

# PROGRAMA DE COMPILADORES

## 1. Introducción

- 1.1. Tipos de traductores
- 1.2. Autómatas
- 1.3. Gramáticas formales
- 1.4. Fases de un compilador

## 2. Análisis Léxico

- 2.1. Definir un reconocedor de cadenas no trivial
- 2.2. Programar sistemáticamente el reconocedor en lo referente a la obtención del autómata, almacenarlo eficientemente y manejar adecuadamente el archivo fuente
- 2.3. Utilería LEX
- 2.4. Notaciones y nomenclatura en LEX
- 2.5. Programación LEX del mismo reconocedor
- 2.6. Comparación de dos tipos de técnicas

## 3. Análisis Sintáctico

- 3.1. Parsers LR
- 3.2. Gramáticas LR
- 3.3. Métodos SLR, LR canónico, LARL
- 3.4. Construcción de tablas de análisis SLR, LR canónico y LARL
- 3.5. Manejo de gramáticas ambiguas
- 3.6. Recuperación de errores
- 3.7. Empleo de YACC
- 3.8. Notación de YACC
- 3.9. Manejo de errores en YACC

## 4. Problemas de Implementación

- 4.1. Estrategias de acceso a memoria. Técnicas de acceso dinámico
- 4.2. Acceso a nombres no locales, bloques, alcance
- 4.3. Paso de parámetros
- 4.4. Tablas de datos, hashing, representación del alcance. Diferentes tipos de tablas

## 5. Generación de Código Intermedio

- 5.1. Representaciones de código intermedio
- 5.2. Generación de código intermedio y análisis semántico en compilación top-down
- 5.3. Generación de código intermedio y análisis semántico en compilación bottom-up

5.4. Instrucciones de asignación. Expresiones booleanas. Instrucciones de control.  
Instrucciones de entrada y salida

## **6. Generación de Código**

- 6.1. Aspectos generales de la generación de código. Manejo de memoria
- 6.2. Selección de instrucciones. Acceso de registros
- 6.3. Aspectos relacionados con la máquina anfitriona
- 6.4. Bloques básicos y gráficas de flujo
- 6.5. Gráficas dirigidas acíclicas
- 6.6. Algoritmos para generación de código
- 6.7. Generación de código en YACC

## **7. Optimización de Código**

- 7.1. Optimización de código
- 7.2. Optimización de código intermedio
- 7.3. Fuentes principales de optimización. Subexpresiones comunes
- 7.4. Propagación de copias. Eliminación de código muerto. Ciclos
- 7.5. Optimización de bloques básicos
- 7.6. Ciclos de gráfica de flujo
- 7.7. Análisis de flujo de datos

## **8. Proyectos**

- 8.1. Definición del proyecto
- 8.2. Análisis y diseño del compilador

## BIBLIOGRAFÍA

**1. Compilers: Principles, Techniques and Tools.**

A.V. Aho, M.S. Lam, R. Sethi, J.D. Ullman.

Addison-Wesley; 2<sup>nd</sup> edition (August 31, 2006)

Se encuentra en las bibliotecas: FISMATIN-Xalapa y ECONOMIA-Xalapa.  
(QA76.76.C65 A36)

**2. Compilers: Construction for Digital Computers**

D. Gries.

Wiley International Edition.

Se encuentra en las bibliotecas: USBI-Xalapa, FISMATIN-Xalapa y  
MIA-Xalapa (QA76.5 G74).

**3. The Theory and Practice of Compiler Writing.**

L.P. Tremblay, P.G. Sorenson.

McGraw Hill (1985).

Se encuentra en la biblioteca: ECONOMIA-Xalapa. (QA76.6 T73)

**4. Machines, Languages and Computations.**

P.J. Denning, J.B. Dennis, J.E. Qualitz.

Prentice Hall (July 1978).

Se encuentra en la biblioteca: INVESBIO-Xalapa. (QA267.D46)

**5. Theory of Computation: Formal Languages, Automata and Complexity.**

J. Glenn

Benjamin/Cummings Series in Computer Science (January 1989).

Se encuentra en las bibliotecas: USBI-Xalapa y ECONOMIA-Xalapa. (QA267.B76)

**6. Fundamentos de Compiladores: Cómo Traducir al Lenguaje Máquina.**

Karen A. Lemone.

Editorial Continental.

Se encuentra en las bibliotecas: USBI-Xalapa, FISMATIN-Xalapa y ECONOMIA-Xalapa.  
(QA76.76.C65 L45)

**7. Construcción de Compiladores: Principios y Práctica.**

Kenneth C. Loudon.

San Jose State University.

Ed. Thomson (2004).

Se encuentra en la biblioteca: USBI-Xalapa y ECONOMIA-Xalapa. (QA76.76.C65 L68  
C6).

**8. Engineering a Compiler.**

Keith D. Cooper and Linda Torczon.

Morgan Kaufman; 1st edition (September 2003).

**9. Introduction to the Theory of Computation (Second Edition)**

Michael Sipser.

Ed. Thomson (2006).

## EVALUACIÓN

• Dos exámenes parciales (20% cada uno)	-----	40%
• Programas computacionales (Proyecto)	-----	40%
• Participación en clase (individual y equipo), Dicha participación incluye trabajos de investigación redactados de forma adecuada	-----	10%
• Prácticas individuales y por equipo	-----	10%

### Importante:

- Para poder exentar el examen ordinario, la mínima calificación parcial deberá ser de 8. Dicha calificación estará conformada por cada uno de los criterios de evaluación arriba descritos.
- Para tener derecho a presentar los exámenes parciales, deberán cubrir como mínimo el 80% de las asistencias. Se pasará lista 15 minutos después de iniciar la clase. Pasado este tiempo, si el estudiante no está presente, se considerará inasistencia (no se consideran retardos).
- El **primer examen parcial** se aplicará el **viernes 8 de Octubre de 2010** y el **segundo examen parcial** el **jueves 18 de Noviembre de 2010**.
- La fecha de entrega de los programas de cómputo será exclusivamente los siguientes días: la **primera parte** del proyecto deberá entregarse el **martes 5 de Octubre de 2010** y la **segunda parte** el **martes 16 de Noviembre de 2010**, durante la clase. No se recibirán programas enviados vía electrónica.
- Para integrar la calificación de las personas que presenten examen ordinario, se promediará la calificación de dicho examen con sus participaciones, prácticas y programas computacionales entregados durante el curso. En otras palabras, el examen ordinario valdrá un máximo de 40% de la calificación final mientras que el 60% restante estará integrado por los criterios de evaluación arriba descritos.
- En el examen extraordinario se aplicará el criterio anterior.
- Para tener derecho a presentar el examen ordinario, el alumno deberá haber cubierto el 80% de asistencias a lo largo del curso. Para poder presentar el examen extraordinario, es necesario haber cubierto el 65% de las asistencias totales.
- El reporte escrito de los programas de cómputo, deberá contener los siguientes puntos.
  1. Introducción. En esta sección se deberá proporcionar el contexto del problema.

2. Planteamiento del problema. En esta sección se describirá el problema a resolver.
3. Solución
  - Análisis. En esta sección se identificarán las posibles soluciones al problema.
  - Diseño. En esta sección se propondrán los métodos para solucionar el problema.
4. Pruebas y Discusión. En esta sección se presentarán los resultados de la implementación así como un amplio y profundo análisis de los mismos.
5. Conclusiones. En esta sección se presentarán las resoluciones obtenidas a partir del estudio y solución del problema propuesto.
6. Referencias. Esta sección contendrá la lista de referencias bibliográficas utilizadas para la presentación de los puntos anteriores.

**Descripción del Proyecto** (Tomado del libro “Fundamentos de Compiladores”. Autor: Karen A. Lemone. Páginas: 47-48 y 80-81).

## Primera Parte

### Análisis Lexicográfico

- Los siguientes son tokens de un muy pequeño subconjunto de Ada:

*comienza*

*termina*

*;*

*:=*

*(*  
*)*

*+*

*-*

*\**

*/*

*mod*

*rem*

*identificadores*

*enteros*

donde un *identificador* se describe como una letra seguida por un número arbitrario de letras o dígitos y una *constante entera* (*enteros*) es una secuencia de dígitos.

- Determinar las clasificaciones para los tokens anteriores y escribir un analizador lexicográfico que los encuentre y clasifique. Para ello hay que hacer clasificaciones diferentes para + y – (que tal vez convenga denominar SumOps) y para \*,/,mod y rem (que quizá convenga llamar MulOps) facilitarán la tarea del analizador gramatical posteriormente.

- Ejecutar su analizador lexicográfico en (a) el programa siguiente, y (b) un programa diseñado por usted.

```
comienza
a:=b3;
xyz := a + b + c
      - p / q;
a := xyz * (p + q);
p:= a - xyz - p;
termina;
```

- Su salida deberá mostrar El lexema de entrada y su tipo como en el ejemplo siguiente:

<b>Tipo</b>	<b>Lexema</b>
Palabra clave	Comienza
Identificador	a
AsigOp	:=
(etc.)	

Nota: Se puede elegir *comienza* como un identificador

## Segunda Parte del Proyecto

### Análisis Sintáctico

- Considere la siguiente gramática:

Programa	→	<b>comienza</b> SecuenciaDeSentencias <b>termina</b> ;
SecuenciaDeSentencias	→	Sentencia {Sentencia}
Sentencia	→	SentenciaSimple
SentenciaSimple	→	SentenciaDeAsignación
SentenciaDeAsignación	→	Nombre := Expresión;
Nombre	→	NombreSimple
NombreSimple	→	Identificador
Expresión	→	Relación
Relación	→	ExpresiónSimple
ExpresiónSimple	→	Término { OperadorSuma Término}
Término	→	Factor {OperadorMultiplicación Factor}
Factor	→	Primario
Primario	→	Nombre   LiteralNumérica   (Expresión)
OperadorSuma	→	+   -
OperadorMultiplicación	→	*   /   mod   rem
LiteralNumérica	→	LiteralDecimal
LiteralDecimal	→	Entero

- Diseñe un analizador sintáctico LR(1) que implementará esta gramática. Ejecute su analizador sintáctico con el programa siguiente, y en un programa de su propio diseño:

```
comienza
a:= b3;
xyz := a + b + c
          - p / q;
a := xyz * (p + q);
p := a - xyz - p;
termina;
```