

Duración			
Desde	Hasta	Cantidad de Semanas	Cantidad de Horas
08/08/2022	18/11/2022	15	135

IV - Fundamentación

Los conceptos involucrados en el diseño y construcción de compiladores forman parte del conocimiento general de un Licenciado en Ciencias de la Computación, con el perfil requerido por los correspondientes planes de estudios. Es importante resaltar la pertinencia de la asignatura y todas las motivaciones que justifican su inclusión como una materia de estudio obligatorio en los planes de estudio de esta licenciatura. Durante todo el cursado se recuperarán los conocimientos previos adquiridos en el contexto de diseño y construcción de un compilador, ya que esta materia profundiza los conocimientos dados en la materia Análisis Comparativo de Lenguajes y aplica conceptos ya apropiados en Autómatas y Lenguajes, Organización de Archivos y Bases de Datos I y Arquitectura I. Los principios, técnicas y herramientas que se aprenderán en el curso son aplicables a múltiples problemas. Seguramente ningún estudiante tendrá que utilizar en su vida profesional los conocimientos y las habilidades adquiridos en esta materia para diseñar un compilador propiamente dicho. No obstante, sí se enfrentará a una variedad de contextos donde estos conocimientos y habilidades son aplicables: traductores de un lenguaje a otro (por ejemplo, de Latex a HTML, de un formato de archivo binario a gráficos, de preguntas en lenguaje natural en un dominio específico a SQL, procesamiento de XML, etc.). Estas aplicaciones se resaltarán durante el cursado para propiciar la transferencia lejana en los estudiantes.

En la asignatura se aborda el diseño y la construcción del front-end de un compilador, haciendo hincapié en las decisiones de diseño que se deben tomar en cada fase que lo componen, y el impacto que tienen las decisiones tomadas en una fase con

respecto al resto. Si bien el propósito de la asignatura es que el estudiante sea capaz de diseñar e implementar parte de un compilador para un subconjunto del lenguaje de programación C++, estos conocimientos son un medio a través del cual se persiguen alcanzar una serie de objetivos necesarios para la formación profesional de un licenciado en ciencias de la computación: se trabaja en un proyecto completo, el cual hay que documentar, depurar, probar y revisar aplicando el concepto de ciclo de vida del software de forma natural. Además, la construcción de un compilador impone desafíos algorítmicos, como ser, la utilización de distintos tipos de estructuras de datos (árboles, grafos, pilas, rebalses, etc.) y algoritmos (tablas LL(k) fuertes, recorridos de árboles, ordenación, etc.). Es necesario, dada la magnitud del proyecto, el uso de herramientas para el desarrollo y presentación del mismo (Flex, depuradores, lex, sistemas de control de versiones, audiovisuales, etc.). Por último, estudiar compiladores permite aprender cómo funcionan y se construyen los lenguajes de programación mejorando así las habilidades de programación de los estudiantes.

Los contenidos del curso se corresponden con las propuestas curriculares recomendadas por la ACM/IEEE Computer Society Joint Currículum Task force en 2013, para el área de compiladores.

V - Objetivos / Resultados de Aprendizaje

Se consideran a continuación un conjunto de objetivos tanto de carácter general o transversal como de carácter específico, que se pretenderá que alcance el estudiantado a lo largo del curso.

Objetivos generales

La asignatura pretende:
- Fomentar el análisis crítico y evaluación de diferentes alternativas.
- Fomentar el espíritu de mejora continua y autoconocimiento.
- Fomentar el trabajo en grupo.
- Fomentar la expresión oral y escrita.

Objetivos específicos

La asignatura pretende:
- Identificar los elementos constitutivos de un compilador: comprender y dominar su funcionamiento.
- Aplicar conocimientos previos de gramáticas y autómatas para la especificación y construcción de compiladores.
- Diferenciar y dominar las diferentes técnicas para llevar a cabo las fases de análisis de lenguajes.
- Generar código intermedio para una máquina virtual para un lenguaje fuente.

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.

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. Detección y recuperación de errores sintácticos: 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. Detección y recuperación de errores: 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.

Las bolillas se agruparon en bloques de acuerdo con la fase del compilador en la que los temas que las componen presentan una mayor articulación:

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 en 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?
- ¿Cuáles son las decisiones involucradas en el diseño de un compilador?

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?
- ¿Cuáles son las decisiones involucradas en el diseño de los analizadores lexicográfico y sintáctico?

Bloque 3
- ¿Qué caracteriza a una definición dirigida por la sintaxis? y ¿qué a un esquema de traducción dirigido por la sintaxis?
- Identificar cuándo un atributo es heredado y cuándo sintetizado. ¿Para qué sirve cada uno de ellos?
- ¿Cómo se integran el esquema de traducción dirigido por la sintaxis y el parser descendente recursivo?
- ¿Cuáles son las decisiones involucradas en el diseño de una traducción dirigida por la sintaxis?

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?
- ¿Cuáles son las decisiones involucradas en el diseño del chequeador semántico y la tabla de símbolos?

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? ¿Cuáles son las consideraciones que deben tomarse?
- ¿Cómo impactan en la arquitectura de la Máquina Abstracta las características del lenguaje fuente?
- ¿De dónde proviene la información para determinar cuáles son las instrucciones de la Máquina Abstracta que debo diseñar?
- ¿Cuáles son las decisiones involucradas en el diseño de la generación de código intermedio?

VII - Plan de Trabajos Prácticos

Metodología de trabajo

Los 5 bloques en los que están agrupados los contenidos se dictarán durante las 15 semanas del cuatrimestre. Para trabajar cada bloque, previo al encuentro en aula, las clases teóricas grabadas, las diapositivas de clase y los apuntes teóricos correspondientes deberán ser abordados de forma individual, en el aula se trabajarán los contenidos a través de realización de mapas conceptuales, consultas, debates, y actividades. El material estará disponible en un aula virtual al inicio del cuatrimestre. Las actividades para cada bloque apuntan a que los estudiantes sean capaces de:

Bloque 1
- Identificar las partes constitutivas de un compilador y sus funcionalidades.
- Recuperar los conceptos previos ya adquiridos para la temática compiladores a lo largo de la carrera.
- Identificar las decisiones de diseño iniciales para la construcción de un compilador.

Bloque 2

- Identificar cuándo una gramática no es LL(k) para ningún k.
- Determinar cuándo una gramática es o no es LL(1) fuerte.
- Dada una gramática libre de contexto, si no es LL(1) fuerte realizar las transformaciones necesarias para que lo sea.
- Construir los conjuntos Firstk y Followk para distintas gramáticas.
- Construir las tablas de parsing descendente para gramáticas LL(1) fuerte.
- Transformar gramáticas expresadas en forma BNF a forma BNFE. Expresar las ventajas/desventajas (si existen) de ambas formas de expresar las gramáticas.
- Implementar los reconocedores del parser descendente recursivo para diferentes gramáticas expresadas en BNF y/o BNFE.
- Implementar en un parser descendente recursivo un esquema de recuperación de errores pánico.
- Implementar en un parser descendente recursivo un esquema de recuperación de errores antipánico.
- Comprender la función de los distintos componentes de los métodos de recuperación de errores estudiados.
- Reconocer cuáles son las características de los lenguajes que favorecen el uso de cada método de recuperación, sus ventajas y desventajas.
- Consolidar los conceptos previos a través de su aplicación en ejercicios sobre lenguajes académicos.
- Identificar decisiones involucradas en el diseño de un analizador lexicográfico y uno sintáctico.

Bloque 3
- Conocer los mecanismos de traducción dirigida por la sintaxis: definiciones dirigidas por la sintaxis y esquemas de traducción dirigidas por la sintaxis.
- Comprender para qué entornos de parser son útiles las definiciones/esquemas S-atribuidos y L-atribuidos.
- Reconocer las diferencias entre ambos métodos y los requerimientos de implementación de cada uno de ellos.
- Implementar esquemas de traducción dirigidas por la sintaxis sobre lenguajes académicos y alguna aplicación.
- Identificar decisiones involucradas en el diseño de una traducción dirigida por la sintaxis.

Bloque 4
- Identificar posibles representaciones de almacenamiento en la tabla de símbolos para diferentes objetos de datos que puedan aparecer en un programa.
- Diseñar entradas en la tabla de símbolos para los objetos computacionales de acuerdo con su clase.
- Identificar en el código fuente (provisto) las principales funciones de administración de la tabla de símbolos comprendiendo las tareas que realizan.
- Explicar la organización de la tabla de símbolos implementada dada.
- Desarrollar esquemas de traducción para controlar diferentes sistemas de tipos.
- Incorporar dichos esquemas en parser descendentes recursivos.
- Identificar las decisiones involucradas en el diseño del chequeador semántico y la tabla de símbolos.

Bloque 5
- 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.
- Desarrollar esquemas de traducción para generar código intermedio para diferentes construcciones sintácticas.
- Incorporar dichos esquemas en parsers descendentes recursivos.
- Identificar las decisiones involucradas en el diseño de la generación de código intermedio.

PLAN DE TRABAJOS PRÁCTICOS DE LABORATORIO

Los prácticos de laboratorio comprenden tres tipos de formación práctica:

- Actividades de diseño y proyecto: El estudiante debe trabajar sobre el proyecto de diseño e implementación de un compilador para un subconjunto de un lenguaje de programación determinado, para ello se trabajará con una metodología modular, esto es diseñar los distintos módulos del compilador de manera incremental, primero el desarrollo del parser con el esquema de recuperación de errores, una vez verificada su corrección, le incorporarán el análisis semántico y luego la generación de código intermedio.
- Actividades de formación experimental: El estudiante debe programar el 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.
- Expresión escrita y oral: El estudiante debe presentar en forma oral y escrita 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 (de ser grupal) y la dinámica grupal con la cual trabajaron, entre otros.

El objetivo general del Laboratorio es diseñar e implementar un compilador para un subconjunto del lenguaje de Programación C++. Los módulos establecidos son:

- PRÁCTICO DE LABORATORIO 1: Parser Descendente Recursivo

Dado que la materia se desarrolla en un cuatrimestre y la programación de los analizadores lexicográfico y sintáctico (parser descendente recursivo) no presentan mayores dificultades, se optó por entregar los códigos de ambos, requiriéndoles a los estudiantes su interpretación y la inclusión de un esquema de recuperación de errores antipánico al parser.

- 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.

Toda consulta teórica-práctica podrá realizarse en los horarios de las clases presenciales (6 horas por semana).

Las vías de comunicación con los estudiantes son las siguientes:
- Correos electrónicos de los docentes:
vsaragon@email.unsl.edu.ar
iferretti@email.unsl.edu.ar
- Oficina: 04 - 1° piso - 2° Bloque - Teléfono: +54 (266) 4520300 - Int 2104
- Sitio web: https://sites.google.com/site/ddcompilersunsl/
- Aula Virtual: (https://classroom.google.com/c/NTI2MjI4OTUyNTcz?cjc=4wp4hiq) donde se pueden descargar las teorías, trabajos prácticos, avisos importantes, apuntes, cronograma, entre otros.

VIII - Regimen de Aprobación

Régimen para estudiantes promocionales

Para promocionar la materia los estudiantes deberán:

1. Aprobar 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 práctico de máquina y/o su recuperación implicará la pérdida automática de la regularidad de la materia. Estos trabajos se Aprueban o No aprueban.

2. Aprobar un examen práctico o alguna de sus dos recuperaciones, con un porcentaje de ejercicios correctos de al menos el 80%, con nota no menor a 7.

3. Aprobar la presentación oral y escrita del informe referente a los prácticos de laboratorio, con nota no menor a 7.

4. Aprobar al menos el 80% de las actividades que se propongan, con nota no menor a 7.

5. Aprobar un examen integrador (oral o escrito) con nota no menor a 7.
6. Tener un mínimo de 80% de asistencia a las clases prácticas y teóricas.

La nota final será el promedio de las notas obtenidas en las entregas, examen práctico, informe e examen integrador no pudiendo ser menor a 7.

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. Estos trabajos se Aprueban o No aprueban.

2. Un examen práctico o alguna de sus dos recuperaciones, con un porcentaje de ejercicios correctos del 70% con nota no menor a 6.

3. La presentación oral y escrita del informe referente a los prácticos de laboratorio, con nota no menor a 6.

4. La participación en el 70% de las actividades que se propongan.

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.

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.

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 Practice 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