Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería Ciencias y Sistemas

ESTRUCTURACIÓN DE LOS LABORATORIOS Y DOCUMENTACIÓN DE APOYO DE LOS CURSOS: LENGUAJES FORMALES Y DE PROGRAMACIÓN, ORGANIZACIÓN DE LENGUAJES Y COMPILADORES 1, ORGANIZACIÓN DE LENGUAJES Y COMPILADORES 2, APLICANDO LA METODOLOGÍA DEL IT CENTER INDIA – GUATEMALA, DE LA CARRERA DE INGENIERÍA EN CIENCIAS Y SISTEMAS DE LA FACULTAD DE INGENIERÍA DE LA UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

Lisbeth Paola de María Ricart Sandoval

Asesorada por el Ingeniero Jorge Armín Mazariegos

Guatemala, enero de 2010

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÌA

ESTRUCTURACIÓN DE LOS LABORATORIOS Y DOCUMENTACIÓN DE APOYO DE LOS CURSOS: LENGUAJES FORMALES Y DE PROGRAMACIÓN, ORGANIZACIÓN DE LENGUAJES Y COMPILADORES 1, ORGANIZACIÓN DE LENGUAJES Y COMPILADORES 2, APLICANDO LA METODOLOGÍA DEL IT CENTER INDIA – GUATEMALA, DE LA CARRERA DE INGENIERÍA EN CIENCIAS Y SISTEMAS DE LA FACULTAD DE INGENIERÍA DE LA UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

TRABAJO DE GRADUACIÓN

PRESENTADO A LA JUNTA DIRECTIVA DE LA
FACULTAD DE INGENIERÍA
POR:

LISBETH PAOLA DE MARÍA RICART SANDOVAL

ASESORADA POR EL ING. JORGE ARMIN MAZARIEGOS

AL CONFERÍRSELE EL TÍTULO DE
INGENIERA EN CIENCIAS Y SISTEMAS
GUATEMALA, ENERO DE 2010

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

NÓMINA DE JUNTA DIRECTIVA

DECANO	Ing. Murphy Olympo Paiz Recinos
VOCAL I	Inga. Glenda Patricia García Soria
VOCAL II	Inga. Alba Maritza Guerrero de López
VOCAL III	Ing. Miguel Ángel Dávila Calderón
VOCAL IV	Br. José Milton De León Bran
VOCAL V	Br. Isaac Sultan Mejía
SECRETARIA	Inga. Marcia Ivónne Véliz Vargas

TRIBUNAL QUE PRACTICÓ EL EXAMEN GENERAL PRIVADO

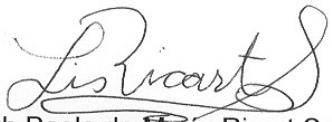
DECANO	Ing. Murphy Olympo Paiz Recinos
EXAMINADORA	Inga. Floriza Felipa Ávila Pesquera de Medinilla
EXAMINADOR	Ing. Marlon Pérez Türk
EXAMINADORA	Inga. Sonia Yolanda Castañeda Ramírez
SECRETARIA	Inga. Marcia Ivónne Véliz Vargas

HONORABLE TRIBUNAL EXAMINADOR

Cumpliendo con los preceptos que establece la ley de la Universidad de San Carlos de Guatemala, presento a su consideración mi trabajo de graduación titulado:

ESTRUCTURACIÓN DE LOS LABORATORIOS Y DOCUMENTACIÓN DE APOYO DE LOS CURSOS: LENGUAJES FORMALES Y DE PROGRAMACIÓN, ORGANIZACIÓN DE LENGUAJES Y COMPILADORES 1, ORGANIZACIÓN DE LENGUAJES Y COMPILADORES 2, APLICANDO LA METODOLOGÍA DEL IT CENTER INDIA – GUATEMALA, DE LA CARRERA DE INGENIERÍA EN CIENCIAS Y SISTEMAS DE LA FACULTAD DE INGENIERÍA DE LA UNIVERSIDAD DE SAN CARLOS DE GUATEMALA,

tema que me fuera asignado por la Dirección de la Escuela de Ingeniería en Ciencias y Sistemas, en septiembre 2009.



Lisbeth Paola de María Ricart Sandoval

Guatemala, 24 de Octubre 2009

Ingeniera
Norma Ileana Sarmiento
Directora Unidad EPS
Dirección de EPS
Facultad de Ingeniería
USAC

Respetable Ingeniera Sarmiento:

Por este medio hado de su conocimiento que he revisado el trabajo de graduación del estudiante **LISBETH PAOLA DE MARÍA RICART SANDOVAL**, titulado: "**ESTRUCTURACIÓN DE LOS LABORATORIOS Y DOCUMENTACIÓN DE APOYO DE LOS CURSOS: LENGUAJES FORMALES Y DE PROGRAMACIÓN, ORGANIZACIÓN DE LENGUAJES Y COMPILADORES 1, ORGANIZACIÓN DE LENGUAJES Y COMPILADORES 2 APLICANDO LA METODOLOGÍA DEL IT CENTER INDIA-GUATEMALA, DE LA CARRERA DE INGENIERÍA EN CIENCIAS Y SISTEMAS DE LA FACULTAD DE INGENIERÍA DE LA UNIVERSIDAD DE SAN CARLOS DE GUATEMALA**", y a mi criterio, el mismo cumple con los objetivos propuestos para su desarrollo, según el protocolo.

Agradeciendo su atención a la presente, aprovecho la oportunidad para suscribirme,

Atentamente,

Jorge Armin Mazariegos Rabanales
ING. EN CIENCIAS Y SISTEMAS
M.c. ADMINISTRADOR DE TECNOLOGIA
COLEGIADO 5547

Ing. Jorge Armin Mazariegos
Asesor del Trabajo de Graduación
Ingeniería en Ciencias y Sistemas

Universidad de San Carlos de Guatemala
Facultad de Ingeniería

UNIDAD DE E.P.S.

Guatemala, 04 de noviembre de 2009.
Ref.EPS.DOC.1572.11.09.

Inga. Norma Ileana Sarmiento Zeceña de Serrano
Directora Unidad de EPS
Facultad de Ingeniería
Presente

Estimada Ingeniera Sarmiento Zeceña.

Por este medio atentamente le informo que como Supervisora de la Práctica del Ejercicio Profesional Supervisado, (E.P.S) de la estudiante universitaria de la Carrera de Ingeniería en Ciencias y Sistemas, **Lisbeth Paola Ricart Sandoval** Carné No. **200312735** procedí a revisar el informe final, cuyo título es **"ESTRUCTURACIÓN DE LABORATORIOS Y DOCUMENTACIÓN DE APOYO DE LOS CURSOS: LENGUAJES FORMALES Y DE PROGRAMACIÓN, ORGANIZACIÓN DE LENGUAJES Y COMPILADORES 1, ORGANIZACIÓN DE LENGUAJES Y COMPILADORES 2 APLICANDO LA METODOLOGÍA DEL IT CENTER INDIA- GUATEMALA, DE LA CARRERA DE INGENIERÍA EN CIENCIAS Y SISTEMAS DE LA FACULTAD DE INGENIERÍA DE LA UNIVERSIDAD DE SAN CARLOS DE GUATEMALA".**

En tal virtud, **LO DOY POR APROBADO**, solicitándole darle el trámite respectivo.

Sin otro particular, me es grato suscribirme.

Atentamente,

"Id y Enseñad a Todos"

Inga. Floriza Felipa Ávila Pesquera de Medinilla
Supervisora de EPS
Área de Ingeniería en Ciencias y Sistemas

SUPERVISOR (A) DE EPS
Unidad de Prácticas de Ingeniería y EPS
Facultad de Ingeniería

FFAPdM/RA

Universidad de San Carlos de Guatemala
Facultad de Ingeniería

UNIDAD DE E.P.S.

Guatemala, 04 de noviembre de 2009.
Ref.EPS.D.769.11.09.

Ing. Marlon Antonio Pérez Turck
Director Escuela de Ingeniería Ciencias y Sistemas
Facultad de Ingeniería
Presente

Estimado Ingeniero Perez Turck.

Por este medio atentamente le envío el informe final correspondiente a la práctica del Ejercicio Profesional Supervisado, (E.P.S) titulado **"ESTRUCTURACIÓN DE LABORATORIOS Y DOCUMENTACIÓN DE APOYO DE LOS CURSOS: LENGUAJES FORMALES Y DE PROGRAMACIÓN, ORGANIZACIÓN DE LENGUAJES Y COMPILADORES 1, ORGANIZACIÓN DE LENGUAJES Y COMPILADORES 2 APLICANDO LA METODOLOGÍA DEL IT CENTER INDIA- GUATEMALA, DE LA CARRERA DE INGENIERÍA EN CIENCIAS Y SISTEMAS DE LA FACULTAD DE INGENIERÍA DE LA UNIVERSIDAD DE SAN CARLOS DE GUATEMALA"**, que fue desarrollado por la estudiante universitaria **Lisbeth Paola Ricart Sandoval** Carné No. **200312735** quien fue debidamente asesorada por el Ing. Jorge Armin Mazariegos y supervisada por la Inga. Floriza Felipa Ávila Pesquera de Medinilla

Por lo que habiendo cumplido con los objetivos y requisitos de ley del referido trabajo y existiendo la aprobación del mismo por parte del Asesor y de la Supervisora de EPS, en mi calidad de Directora apruebo su contenido solicitándole darle el trámite respectivo.

Sin otro particular, me es grato suscribirme.

Atentamente,
"Id y Enseñad a Todos"

Inga. Norma Ileana Sarmiento Zeceña de Serrano
Directora Unidad de EPS

DIRECCIÓN
Unidad de Prácticas de Ingeniería y EPS

NISZ/ra

Universidad San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería en Ciencias y Sistemas

Guatemala, 28 de Octubre de 2009

Ingeniero
Marlon Antonio Pérez Turk
Director de la Escuela de Ingeniería
En Ciencias y Sistemas

Respetable Ingeniero Pérez:

Por este medio hago de su conocimiento que he revisado el trabajo de graduación de la estudiante **LISBETH PAOLA DE MARIA RICART SANDOVAL,** titulado: **"ESTRUCTURACION DE LOS LABORATORIOS Y DOCUMENTACION DE APOYO DE LOS CURSOS: LENGUAJES FORMALES Y DE PROGRAMACION, ORGANIZACIÓN DE LENGUAJES Y COMPILADORES 1, ORGANIZACIÓN DE LENGUAJES Y COMPILADORES 2 APLICANDO LA METODOLOGIA DEL IT CENTER INDIA – GUATEMALA, DE LA CARRERA DE INGENIERIA EN CIENCIAS Y SISTEMAS DE LA FACULTAD DE INGENIERIA DE LA UNIVERSIDAD DE SAN CARLOS DE GUATEMALA",** y a mi criterio el mismo cumple con los objetivos propuestos para su desarrollo, según el protocolo.

Al agradecer su atención a la presente, aprovecho la oportunidad para suscribirme,

Atentamente,

Ing. Carlos Alfredo Azurdia
Coordinador de Privados
y Revisión de Trabajos de Graduación

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
INGENIERIA EN CIENCIAS Y SISTEMAS
COORDINACION TESIS

ESCUELA DE CIENCIAS Y SISTEMAS

UNIVERSIDAD DE SAN CARLOS
DE GUATEMALA

FACULTAD DE INGENIERÌA
ESCUELA DE CIENCIAS Y SISTEMAS
TEL: 24767644

El Director de la Escuela de Ingeniería en Ciencias y Sistemas de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer el dictamen del asesor con el visto bueno del revisor y del Licenciado en Letras, de trabajo de graduación titulado ***"ESTRUCTURACIÓN DE LOS LABORATORIOS Y DOCUMENTACIÓN DE APOYO DE LOS CURSOS: LENGUAJES FORMALES Y DE PROGRAMACIÓN, ORGANIZACIÓN DE LENGUAJES Y COMPILADORES 1, ORGANIZACIÓN DE LENGUAJES Y COMPILADORES 2, APLICANDO LA METODOLOGÍA DEL IT CENTER INDIA – GUATEMALA, DE LA CARRERA DE INGENIERÍA EN CIENCIAS Y SISTEMAS DE LA FACULTAD DE INGENIERÍA DE LA UNIVERSIDAD DE SAN CARLOS DE GUAEMALA"***, *presentado por la estudiante LISBETH PAOLA DE MARÍA RICART SANDOVAL, aprueba el presente trabajo y solicita la autorización del mismo.*

"ID Y ENSEÑAD A TODOS"

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DIRECCION DE
INGENIERIA EN CIENCIAS
Y SISTEMAS

Ing. Marlon Antonio Pèrez Turk
Director, Escuela de Ingeniería en Ciencias y Sistemas

Guatemala, 29 de enero 2010

Universidad de San Carlos
de Guatemala

Facultad de Ingeniería
Decanato

Ref. DTG.031.2010

El Decano de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer la aprobación por parte del Director de la Escuela de Ingeniería en Ciencias y Sistemas, al trabajo de graduación titulado: **ESTRUCTURACIÓN DE LOS LABORATORIOS Y DOCUMENTACIÓN DE APOYO DE LOS CURSOS: LENGUAJES FORMALES Y DE PROGRAMACIÓN, ORGANIZACIÓN DE LENGUAJES Y COMPILADORES 1, ORGANIZACIÓN DE LENGUAJES Y COMPILADORES 2, APLICANDO LA METODOLOGÍA DE IT CENTER INDIA – GUATEMALA, DE LA CARRERA DE INGENIERÍA EN CIENCIAS Y SISTEMAS DE LA FACULTAD DE INGENIERÍA DE LA UNIVERSIDAD DE SAN CARLOS DE GUATEMALA,** presentado por la estudiante universitaria **Lisbeth Paola de María Ricart Sandoval,** procede a la autorización para la impresión del mismo.

IMPRÍMASE.

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DECANO
FACULTAD DE INGENIERIA

Ing. Murphy Olympo Paiz Recinos
DECANO

Guatemala, enero de 2010

ACTO QUE DEDICO A:

Dios:	Quién me dio la fortaleza para llegar hasta el final y poder culminar esta etapa y sin quien no hubiera sido posible alcanzar este triunfo.
Mi madre:	Gladys Sandoval, por su ejemplo de superación incasable, por su comprensión y confianza, por su amor y amistad incondicional, porque sin su apoyo no hubiera sido posible la culminación de mi carrera profesional.
Mi hermano:	Jónathan Ricart, porque ha sido un amigo, un compañero fiel en el camino hasta aquí recorrido.
Mi familia	Por la colaboración brindada durante mi vida, y su apoyo incondicional.
Mis amigos	Por su amistad sincera y apoyo en todos los momentos de mi vida.
Mi novio	Luis Arturo Andrade, por motivarme a seguir adelante en todo momento y brindarme su apoyo incondicional.

AGRADECIMIENTOS A:

Mi asesor:	Ing. Armin Mazariegos, por el tiempo dedicado a la realización de este proyecto.
Universidad San Carlos de Guatemala, Facultad de Ingeniería:	Por haberme brindado una excelente formación académica durante toda la carrera profesional.
Colaboradores	A los ingenieros, auxiliares y alumnos de los cursos realizados en este proyecto, por sus sugerencias y aportes que lo enriquecieron, mil gracias.

ÍNDICE GENERAL

ÍNDICE DE ILUSTRACIONES

FIGURAS

TABLAS

GLOSARIO

Instructor Guideline	En inglés Guía del Instructor, se define como una guía estructurada que tiene como fin estandarizar un módulo.
EPS	Ejercicio Profesional Supervisado.
Módulo	Unidad de medida por medio de la cual se agrupan todas las actividades relacionadas para poder impartir un curso.
Sesión	Se define como un conjunto de temas que se agrupan con el fin de ser impartidos por un instructor en una unidad determinada de tiempo.

RESUMEN

Los cambios en las tecnologías de comunicación e información están en constante evolución y la única manera de mantener su paso es actualizando los contenidos que se imparten en la Escuela de Ingeniería en Ciencias y Sistemas. Así mismo es necesario velar por la calidad de la enseñanza impartida por esta institución.

Todo esto nos lleva a la necesidad de crear un sistema para definir de manera clara y detallada la estructura que debe llevar cada curso de la carrera de Ingeniería en Ciencias y Sistemas. Para este Ejercicio Profesional Supervisado (EPS), se busca crear dicha estructura por medio de una documentación detallada. Esta documentación incluye referencias de apoyo sobre artículos de interés de los eventos más recientes sobre dichos temas.

El primer paso para crear una plataforma sobre la cual se pueda crear una guía para todos los cursos fue la creación de los *Instructor Guidelines*, documentos que son utilizados por el Centro Tecnológico de la India para manejar e impartir los cursos que ofrecen. El personal del Centro Tecnológico aporto sus valiosos conocimientos y participo en la supervisión de la creación de dichos documentos, basado en las experiencias en capacitación obtenidas por la empresa transnacional TATA *Consultancy Services*.

Se impartió un curso intensivo en la elaboración de I*nstructor Guidelines* durante 4 meses, en el período de enero a abril del 2008. Con base a este conocimiento se elaboraron *Instructor Guidelines* para los cursos de Lenguajes

Formales y de Programación, Organización de Lenguajes y Compiladores 1, Organización de Lenguajes y Compiladores 2.

Posteriormente a la creación de estos documentos se crearon guías para las clases de Lenguajes Formales y de Programación, Organización de Lenguajes y Compiladores 1, Organización de Lenguajes y Compiladores 2, por medio de un libro de texto que contiene las unidades más importantes de cada curso.

Los documentos descritos en los párrafos anteriores fueron elaborados en el presente Ejercicio Profesional Supervisado, sobre los cuales se aplicó la metodología enseñada por el Centro Tecnológico de la India.

OBJETIVOS

General

Crear una guía en la cual se pueda definir la estructura completa de tres cursos impartidos por la Escuela de Ingeniería en Ciencias y Sistemas aplicando la metodología de los documentos denominados *Instructor Guidelines*, con el fin de estandarizar su contenido. También proporcionar material de apoyo para tres cursos en formato de un libro de texto, conteniendo las unidades más importantes de cada uno.

Específicos:

1. Elaborar un "*Instructor Guideline*", que por medio de módulos pueda estructurar el contenido y las sesiones de un curso, también definir la forma en que se evalúa e imparten los siguientes:
    - Lenguajes Formales y de Programación
    - Organización de Lenguajes y Compiladores 1
    - Organización de Lenguajes y Compiladores 2

2. Proporcionar material gráfico para detallar cada sesión en los Instructor Guidelines y su contenido.

3. Desarrollar documentación de apoyo proporcionando al lector un texto de referencia que contiene las unidades más importantes de cada curso, este material se desarrollo para los siguientes cursos:
    - Lenguajes Formales y de Programación
    - Organización de Lenguajes y Compiladores 1
    - Organización de Lenguajes y Compiladores 2

4. Documentar una serie de artículos relacionados en cada unidad del curso, para el material de apoyo de clase

INTRODUCCIÓN

Mantener el nivel educativo de los cursos impartidos en la Escuela de Ingeniería en Ciencias y Sistemas es crucial para garantizar un nivel de enseñanza de alto nivel como el que ha caracterizado a la Facultad de Ingeniería. De aquí surge la necesidad de estandarizar el contenido de los cursos para asegurar su nivel de calidad.

Es muy importante también mantener actualizados los contenidos de los cursos periódicamente, especialmente en una carrera que cambia constantemente como sucede en la carrera de ciencias y sistemas.

Observando los problemas que la Escuela ha tenido para coordinar diferentes secciones de un mismo curso y el contenido impartido tanto en la clase como en los laboratorios, se ve en la necesidad de crear un sistema por medio del cual se pueda estandarizar el contenido de los mismos.

Como solución a este problema la Escuela de Ingeniería en Ciencias y Sistemas trabajando en conjunto con la empresa TATA *CONSULTANCY SERVICES* y por medio del proyecto *India-Guatemala IT Education Centre of Excellence*, creó un curso impartido a los estudiantes para poder realizar *Instructor Guidelines*. En este curso se imparte la metodología que el *IT Education Centre of Excellence* ha utilizado para impartir sus propios cursos, mostrando la manera en que organizan y estructuran cada módulo.

Se le indicó a la persona que trabajó en este proyecto que iba a tomar 3 cursos del área de Ciencias y Sistemas y que iba a aplicar esta nueva metodología. Cada *Instructor Guideline* fue creado de acuerdo al curso impartido por el señor Mrutunjaya Panda, que junto a catedráticos de cada curso ayudó al personal del proyecto a crear los documentos.

Este documento comienza explicando en qué consiste esta metodología y en los capítulos posteriores se aplica la metodología a los cursos de Lenguajes Formales y Programación, Organización de Lenguajes y Compiladores 1, Organización de Lenguajes y Compiladores 2, asimismo la documentación respectiva de cada curso.

1. MARCO TEÓRICO

1.1 Principios de Pedagogía

De la definición de los objetivos principales de este proyecto surge la necesidad de definir algunos conceptos básicos de pedagogía.

La Pedagogía es la rama de la ciencia que se puede resumir en la frase “el arte de enseñar”. Esta ciencia busca tanto definir como aplicar las técnicas y prácticas que se realizan con el fin de facilitar el proceso de aprendizaje. Se busca optimizar el proceso de aprendizaje con el objetivo de beneficiar a ambos el educador y el objetivo a ser educado.

Ya que mediante esta ciencia estamos llegando al pensamiento humano, es imposible definir esta disciplina sin mencionar a la psicología. La psicología como ciencia individual busca estudiar los procesos mentales, mediante el estudio del comportamiento humano se pueden crear programas con el fin de optimizarlo.

De acuerdo a Chickerin y Gamson, los principios de la pedagogía son siete.

a. Propiciar el contacto entre estudiantes y profesores.

Aquí vemos el elemento fundamental de la psicología en la definición de la pedagogía. Es muy importante que el profesor logre crear un vínculo con sus estudiantes, la dinámica que se practica aquí varía dependiendo del número de

estudiantes que se tiene. Si el estudiante se siente cómodo se logra crear un ambiente positivo en el cual es más fácil impartir el conocimiento.

b. Fomentar la cooperación entre los estudiantes.

Nunca se debe de olvidar que la interacción entre los estudiantes es muy importante en el proceso educativo. El profesor siempre debe incentivar el trabajo en equipo

c. Propiciar el aprendizaje activo.

El aprendizaje activo se refiere al grado de participación del estudiante. Este objetivo se puede derivar del primer y segundo objetivo. Si el estudiante siente un vínculo con el profesor y se ha creado un ambiente cooperativo entre los alumnos la participación del alumno viene de manera natural. La participación tiene como objetivo tanto ayudar al alumno como al profesor. De esta manera se logra una retroalimentación que se obtiene del alumno, tanto las opiniones como las vivencias pueden complementar el programa de estudios base en una clase.

d. Proporcionar retroalimentación a tiempo.

Es importante saber manejar la retroalimentación que se obtiene del estudiante y saber en qué casos es necesaria una respuesta inmediata. De esta manera se incentiva este tipo de comportamiento mejorando la sinergia del grupo completo.

e. Enfatizar el uso apropiado del tiempo.

La planeación del tiempo es crucial en el proceso de aprendizaje, los beneficios que obtenemos de una buena planeación van desde impartir el contenido deseado hasta aliviar la carga de trabajo en los alumnos.

Tanto el alumno como el profesor se benefician de un programa bien definido en términos de tiempo. También es muy importante el manejo del tiempo en cada una de las sesiones de clase.

f. Propiciar altas expectativas al estudiante.

Al incentivar al estudiante a participar e intentar obtener un poco más de lo que se le pide normalmente a un alumno estamos creando metas más altas para él. Es importante hacerle saber al alumno que estas metas pueden alcanzarse y que los beneficios que se obtienen al lograr estas metas son muchos.

g. Respetar los diferentes estilos de aprendizaje

Este principio afirma que un programa de estudio, o un método de estudio no puede mantenerse estático. Es importante considerar la necesidad de cambiar e incluso aceptar otros estilos de aprendizaje. Mediante la observación activa podemos incorporar al programa básico de estudios ideas que podrían beneficiarlo.

1.2 Proceso de Enseñanza – Aprendizaje

Educación

Es un proceso social en el cual se busca que el alumno desarrolle sus capacidades mentales, técnicas o físicas para aumentar su habilidad en determinado campo o tema.

Enseñanza

El sistema de enseñanza se basa en el modelo Estudiante-Profesor. El elemento llamado profesor tiene como objetivo impartir el conocimiento que posee sobre cierta área, la manera en que imparte dicho conocimiento

demanda de él creatividad y habilidad. El segundo elemento en este modelo es llamado estudiante y es el sujeto al cual se le busca impartir conocimiento. Para que este sistema funcione se necesita del compromiso de ambos componentes.

Aprendizaje

A su nivel más bajo se define como el proceso de adquirir conocimientos. La naturaleza de este conocimiento abarca tanto aspectos intelectuales como físicos y emocionales. Si el alumno logra aumentar dicho nivel de conocimiento podemos afirmar que se está logrando llegar al aprendizaje.

El tipo de aprendizaje que se busca en este proyecto es el aprendizaje intelectual, el objetivo que se busca es impartir conocimiento. Bajo este principio definimos lo que es el proceso de enseñanza-aprendizaje, en este proceso tenemos tres elementos principales. El primer elemento es el estudiante y el segundo es el alumno. El tercer elemento es lo que llamamos cuerpo de conocimiento, que se puede definir como el resultado del proceso de aprendizaje. El cuerpo de conocimiento en sí es el conocimiento adquirido después de impartirse una clase, conocimiento que aumenta su capacidad mental o física.

Podemos ver que el proceso de enseñanza-aprendizaje es bastante complejo e involucra una gran cantidad de variables. Es por esa razón que resulta crucial tener una guía sobre la cual se pueda basar un plan para impartir conocimiento.

De los tres elementos principales descritos en el proceso enseñanza-aprendizaje existe tres interacciones directas que deben ser detalladas:

Interacción profesor - conocimiento

El elemento profesor debe tener claro cuáles son los recursos que tiene a su disposición para impartir conocimiento.

Interacción alumno - conocimiento

El alumno es un elemento crucial en este sistema y sin su colaboración no se puede completar el proceso de aprendizaje.

Interacción profesor - alumno

Es muy importante que el profesor pueda crear un vínculo con el estudiante para involucrarlo en el proceso y despertar en él un deseo de aprender.

Acción didáctica

En el proceso de enseñanza-aprendizaje tenemos dos elementos que están íntimamente relacionados, en este proceso si definimos la acción por medio de la cual se imparte el conocimiento estamos describiendo a la acción didáctica. Para ejecutar la acción didáctica debemos conocer sus tres componentes principales:

1. **Planteamiento.** En esta etapa se definen los objetivos que se buscan lograr, a su vez definimos el conjunto de actividades que se deben realizar para orientarnos hacia dichos objetivos.

2. **Ejecución.** Basándonos en el plan descrito anteriormente y en los recursos que el profesor tiene a su disposición, se comienza el proceso de impartir conocimiento.

3. **Evaluación.** En esta etapa observamos si se han cumplido los objetivos propuestos. En base a esta observación podemos realizar cambios a nuestro planteamiento inicial.

1.3 Método de clases magistrales

Existen diferentes maneras en que un profesor puede impartir una clase, podemos definir dos métodos principales. El primer método es lo que se llama una clase activa, en este tipo de método se incentiva al estudiante a participar en la clase. Esto se logra haciendo directamente preguntas a los estudiantes o incentivando la participación directa.

El segundo método se llama clase magistral, aquí se tiene un sistema diferente a la clase activa ya que lo que se busca es que el profesor imparta una clase sin ser interrumpido por los estudiantes. El profesor comienza a dar la clase y los estudiantes pueden tomar nota de los elementos más importantes, pero su participación es pasiva ya que no se involucra directamente en la conversación.

Los beneficios principales de una clase magistral son la facilidad de administrar el tiempo y la capacidad de impartir mayor cantidad de material en una unida de tiempo determinada. Es por esta razón que este método se utiliza con mayor frecuencia. Con este método es más fácil para el profesor preparar las sesiones a impartir.

Los elementos principales que debemos manejar durante una clase magistral son:

- **La voz**

Debemos tomar en cuenta el lugar en el cual se va a impartir la clase, si el lugar es muy grande es posible que tengamos necesidad de un micrófono. Por otro lado si el lugar que utilizamos es pequeño o la audiencia es de tamaño reducido tener un micrófono solamente molestaría al público.

- **El cuerpo**

El uso adecuado del cuerpo al realizar una exposición es muy importante. Es más fácil para el público seguir a una persona que está en constante movimiento ya que evita que la exposición se vuelva aburrida. Por otro lado no debemos exagerar nuestros movimientos ya que el público pierde el interés en el contenido.

- **Las imágenes visuales**

El uso de diferentes ayudas visuales nos permite captar la atención del público con mayor facilidad y en muchas ocasiones facilita el entendimiento de algunos temas.

- **Invitados**

Traer como invitado a un experto en cierta materia puede ser beneficioso para una exposición.

- **El plan de exposición**

Es importante mostrarle al público al inicio cuál es el plan completo a seguir durante la exposición, ya que de esta manera puede saber de antemano que temas son de mayor interés para él.

- **El texto de la clase**

En algunas ocasiones puede ser de gran ayuda proveer al público con material escrito para que pueda seguir con mayor facilidad la exposición.

- **Resumen final**

Nos ayuda a repetir elementos importantes de la exposición.

1.4 Aprendizaje práctico por medio de laboratorios

Existen ciertos temas que no pueden impartirse únicamente como una clase teórica, temas en los cuáles la parte práctica es tan importante o en algunos casos más importante. En este tipo de casos es una buena idea pensar en la utilización de un laboratorio, es una manera sencilla de involucrar directamente al estudiante y ayudarle a comprender conceptos teóricos.

El lado negativo de este método es su alto costo en ciertos casos, si no se tienen los recursos adecuados y se tiene un grupo numeroso de estudiantes el laboratorio no es efectivo. Por esta razón se debe realizar un estudio detallado de quien será el grupo de estudiantes al cual el laboratorio está destinado y también se debe saber con qué tipo de equipo se puede contar.

2. DEFINICIÓN DE UNA GUÍA DEL INSTRUCTOR

Un *Instructor Guideline* es un documento que tiene como objetivo estructurar, estandarizar y detallar las diferentes actividades que se realizan al impartir una clase. Buscan también definir de manera clara el contenido que se impartirá y el tiempo en el cual se debe hacer.

El proceso de enseñanza requiere una planeación del tiempo bien definida ya que es el elemento principal que se busca optimizar. Debemos saber qué tipo de actividades se deben realizar y con qué frecuencia se debe realizar cada una.

La planificación se detalla en un documento escrito, y dicha planeación se realiza en base a una serie de módulos bien definidos. La estructura de este documento y la manera en la cual se trabaja fue probada y utilizada por varios años por la empresa TATA.

Cada *Instructor Guideline* va dirigido directamente a la persona que está encargada de impartir la clase, ya sea un auxiliar o un catedrático. Cada una de las secciones de este documento provee una herramienta importante cuya finalidad es facilitar la labor de impartir clases.

La planificación del proceso de enseñanza debe poseer las siguientes características:

- **Flexibilidad.** Debemos tener en cuenta que los planes que se hacen pueden ser adaptados a los cambios que sean necesarios.

- **Realidad.** Se debe saber exactamente con que recursos se cuenta en el lugar que se imparte una clase, a partir de este conocimiento se puede crear un plan efectivo para lograr los objetivos planteados para cada módulo.

- **Precisión.** Cada guía debe definir cada elemento de manera clara y precisa. Si se da lugar a la ambigüedad se puede llegar a tener problemas de entendimiento y por consecuencia la guía se aleja de los objetivos planteados.

2.1 Objetivos de una Guía de Instructor

Los objetivos principales del *Instructor Guideline* son estructurar, ordenar, estandarizar y especificar el contenido que se utiliza para impartir una clase o un laboratorio.

Esta guía permite llevar el control con más facilidad de un curso cuando se tiene diferentes secciones u horarios. Evita la necesidad de verificar constantemente entre los diferentes catedráticos sobre el contenido y el ritmo impartido en el curso. Al tener un punto de referencia claro sobre el cual respaldarse es más fácil llevar el control del curso y evitar problemas de tiempo y de contenido debido a la planeación.

2.2 Elementos de una Guía de Instructor

Un *Instructor Guideline* está compuesto por varios elementos, estos son los componentes principales:

- Información general del curso.
- Distribución de horas y actividades.
- Forma de evaluación.
- Detalle de sesiones.
- Distribución de sesiones.
- Detalle de tareas asignadas
- Detalle de tutoriales
- Detalle de exámenes.
- Ejemplo de exámenes.
- Bibliografía.
- Anexos.

2.3 Proceso de construcción de la Guía de Instructor

Información general del curso

Este elemento trata de describir la idea general e introductoria del curso, esta debe contener:

- Nombre del curso, según el pensum, incluye el código del curso.
- Prerrequisitos, los cursos previos que son requisitos obligatorios.

- Objetivo, la meta a alcanzar para el curso.
- Módulos, componentes del curso con diferente finalidad.

Distribución de horas y actividades.

En esta sección especificamos cuantos módulos se trabajan por curso y se detalla para cada uno las diferentes actividades que se realizan y la cantidad en horas de tiempo que se tarda el instructor en impartirla.

Las actividades que se detallan son: asistencias, tareas, presentaciones, y evaluaciones. La siguiente tabla muestra un ejemplo de la distribución de horas y actividades:

Tabla I. Distribución de horas y actividades

Módulo	Teoría	Práctica	Total	Examen	Práctica de lab.	Proyectos
Módulo 1	30 horas	8 horas	38 horas	1	10	1
Total	30 horas	8 horas	38 horas	1 hora	4	1

Forma de evaluación.

Se le asigna un valor numérico a cada actividad que se realizará para llegar a un 100% de la nota total. Aquí se tiene una tabla de ejemplo:

Tabla II. Forma de evaluación

	Número	Porcentaje	Total
Tutorial	3	2.5%	7.50 %
Tareas	10	4.0%	40.00%
Proyecto	1	30.0%	30.00%
Examen	1	20.0%	20.00%
		TOTAL	100.00%

Detalle de sesiones.

Se define el número de sesiones en las cuales se divide cada módulo del curso, se muestra también la duración, definiendo si existirá alguna tarea en cada sesión.

Tabla III. Detalle de sesiones

No.	Teoría	No.	Práctica	Tarea
1	Módulo – Sesión 1			
2	Módulo – Sesión 2	1	Módulo – Lab Assignment 1	Tarea 1
3	Módulo – Sesión 3	2	Módulo – Lab Assignment 2	Tarea 2

Distribución de sesiones.

Previo a la realización de una sesión para un módulo se deben tomar en cuenta una serie de factores diferentes. El primer factor es la naturaleza del contenido que se debe incluir, para considerar esto se debe saber que cursos ha llevado anteriormente el estudiante para establecer su base de conocimiento. Otro factor importante es el tiempo de cada sesión, cada sesión está programada para impartirse en dos horas, de manera que se debe estimar el grado de dificultad del contenido o si habrá necesidad de mostrar varios ejemplos para explicar bien el tema. Finalmente se especifica en cada sesión si se estará haciendo un examen y cuál debe ser su duración.

La distribución de sesiones se realiza dividiendo el módulo completo en una cantidad fija de sesiones, en el documento debemos identificar el nombre del curso, el módulo al que pertenece, el numero de sesión, el objetivo, la distribución de temas y el objetivo del módulo.

Las sesiones pueden ser teóricas o prácticas, donde una sesión teórica constituye una clase magistral, y una sesión práctica un ejerció que se realizará durante el período de clase.

Por cada sesión teórica definida en la guía del instructor, se debe desarrollar una presentación con el contenido de dicha sesión, para obtener tanto la definición de los puntos a tratar en la sesión, y el contenido de la sesión en sí.

Detalle de tareas asignadas

En este módulo se dividen de manera similar a la distribución de sesiones las diferentes tareas que se deben realizar. Se detalla aquí: el módulo al que pertenece, el objetivo que se busca, la forma de evaluación, el material a entregar y finalmente en qué momento se debe hacer.

Detalle de tutoriales

En esta sección se describen los objetivos y sistema de evaluación de los tutoriales (conocidos como prácticas presenciales) que se realizarán en el laboratorio.

Detalle de exámenes.

Se define aquí la manera en que se debe hacer un examen para el módulo. Se detalla aquí: el módulo al que pertenece, cuando se debe realizar, el contenido a evaluar, la duración y el modo de evaluación.

Ejemplo de exámenes.

Para tener una mejor idea de los objetivos que se buscan al realizar una prueba se muestra en esta sección un ejemplo completo de un examen. Por medio de un ejemplo es más fácil apreciar de qué manera se debe crear la evaluación. Representa solamente un ejemplo de una evaluación, cada auxiliar de laboratorio deberá elaborar sus propios exámenes durante cada semestre.

Bibliografía.

Es el conjunto de material didáctico que se utiliza como base para creación del contenido incluido en cada sesión. También se detalla para tener un texto de referencia en caso de necesitar ampliar la información dada.

Anexos

En esta sección se incluyen elementos extras que complementan la guía de instructor. Pueden incluirse tutoriales de ejemplo, el programa del laboratorio y otros elementos que se consideren necesarios para mejorar la comprensión de la guía.

2.4 Forma de uso de la Guía del Instructor

Cada *Instructor Guideline* tiene como objetivo ayudar a la persona que imparte las sesiones de cada módulo. Esta ayuda abarca varios aspectos entre ellos: administración del tiempo al detallar cada sesión, el contenido a abarcar en cada sesión, el número de ejercicios y evaluaciones que se deben realizar, la manera en que se deben calificar y los objetivos que se buscan para cada módulo.

3. GUÍA DEL INSTRUCTOR DEL CURSO LENGUAJES FORMALES Y DE PROGRAMACIÓN

A continuación se presenta el desarrollo de la Guía del curso de Lenguajes Formales y de Programación de acuerdo al formato sugerido en el curso de "Estructuración de laboratorios", impartido por el señor. Mrutunjaya Panda.

3.1 Datos Generales

NOMBRE DE CURSO: Lenguajes Formales y de Programación (796)

PRE- REQUISITOS: Introducción a la Programación 1 (770)
Lógica de Sistemas (795)
Matemática de Cómputo 1 (960)

POST – REQUISITOS: Organización de Lenguajes y Compiladores 1 (777)
Organización Computacional (964)

OBJETIVO: Al finalizar el curso, el estudiante podrá diseñar compiladores o intérpretes sencillos para cualquier tipo de lenguaje, utilizando las expresiones regulares, la definición de gramáticas y el análisis a través de autómatas.

3.2 Distribución del curso

Tabla IV. Distribución del curso Lenguajes Formales y de Programación

MÓDULO	TEORÍA	PRACTICA	TOTAL	EXAMEN	ASIGNACIÓN DE LABORATORIO	PROYECTOS
LENGUAJES Y COMPILADORES PRINCIPIANTES	17 horas	9 horas	26 horas	1 ½ horas	4	2
			28 horas			

3.3 Evaluación

Ponderación de tareas, prácticas y proyectos.

Tabla V. Evaluación del curso Lenguajes Formales y de Programación

ACTIVIDAD	NÚMERO	PORCENTAJE	TOTAL
Caso de estudio	1	4.0 %	4.00 %
Tutorial	3	1.0	3.00%
Asignación de laboratorio	4	2.0 %	8.00 %
Proyecto	2	35.0 %	70.00 %
Examen	1	15.0 %	15.00 %
		TOTAL	**100.00 %**

3.4 Detalle de sesiones

Tabla VI. Sesiones del curso Lenguajes Formales y de Programación

No	Teoría	No	Práctica	Tutorial / Proyecto / Asignación
1	Lenguajes y Compiladores Principiantes- 1			
2	Lenguajes y Compiladores Principiantes- 2			Caso de Estudio – 1
3	Lenguajes y Compiladores Principiantes- 3			Asignación de Laboratorio– 1
4	Lenguajes y Compiladores Principiantes- 4	5	Laboratorio – 1	Ejercicio – 1 Tutorial – 1
6	Lenguajes y Compiladores Principiantes– 5	7	Laboratorio – 2	Asignación de Laboratorio– 2
8	Lenguajes y Compiladores Principiantes– 6	9	Laboratorio – 3	Ejercicio – 2 Proyecto – 1
10	Lenguajes y Compiladores Principiantes– 7	11	Laboratorio – 4	Ejercicio – 3
12	Lenguajes y Compiladores Principiantes– 8	13	Laboratorio – 5	Asignación de Laboratorio– 3

14	Lenguajes y Compiladores Principiantes– 9	15	Laboratorio – 6	Tutorial – 2
16	Lenguajes y Compiladores Principiantes– 10	17	Laboratorio -7	
18	Lenguajes y Compiladores Principiantes– 11			Proyecto – 2
19	Lenguajes y Compiladores Principiantes– 12			Tutorial -3
		20	Laboratorio -8	Asignación de Laboratorio– 4
--	Evaluación Final			

3.5 Distribución de sesiones

Tabla VII. Sesiones del curso Lenguajes Formales y de Programación

SESIÓN NO. 1:	**Duración: 2 horas**
OBJETIVO:	**Identificar las características por las cuales se clasifican los lenguajes de programación y los paradigmas de programación.**
CONTENIDO:	1. Lenguajes de programación 2. Clasificación de los lenguajes de programación 2.1. Según su nivel de abstracción

	2.2. Según su forma de ejecución 2.3. Según el paradigma de programación 2.3.1. Paradigma Imperativo 2.3.2. Paradigma Funcional 2.3.3. Paradigma Lógico 2.3.4. Paradigma Orientado a Objetos
SESIÓN NO. 2:	**Duración: 2 horas**
OBJETIVO:	**Introducir a los estudiantes los conceptos principales de los procesadores de lenguaje y su estructura**
CONTENIDO:	1. Procesadores de lenguaje 1.1. Traductor: Compilador, Ensamblador, Preprocesador. 1.2. Intérprete: Intérpretes puros, Intérpretes avanzados, Intérpretes Incrementales. 2. Estructura de los traductores 3. Fase de análisis: Análisis léxico, Análisis sintáctico, Análisis semántico 4. Tabla de símbolos 5. Manejador de errores 6. Fase de síntesis: Generación de código intermedio, Optimización de código , Generación de código

SESIÓN NO. 3:	**Duración: 2 horas**
OBJETIVO:	**Establecer e identificar las lenguajes formales y su jerarquía**
CONTENIDO:	1. Lenguajes Formales 1.1. Lenguajes recursivamente enumerables 1.2. Lenguajes dependientes del concepto 1.3. Lenguajes independientes del concepto 1.4. Lenguajes regulares 2. Jerarquía de Chomsky 2.1. Tipo 0 2.2. Tipo 1 2.3. Tipo 2 2.4. Tipo 3 <u>Realización del asignación de laboratorio No. 1</u> Utilización de archivos de almacenamiento y manipulación de cadenas y la utilización de listas para el manejo de datos.
SESIÓN NO. 4:	**Duración: 1 horas**
OBJETIVO:	**Determinar todos los componentes que conforman una gramática.**
CONTENIDO:	1. ¿Qué es una gramática? 2. Elementos que componen una gramática 2.1. Definición 2.2. Símbolos terminales 2.3. Símbolos no terminales 2.4. Gramática 2.5. Estado inicial

	3. Construcción de Gramáticas Regulares 3.1. Producciones 3.2. Clasificación 3.3. Árbol de derivación 3.4. Gramáticas ambiguas
SESIÓN NO. 5:	**Duración: 1 horas**
OBJETIVO:	**Practicar la construcción de gramáticas para diversas cadenas, determinando todos sus componentes**
CONTENIDO:	Desarrollo del ejercicio de laboratorio No. 1: Construcción colectiva de definiciones de gramáticas para diversos lenguajes. Desarrollo en casa tutorial No. 1 Construcción de gramáticas
SESIÓN NO. 6:	**Duración: 1 horas**
OBJETIVO:	**Construir expresiones regulares para definir un lenguaje.**
CONTENIDO:	1. Expresiones regulares 1.1. Definición 1.2. Elementos de definición 1.3. Precedencia de operadores 1.4. Propiedades 1.5. Ejemplos 2. Gramáticas y expresiones regulares 2.1. Reglas de construcción 2.2. Relación gramáticas y expresiones regulares 2.3. Equivalencia de expresiones regulares

SESIÓN NO. 7:	**Duración: 1 horas**
OBJETIVO:	**Ejercitar la utilización de expresiones regulares para definir estructuras de información.**
CONTENIDO:	Desarrollo del ejercicio de laboratorio No. 2: Construcción colectiva de expresiones regulares para diferentes casos. Realización del asignación de laboratorio No. 2 Utilización de estructuras para el almacenamiento y manipulación de información, tablas y árboles.
SESIÓN NO. 8:	**Duración: 1 horas**
OBJETIVO:	**Construir DFA y evaluar cadenas de entrada.**
CONTENIDO:	1. Autómatas Finitos 1.1. Definición 1.2. Estados 1.3. Notación (Definición abstracta, diagrama de Moore, tabla transición) 2. Autómatas finitos deterministas. 2.1. Definición. 2.2. Función de transición 2.3. Lenguaje reconocido por un autómata finito. 3. Ejemplos 3.1. ER –DFA 3.2. Tablas transición – DFA 3.3. Definición LR - DFA

SESIÓN NO. 9:	**Duración: 1 horas**
OBJETIVO:	**Ejercitar la construcción de expresiones regulares y DFA a través de JFLAP**
CONTENIDO:	Realización del Ejercicio No. 2 Herramientas para el aprendizaje de autómatas. Modo de uso de JFLAP - DFA Elaboración de autómatas DFA por medio JFLAP Entrega del enunciado del primer proyecto.
SESIÓN NO. 10:	**Duración: 1 horas**
OBJETIVO:	**Construir NFA y evaluar cadenas de entrada**
CONTENIDO:	1. Autómatas finitos no deterministas. 1.1. Definición. 1.2. Estados 1.3. Tabla de transición 2. Ejemplos de construcción 2.1. ER-NFA 2.2. Tabla de transición-NFA 2.3. GR - NFA **3.** Evaluación de cadenas

SESIÓN NO. 11:	**Duración: 1 horas**
OBJETIVO:	**Ejercitar la construcción de NFA y la evaluación de cadenas**
CONTENIDO:	Desarrollo del ejercicio de laboratorio No. 3 Construcción de NFA obteniendo los datos de ER, Tablas de transición o GR Evaluación de cadenas por diversos medios.
SESIÓN NO. 12:	**Duración: 1 horas**
OBJETIVO:	**Definir el proceso del algoritmo de construcción de Thompson a partir de una ER y el cálculo de la cerradura.**
CONTENIDO:	1. Método de construcción de NFA de Thompson. 1.1. Kenneth Lane Thompson. 1.2. Nomenclatura 2. Transformación de NFA – DFA 2.1. Cálculo de la cerradura 2.2. Proceso de transformación
SESIÓN NO. 13:	**Duración: 1 horas**
OBJETIVO:	**Ejercitar la utilización del método de Thompson y de subconjuntos.**
CONTENIDO:	Desarrollo del ejercicio de laboratorio No. 4 Ejercitar la construcción de DFA por el método de Thompson. Realización del asignación de laboratorio No. 3 Utilización de estructuras de almacenamiento como pilas y colas.

SESIÓN NO. 14:	**Duración: 2 horas**
OBJETIVO:	**Definir el método de construcción de un DFA óptimo por medio de los métodos del árbol y de subconjuntos.**
CONTENIDO:	1. Método de árbol 1.1. Definición 1.2. Proceso 1.3. Ejemplo 2. Método de subconjuntos 2.1. Definición. 2.2. Proceso 2.3. Ejemplo
SESIÓN NO. 15:	**Duración: 1 horas**
OBJETIVO:	**Ejercitar la construcción de DFA óptimos por el método del árbol y el método de subconjuntos.**
CONTENIDO:	Desarrollo del ejercicio de laboratorio No. 5 Ejercitar la construcción de DFA por el método de árbol. Elaboración del tutorial No. 2 Ejercitar la construcción de DFA por el método del árbol.

SESIÓN NO. 16:	**Duración: 2 horas**
OBJETIVO:	**Construir gramáticas libres de contexto utilizando la notación BNF**
CONTENIDO:	1. Lenguajes libres de contexto 2. Gramáticas libres de contexto 2.1. Diseño de gramáticas libres de contexto 2.2. Notación BNF 2.3. Derivación y árboles de derivación 2.4. Recursividad por la izquierda y por la derecha
SESIÓN NO. 17:	**Duración: 2 horas**
OBJETIVO:	**Identificar plataformas para el desarrollo del software.**
CONTENIDO:	1. Conceptos principales 1.1. Plataforma, IDE, API, Framework 2. Microsoft .NET 2.1. Net, Arquitectura, Framework. 3. JEE 3.1. JEE (J2EE), Componentes, API´s, IDE´s 4. Comparativa java – vb.net Entrega del enunciado del segundo proyecto

SESIÓN NO. 18:	**Duración: 2 horas**
OBJETIVO:	**Definir que son y para que sirven los Web Services y las tecnologías que colaboran en la construcción de Web Services.**
CONTENIDO:	1. Web Services 1.1. Definición. 1.2. Arquitectura 2. Estándares y protocolos 2.1. XML, SOAP, WSDL, UDDI 3. SOA 3.1. JEE (J2EE), componentes, API´s, IDE´s 4. Comparativa java – vb.net <u>Elaboración del tutorial No. 3</u> Evaluación de la utilización de Web Services en varias plataformas.
SESIÓN NO. 19:	**Duración: 2 horas**
OBJETIVO:	**Utilizar los Web Services como medio de comunicación.**
CONTENIDO:	<u>Desarrollo del ejercicio de laboratorio No. 7</u> Utilización de Web Services <u>Realización del asignación de laboratorio No. 4</u> Utilización de Web Services como medio de comunicación entre dos aplicaciones.

3.6 Detalle de asignación laboratorio

PRÁCTICAS:

Criterio de evaluación:

- Funcionamiento - 75
- Presentación - 05
- Conocimiento técnico - 10
- Documentación - 10

Documentación a entregar: Si

- Enunciado

Tabla VIII. Prácticas y ejercicios del curso Lenguajes Formales

ASIGNACIÓN DE LABORATORIO1:	
OBJETIVO:	Ejercitar la utilización de archivos para el almacenamiento y lectura de información. Y la manipulación de estos a través de la clase java.lang.string y la implementación de listas.
Iniciada en sesión	3
Entregada en sesión	6
Viva / Demostración	NO

ASIGNACIÓN DE LABORATORIO2:	
OBJETIVO:	Optimizar a través de la utilización de jtables y jtree el manejo y manipulación de datos, del Asignación de Laboratorio No. 1.
Iniciada en sesión	7
Entregada en sesión	10
Viva / Demostración	SI
ASIGNACIÓN DE LABORATORIO3:	
OBJETIVO:	Ejercitar la utilización de pilas y colas para procesar información y el funcionamiento de un autómata.
Iniciada en sesión	13
Entregada en sesión	16
Viva / Demostración	NO
ASIGNACIÓN DE LABORATORIO4:	
OBJETIVO:	Utilización de Web Services como medio de Comunicación entre dos aplicaciones.
Iniciada en sesión	19
Entregada en sesión	---
Viva / Demostración	SI

3.7 Detalle de proyectos

PROYECTO:

Criterio de evaluación:

- Funcionamiento - 70
- Conocimiento técnico - 10
- Documentación - 20

Documentación a entregar: SI

- Enunciado

Tabla IX. Proyectos del curso Lenguajes Formales y de Programación

PROYECTO 1:	
OBJETIVO:	Reconocimiento de un lenguaje formal definiendo un DFA y utilizando el método del árbol para el análisis. Poniendo en práctica los conocimientos para procesar documentos de texto, la utilización de jtables, jtree, etc. que le puedan ser de utilidad para el manejo y análisis de los datos.
Iniciada en sesión	9
Entregada en sesión	16
Viva / Demostración	NO
PROYECTO 2:	
OBJETIVO:	Construcción de un intérprete para un lenguaje formal, debiendo utilizar los conceptos aprendidos en el curso con

	la utilización de Web Services para un proceso de comunicación.
Iniciada en sesión	17
Entregada en sesión	-----
Viva / Demostración	NO

3.8 Detalle de exámenes

EXAMEN:

Número total de exámenes 1

Examen realizado después de sesión 18

Temas para el examen Lenguajes Formales
Definición de gramáticas
NFA y DFA
Método del árbol

Duración 1 ½ hrs.

Criterio de evaluación:

Objetivo - 40%

Subjetivo - 60%

3.9 Examen de ejemplo

Figura 1. Ejemplo de examen del curso Lenguajes Formales

Serie I (40 pts): Resuelva los siguientes casos

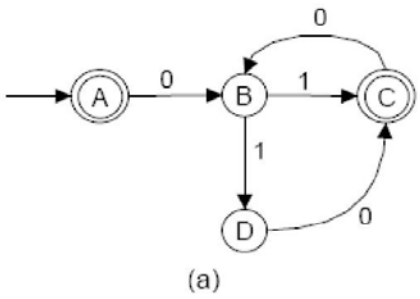
1. Dé 3 ejemplos de cadenas pertenecientes a los conjuntos definidos por cada una de las siguientes expresiones regulares:
   a. (0|1)* 00 (0|1)*
   b. (a*|ba|b)+
   c. (0 (0|1)+ 1 | 1(1|0)* 0)

2. Escriba una expresión regular que reconozca cadenas con al menos un par de ceros consecutivos y al menos un par de unos consecutivos.

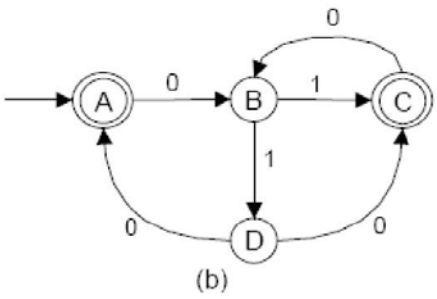
3. Construya una gramática regular no ambigua que genere todas las cadenas de 0 y 1 en las cuales los 0, si aparecen, lo hacen en grupos de tres.

Ej. de hileras que pertenecen al lenguaje	Ej. de hileras que NO pertenecen al lenguaje
1 (no tiene 0)	0 (el 0 no está en grupo de 3)
11111 (no tiene 0)	00 (el 0 no está en grupo de 3)
000 (tiene 0 y aparece en grupo de 3)	11000011 (el 0 está en grupo de 4)
1000 (tiene 0 y aparece en grupo de 3)	000000 (el 0 está en grupo de 6)
1110001000 (los 0 aparecen en grupo de 3)	000100 (el segundo grupo de 0 está en grupo de 2)

4. Los siguientes autómatas, ¿son equivalentes (reconocen el mismo lenguaje)?



(a)



(b)

5. Escriba una gramática que reconozca oraciones sencillas, que utilicen sujeto y predicado.

Serie II (60 pts): Desarrollo los siguientes temas.

- Escriba que es un web Service y cuales son sus protocolos y estándares que utiliza.
- Describa las diferentes clasificaciones de los lenguajes de programación
- Explique el funcionamiento estándar de un compilador (Dibuje el esquema)
- Liste los tipos de lenguajes que estableció Chomsky
- Defina los elementos que componen una Gramática

3.10 Bibliografía sugerida

1. **Aho, Alfred V., Sethi, Ravi y Ullman, Jeffrey D.** Compiladores, pricipios, técnicas y herramientas. 1990.
2. **Gálvez Rojas, Sergio y Mora Mata, Miguel Ángel.** Traductores y Compiladores con LEX/YACC, JFLEX/CUP Y JAVACC. 2005.
3. **Louden, Kenneth C.** Construcción de compiladores/ Construction of compilers: Principios y práctica. 2004.
4. **Rodríguez Avila, Eduardo René.** Diseño de Compiladores. 2000.
5. **Taylor, Gregory.** Models of Computation and Formal Languages . 2003.

3.11 Anexos

A. Ejercicios de laboratorio

Ejercicio de laboratorio No. 1

Objetivo:

Practicar la construcción de gramáticas para diversos lenguajes, determinando todos sus componentes.

Especificaciones:

- Construir gramáticas que reconozcan cadenas específicas.
- Construir el árbol de derivación asociado
- Evaluar si la gramática es ambigua o no.
- Comparar entre las gramáticas generadas por otros grupos y discutir.

Ejercicio de laboratorio No. 2

Objetivo:

Ejercitar la utilización de expresiones regulares para definir estructuras de información.

Especificaciones:

- Construir expresiones regulares que reconozcan cadenas específicas.
- Comparar expresiones regulares equivalentes, y no equivalentes
- Evaluar si cada expresión regular corresponde a los requerimientos de la misma.

Ejercicio de laboratorio No. 3

Objetivo:

Ejercitar la construcción de expresiones regulares y DFA a través de JFLAG

Especificaciones:

- Construir a través de JFLAG DFA y ER, para afianzar los conocimientos.
- Diseñar DFA que acepten cadenas establecidas, verificando su funcionamiento paso por paso.
- Se puede practicar con los ejercicios realizados en el laboratorio No. 1.

Ejercicio de laboratorio No. 4

Objetivo

Ejercitar la construcción autómatas finitos determinísticos a partir de diferentes definiciones.

Especificaciones:

- Construcción de NFA obteniendo los datos de expresiones regulares, tablas de transición o gramáticas regulares.
- Evaluación de cadenas luego de la construcción del NFA, recorriendo el diagrama y verificando los estados.

Ejercicio de laboratorio No. 5

Objetivo:

Ejercitar la construcción de NFA a través del método de Thompson y el cálculo de la cerradura

Especificaciones:

- Ejercitar la construcción de NFA partiendo de ER, haciéndolo por medio del método de Thompson y el cálculo de la cerradura.
- Puede utilizarse cualquier tipo de expresión regular, o puede proponerse a los estudiantes que creen una propia definición de expresión de regular.

Ejercicio de laboratorio No. 6

Objetivo:

Ejercitar el uso del método del árbol para la construcción de DFA óptimos.

Especificaciones:

- Ejercitar la construcción de DFA partiendo de ER, haciéndolo por medio del método del árbol y el método de subconjuntos.
- Realizar uno o dos ejercicios en clase para afianzar los conocimientos.

Ejercicio de laboratorio No. 7

Objetivo:

Practicar la comunicación entre aplicaciones por medio de Web Services.

Especificaciones:

- Realizar en grupos ejemplos de la construcción de Web Services. Es importante dar a conocer los métodos y funciones necesarias para la construcción y comunicación.
- Implementar Web Services en diferentes lenguajes de programación, y sistema operativo, para reforzar el conocimiento.

B. Asignación de laboratorio

Asignación de laboratorio No. 1

Objetivo:

Ejercitar la utilización de archivos para el almacenamiento y lectura de información. Y la manipulación de estos a través de la clase java.lang.string y la implementación de listas.

Especificaciones:

- Creación de un entorno gráfico adecuado para un editor. Como mínimo deberá incluir:
    - Menú archivo: abrir, guardar, guardar como, cerrar
    - Menú lectura: carga de archivo, mostrar bitácora, ,mostrar errores
- Creación de un editor de texto, que permita la manipulación adecuada de archivos.
- Implementar un análisis léxico en el lenguaje Java, que reconozca errores léxicos.
- Deberá proveer una bitácora en la que se lleve el registro de todas las operaciones realizadas.

- Implementación de listas en java (No se puede utilizar la que ya provee el lenguaje)
- Manipulación adecuada de la información almacenada en las listas.
- Utilizar los métodos de la clase String.
- Deberá ser realizado en forma individual para un mejor aprendizaje.

Asignación de laboratorio No. 2

Objetivo:

Optimizar a través de la utilización de jtables y jtree el manejo y manipulación de datos, de la asignación de laboratorio No. 1.

Especificaciones:

- Utilice el modo gráfico implementado en la asignación de laboratorio 1.
- Siguiendo con la relación del sistema, optimice el manejo de información por medio de la utilización de las clases de jtables, y jtree.
- Deberán crearse elementos que permitan la búsqueda, de acuerdo a diversos parámetros. Tomando en cuenta que se deben de utilizar y demostrar la utilización de jtables y de jtree para la implementación.
    - Buscar_lista1 (código);
    - Buscar_lista1 (nombre);
- Deberá proveer un diagrama o gráfica que ejemplifique el proceso de búsqueda con ambos elementos.

Asignación de laboratorio No. 3

Objetivo:

Ejercitar la utilización de pilas y colas para procesar información y el funcionamiento de un autómata.

Especificaciones:

- Realice e implementa el funcionamiento estándar de un pila y una cola para el manejo de la información.
- Implemente las funciones asociadas a cada estructura. (Pila: push, pop. Cola: almacenar, extraer)
- Utilice estas estructuras para el análisis de cadenas de entradas y evaluación dentro de autómatas.

Asignación de laboratorio No. 4

Objetivo:

Utilización de Web Services como medio de Comunicación entre dos aplicaciones.

Especificaciones:

- Elabore un Web Service para la comunicación entre dos aplicaciones basado en el asignación de laboratorio No. 3
- La comunicación debe darse entre dos Web Services elaborados en lenguajes diferentes, y puede ser la comunicación entre diferentes sistemas operativos.
- Dependiendo de la aplicación o juego, puede realizarse en parejas o grupos, partiendo a que cada máquina se ve como un solo jugador.
- Se debe realizar un diagrama del proceso de comunicación, análisis y evaluación por el autómata.

C. Tutoriales

Tutorial No. 1

Objetivo:

Practicar la construcción de gramáticas para diversos lenguajes, determinando todos sus componentes

Especificaciones:

Resolver al menos dos siguientes problemas:

Problema No. 1

Dada la gramática G ({S}, {a,b}, { S → aS | aSbS | ε }, {S})

a. Demostrar que es ambigua especificando dos posibles árboles de derivación para la cadena aab.

b. Encontrar otra gramática que genere L(G) y no sea ambigua

Problema No. 2

Construir una gramática con el conjunto de todas las palabras sobre {a, b} tales que toda cadena de cinco símbolos contiene como mínimo dos a's.

Problema No. 3

Palabras sobre {0, 1} en que cada símbolo que ocupa una posición múltiplo de tres es un 1

Problema No. 4

Dada la gramática G ({S}, {a,b}, { S → aS | Sb | a | b }, {S})

Demostrar que no existe ninguna cadena que posea una subcadena ba.

Problema No. 5

Construya una gramática que reconozca palabras con prefijo < y sufijo /> y donde la subcadena interna se forma a través de secuencia de letras y no puede estar vacía.

Tutorial No. 2

Objetivo:

Ejercitar la construcción de DFA óptimos por el método del árbol y el método de subconjuntos.

Especificaciones:

- A partir de las siguientes expresiones regulares, construir el árbol, hacer los cálculos asociados y a través del método de subconjuntos construir un DFA óptimo.
    - b*|ba*
    - ((10)*1)*
    - (a(b|aa*))*a
    - z*|x*(ε|z)
    - 0|((00)*1)*
    - f|j(f|(ja*)*)
- Puede utilizarse Graphviz para la creación de los autómatas finitos.

Tutorial No. 3

Objetivo:

Evaluar fortalezas y debilidades de la utilización de DotNet o JE22 para la creación de Web Services.

Especificaciones:

- Realizar un cuadro comparativo de estas dos plataformas, en cuanto a la utilización y creación de Web Services.
- Además de investigar la compatibilidad entre ellos, al momento de realizar un Web Service.

D. Proyecto de Laboratorio

Proyecto No. 1

Objetivo:

Reconocimiento de un lenguaje formal definiendo un DFA y utilizando el método de Thompson para el análisis. Poniendo en práctica los conocimientos para procesar documentos de texto, la utilización de jtables, jtree, etc. que le puedan ser de utilidad para el manejo y análisis de los datos.

Especificaciones:

- Se utilizará la arquitectura cliente/servidor
- Se debe desarrollar un módulo en el servidor que emule una base de datos donde se almacenaran todos los procesos ejecutados.
- Se debe desarrollar un módulo en servidor que distribuya los procesos y pueda verlos como si fueran propios del sistema operativo.
- Se debe desarrollar una aplicación cliente que permita la creación de los procesos, a través de una interfaz gráfica.
- Se debe implementar seguridad a la aplicación por medio de un módulo de usuarios, con diferentes roles y manejo de sesiones.
- Se debe de implementar una bitácora que guarde la información de cada acción realizada.
- Se debe hacer un instalador interactivo, para la aplicación cliente y para la aplicación del servidor.
- Realizar documentación técnica, de usuario y manual de instalación.
- Trabajarlo sobre sistema operativo Linux.
- Implementarlo en Java o QT. (a criterio del encargado)

Proyecto No. 2

Objetivo:

Construcción de un intérprete para un lenguaje formal, debiendo utilizar los conceptos aprendidos en el curso con la utilización de Web Services para un proceso de comunicación.

Especificaciones:

- Se debe desarrollar aplicaciones similares en diferentes lenguajes de programación. Utilizando para el análisis de los datos de entrada un autómata.
- Se debe desarrollar una aplicación cliente que permita la creación de los procesos, a través de una interfaz gráfica.
- Se debe implementar seguridad a la aplicación por medio de un módulo de usuarios, con diferentes roles y manejo de sesiones.
- Se debe de implementar una bitácora que guarde la información de cada acción realizada.
- Realizar documentación técnica, de usuario y manual de instalación.
- Trabajarlo sobre varios sistemas operativos y lenguajes diferentes.
- Puede utilizarse como base un juego de estrategia o de mesa, definiendo un lenguaje para la manipulación o movimiento del juego. Haciendo la comunicación por medio de un Web Services.

4. GUÍA DEL INSTRUCTOR DEL CURSO ORGANIZACIÓN DE LENGUAJES Y COMPILADORES 1

A continuación se presenta el desarrollo de la Guía del curso Organización de Lenguajes y Compiladores 1, de acuerdo al formato sugerido en el curso de "Estructuración de laboratorios", impartido por el señor. Mrutunjaya Panda.

4.1 Datos Generales

NOMBRE DE CURSO: Organización de Lenguajes y Compiladores 1 (777)

PRE- REQUISITOS: Introducción a la Programación 2 (771)
Lenguajes Formales y de Programación (796)
Matemática de Cómputo 2 (962)

POST – REQUISITOS: Organización de Lenguajes y Compiladores 2 (781)

OBJETIVO: Al finalizar el curso, el estudiante podrá iniciar la construcción de un compilador a través de herramientas específicas para realizar el análisis léxico y sintáctico.

4.2 Distribución del curso

Tabla X. Distribución Organización de Lenguajes y Compiladores 1

MÓDULO	TEORÍA	PRÁCTICA	TOTAL	EXAMEN	ASIGNACIÓN DE LABORATORIOS	PROYECTOS
LENGUAJES Y COMPILADORES INTERMEDIO	16 horas	8 horas	24 horas	1 ½ horas	4	2
			24 horas			

4.3 Evaluación

Ponderación de tareas, prácticas y proyectos.

Tabla XI. Evaluación del curso Compiladores 1

ACTIVIDAD	NÚMERO	PORCENTAJE	TOTAL
CASO DE ESTUDIO	1	4.0 %	4.00 %
TUTORIAL	3	1.0 %	3.00%
ASIGNACIÓN DE LABORATORIO	4	2.0 %	8.00 %
PROYECTO	2	25.0 % 45.0 %	70.00 %
EXAMEN	1	15.0 %	15.00 %
		TOTAL	**100.00 %**

4.4 Detalle de sesiones

Tabla XII. Sesiones del curso Compiladores 1

No	TEORIA	No	PRACTICA	TUTORIAL / PROYECTO / ASIGNACION LABORATORIO
1	Lenguajes y Compiladores Intermedio- 1	2	Laboratorio – 1	Caso de estudio – 1
3	Lenguajes y Compiladores Intermedio- 2	4	Laboratorio – 2	Asignación laboratorio – 1
5	Lenguajes y Compiladores Intermedio- 3	6	Laboratorio – 3	
7	Lenguajes y Compiladores Intermedio- 4			Proyecto – 1
8	Lenguajes y Compiladores Intermedio– 5	9	Laboratorio – 4	Tutorial - 1
10	Lenguajes y Compiladores Intermedio– 6			
11	Lenguajes y Compiladores Intermedio– 7			Asignación Laboratorio – 2
12	Lenguajes y	13	Laboratorio – 5	Asignación Laboratorio – 3

	Compiladores Intermedio– 8			Proyecto – 2
14	Lenguajes y Compiladores Intermedio– 9	15	Laboratorio – 6	Tutorial – 2
16	Lenguajes y Compiladores Intermedio– 10	17	Laboratorio – 7	Tutorial – 3
18	Lenguajes y Compiladores Intermedio– 11	19	Laboratorio – 8	Asignación laboratorio – 4
20	Lenguajes y Compiladores Intermedio– 12			
--	Evaluación Final			

4.5 Distribución de sesiones

Tabla XIII. Sesiones curso Compiladores 1

SESIÓN NO. 1:	**Duración: 1 hora 30 minutos**
OBJETIVO:	**Reafirmar los conocimientos referentes al proceso de procesadores del lenguaje.**
CONTENIDO:	1. Procesadores de lenguaje 1.1. Traductor: compilador, ensamblador, preprocesador. 1.2. Intérprete: intérpretes puros, intérpretes avanzados, intérpretes incrementales.

	2. Estructura de los traductores 3. Fase de análisis: análisis léxico, análisis sintáctico, análisis semántico 4. Tabla de símbolos 5. Manejador de errores 6. Fase de síntesis: generación de código intermedio, optimización de código , generación de código
SESIÓN NO. 2:	**Duración: 30 minutos**
OBJETIVO:	**Recopilar información relacionada con el concepto de GNU / Linux y sus diversas distribuciones.**
CONTENIDO:	1. GNU / Linux 1.1. Proyecto GNU 1.2. Kernel Linux 2. Distribuciones GNU / Linux 3. Gestores de ventana <u>Realización del caso de estudio No. 1</u> Documentación del proceso de instalación de una distribución GNU/Linux.

SESIÓN NO. 3:	**Duración: 1 hora**
OBJETIVO:	**Definir el proceso de análisis léxico y la utilización de expresiones regulares para dicho análisis.**
CONTENIDO:	1. Análisis de léxico 1.1. Función del analizador de léxico 1.2. Especificación de los tokens 1.3. Cadenas y lenguajes 1.4. Operaciones y aplicaciones a lenguajes 2. Expresiones regulares 2.1. Operadores 2.2. Definiciones regulares

SESIÓN NO. 4:	**Duración: 1 hora**
OBJETIVO:	**Establecer e identificar las diferentes herramientas para la fase de análisis**
CONTENIDO:	1. Herramientas para la fase de análisis 1.1. Lex y Yacc 1.2. JLex y Cup 1.3. Flex y Bison 2. Proyectos 2.1. Anagra 2.2. Sefasgen 2.3. JavaPars 2.4. SEPa! 3. Obtención de las herramientas JLex y Cup Realización del asignación de laboratorio No. 1 Instalación de las herramientas JLex y Cup en ambiente GNU / Linux

SESIÓN NO. 5:	**Duración: 90 minutos**
OBJETIVO:	**Definir el proceso de construcción y análisis a partir de un autómata finito**
CONTENIDO:	4. Diagramas de transición 4.1. Implantación de diagramas de transición o Moore 5. Autómatas finitos 5.1. Autómatas finitos no determinados (AFN) 5.2. Autómatas finitos determinísticos (AFD)
SESIÓN NO. 6:	**Duración: 30 minutos**
OBJETIVO:	**Determinar todos los componentes que conforman un análisis léxico, en la herramienta JLex.**
CONTENIDO:	1. Definir que es un analizador léxico 1.1. Estructura del archivo de JLex 1.1.1. Declaración de paquete 1.1.2. Declaración de símbolos 1.1.3. Expresiones 1.1.4. Funciones de reconocimiento 1.1.5. Sintaxis general 1.2. Instrucciones más usadas por los archivos JLex 1.2.1. yytext( ) 1.2.2. Symbol( ) 1.2.3. System.out.print() 1.2.4. System.err.println() 1.3. Como compilar en JLex 1.4. Ejemplos de archivo JLex

SESIÓN NO. 7:	**Duración: 2 horas**
OBJETIVO:	**Establecer varios métodos para la construcción de autómatas finitos.**
CONTENIDO:	1. Construcción de Thompson - convertir una expresión regular a un AFN 2. Construcción de subconjuntos - convertir un AFN a un AFD 3. Construcción de un AFD por el método del árbol
SESIÓN NO. 8:	**Duración: 1 horas 30 minutos**
OBJETIVO:	**Determinar el proceso de análisis de sintaxis por medio de gramáticas.**
CONTENIDO:	1. Análisis de sintaxis 1.1. El papel del analizador sintáctico 1.2. Gramáticas independientes del contexto 1.3. Definiciones y notación 1.4. Derivación 1.5. Árboles de análisis sintácticos y derivaciones 1.6. Ambigüedad 2. Escritura de una gramática 3. Comprobación del lenguaje generado por una gramática <u>Realización del tutorial No. 1</u> Ejercitar la construcción de gramáticas independientes del contexto.

SESIÓN NO. 9:	**Duración: 30 minutos**
OBJETIVO:	**Determinar todos los componentes que conforman un análisis sintáctico, en la herramienta CUP.**
CONTENIDO:	1. Analizadores sintácticos con CUP 1.1. Definir que es un analizador sintáctico 1.2. Estructura del archivo de Cup 1.2.1. Declaración de paquete 1.2.2. Definición de código (java) 1.2.3. Definición de código parser 1.2.4. Declaración de terminales y no terminales 1.2.5. Declaración de precedencia 1.2.6. Definición de gramática 1.3. Instrucciones más usadas por los archivos Cup 1.4. Cómo compilar en CUP 1.5. Ejemplos de archivo CUP
SESIÓN NO. 10:	**Duración: 1 horas 30 minutos**
OBJETIVO:	**Practicar la reescritura de una gramática, mejorando la definición.**
CONTENIDO:	1. Construcción y uso del árbol de derivación 2. Escritura de gramáticas 2.1. Recursividad 2.1.1. Recursividad por la izquierda 2.1.2. Recursividad por la derecha 2.1.3. Eliminación de recursividad directa 2.1.4. Eliminación de recursividad indirecta

	2.2. Ambigüedad 2.2.1. Tipos de ambigüedad 2.2.2. Causas de ambigüedad 2.2.3. Eliminación de ambigüedad 2.3. Factorización 2.4. Eliminación de símbolos inalcanzables 2.5. Eliminación de producciones no generativas 2.6. Eliminación de producciones épsilon
SESIÓN NO. 11:	**Duración: 1 hora 30 minutos**
OBJETIVO:	**Definir los métodos de analizadores para reconocer una cadena.**
CONTENIDO:	1. Métodos de analizadores sintácticos 1.1. Método descendentes 1.2. Métodos ascendentes 2. Tipos de atributos 2.1. Atributos sintetizados 2.2. Atributos heredados Realización del asignación de laboratorio No. 2 Ejercitar la construcción de analizadores con JLex y CUP e integrar dichas herramientas
SESIÓN NO. 12:	**Duración: 90 minutos**
OBJETIVO:	**Establecer los tipos de análisis sintáctico ascendente y sus métodos.**

CONTENIDO:	1. Análisis sintáctico descendente 1.1. Análisis sintáctico descendente con retroceso 1.2. Análisis sintáctico descendente predictivo 1.3. Condiciones para el reconocimiento 1.3.1. Recursivo por la izquierda 1.3.2. Factorizado por la izquierda 1.3.3. Sin ambigüedad 2. LL(1) 2.1. Construcción de tablas LL(1). 2.2. Algoritmo para el cálculo de first & follow 2.3. Comprobación de entradas por medio de pila
SESIÓN NO. 13:	**Duración: 30 minutos**
OBJETIVO:	**Ejercitar el análisis sintáctico descendente por medio de la utilización de LL(1)**
CONTENIDO:	Realización del asignación de laboratorio No. 3 Practicar la construcción de analizadores sintácticos tipo LL(1)
SESIÓN NO. 14:	**Duración: 90 minutos**
OBJETIVO:	**Establecer los tipos de análisis sintáctico ascendente y sus métodos.**
CONTENIDO:	1. SLR (1) 1.1. Algoritmo SLR(1). 1.2. Definición de gramática SLR(1). 1.3. Propiedades de la gramática SLR 1.4. Tabla de estados y grafos 1.5. Tratamiento de errores

SESIÓN NO. 15:	**Duración: 30 minutos**
OBJETIVO:	**Ejercitar el análisis sintáctico ascendente por medio de la utilización de SLR**
CONTENIDO:	Realización del tutorial No. 2 Practicar la construcción de tablas SLR y su utilización para el reconocimiento mediante la pila
SESIÓN NO. 16:	**Duración: 90 minutos**
OBJETIVO:	**Establecer los tipos de análisis sintáctico ascendente tipo LR.**
CONTENIDO:	1. LR (0) 1.1. Algoritmo LR (0) 1.2. Definición de gramática LR (0) 1.3. Propiedades de la gramática LR (0) 1.4. Tabla de estados y grafos 1.5. Tratamiento de errores 1.6. Ejemplos 2. LR - canónico 2.1. Algoritmo LR - canónico 2.2. Definición de gramática LR - canónico 2.3. Propiedades de la gramática LR - canónico 2.4. Tabla de estados y grafos 2.5. Tratamiento de errores 2.6. Ejemplos

SESIÓN NO. 17:	**Duración: 30 minutos**
OBJETIVO:	**Ejercitar el análisis sintáctico ascendente por medio de la utilización de LR canónico.**
CONTENIDO:	Realización del tutorial No. 3 Practicar la construcción de tablas LR – canónico y su utilización para el reconocimiento mediante la pila
SESIÓN NO. 18:	**Duración: 1 hora 30 minutos**
OBJETIVO:	**Establecer los tipos de análisis sintáctico ascendente LALR.**
CONTENIDO:	1. LALR 1.1. Construcción de tablas LALR 1.2. Definición de gramática LALR 1.3. Propiedades de la gramática LALR 1.4. Cálculo de los núcleos de la colección de subconjuntos de elementos LALR (1) 1.5. Tratamiento de errores 1.6. Ejemplos 1.7. Reconocimiento a través de la tabla de estados y pila
SESIÓN NO. 19:	**Duración: 30 minutos**
OBJETIVO:	**Ejercitar el análisis sintáctico ascendente por medio de la utilización de LALR**
CONTENIDO:	Realización del asignación de laboratorio No. 4 Practicar la construcción de analizadores sintácticos tipo LALR

SESIÓN NO. 20:	Duración: 2 horas
OBJETIVO:	**Reafirmar los conocimientos sobre el análisis sintáctico.**
CONTENIDO:	1. Análisis sintáctico 1.1. Análisis SLR 1.2. Análisis LR 1 1.3. Análisis LL 1 1.4. Análisis LALR 2. Reescritura de gramáticas 2.1. Eliminación de recursividad (izquierda – derecha) 2.2. Eliminación de ambigüedad 2.3. Factorización

4.6 Detalle de asignación laboratorio

Criterio de evaluación:

- Funcionamiento - 75
- Presentación - 05
- Conocimiento técnico - 10
- Documentación - 10

Documentación a entregar: SI

Enunciado

Tabla XIV. Prácticas curso Organización de Lenguajes y Compiladores 1

ASIGNACIÓN LABORATORIO 1:	Instalación de JLex y Cup
OBJETIVO:	Instalar dentro del sistema operativo Linux, una herramienta para la fase de análisis de un compilador
Iniciada en sesión	4
Entregada en sesión	6
Viva / Demostración	SI
ASIGNACIÓN LABORATORIO 2:	Implementación de un analizador sencillo
OBJETIVO:	Ejercitar la construcción de analizadores con JLex y CUP e integrar dichas herramientas
Iniciada en sesión	11
Entregada en sesión	12
Viva / Demostración	SI
ASIGNACIÓN LABORATORIO 3:	Análisis sintáctico descendente.
OBJETIVO:	Ejercitar el análisis sintáctico descendente por medio de la utilización de LL(1)
Iniciada en sesión	12
Entregada en sesión	14
Viva / Demostración	SI

ASIGNACIÓN LABORATORIO 4:	Análisis sintáctico ascendente.
OBJETIVO:	Ejercitar el análisis sintáctico ascendente. (LALR)
Iniciada en sesión	18
Entregada en sesión	20
Viva / Demostración	SI

4.7 Detalle de proyectos

PROYECTO 1:

Criterio de Evaluación:

- Funcionamiento - 70
- Conocimiento técnico - 10
- Documentación - 20

Documentación a entregar: SI

- Enunciado detallado en el anexo

Tabla XV. Proyectos curso Organización de Lenguajes y Compiladores 1

PROYECTO 1:	Poner en práctica los conocimientos del curso, sobre análisis léxico y sintáctico.
OBJETIVO:	Que el estudiante ponga en práctica los conocimientos adquiridos sobre las fases de compilación de análisis léxico y análisis sintáctico, por medio de herramientas que le sean de ayuda para la construcción de estos análisis.
Iniciada en sesión	9
Entregada en sesión	14
Viva / Demostración	NO
PROYECTO 2:	Poner en práctica los conocimientos adquiridos los métodos ascendentes y descendentes en la construcción de un intérprete para una función específico.
OBJETIVO:	Que el estudiante aplique los conocimientos adquiridos sobre el proceso de compilación e interpretación en el desarrollo de un sistema computacional.
Iniciada en sesión	9
Entregada en sesión	14
Viva / Demostración	NO

4.8 Detalle de exámenes

Número total de exámenes	1
Examen realizado después de sesión	18
Temas para el examen	Lenguajes y Compiladores Intermedio
Duración	1 ½ hrs.

Criterio de evaluación:

- Objetivo - 40%
- Subjetivo - 60%

4.9 Examen de ejemplo

Figura 2. Ejemplo de examen del curso Compiladores 1

Serie I (40 pts): Resuelva los siguientes casos

1. Dada la siguiente gramática
   E → E + T | T
   T → T * F | F
   F → (E) | **id**
   (a) Elimine la recursión por la izquierda
   (b) Elimine la ambigüedad
   (c) Factorice la solución

2. Considere la gramática
   S → (L) | a
   L → L , S | S
   (a) Cuales son terminales y no terminales
   (b) Construya el árbol sintáctico para (a, (a, a))
   (c) Realice una derivación por la izquierda
   (d) Realice una derivación por la derecha

3. Construya una gramática ascendente que genere todas las cadenas de 0 y 1 en las cuales los 0, si aparecen, lo hacen en grupos de tres.

Ej. de hileras que pertenecen al lenguaje	Ej. de hileras que NO pertenecen al lenguaje
1 (no tiene 0)	0 (el 0 no está en grupo de 3)
11111 (no tiene 0)	00 (el 0 no está en grupo de 3)
000 (tiene 0 y aparece en grupo de 3)	11000011 (el 0 está en grupo de 4)
1000 (tiene 0 y aparece en grupo de 3)	000000 (el 0 está en grupo de 6)
1110001000 (los 0 aparecen en grupo de 3)	000100 (el segundo grupo de 0 está en grupo de 2)

4. Construya una gramática ascendente que reconozca operaciones matemáticas entre enteros.

Serie II (60 pts): Desarrollo los siguientes temas.

- Explique el funcionamiento estándar de un compilador (Dibuje el esquema)
- Explique la diferencia entre analizadores ascendentes y descendentes.
- Explique el procedimiento para crear una tabla de LR 1

4.10 Bibliografía sugerida

1. **Aho, Alfred V., Sethi, Ravi y Ullman, Jeffrey D.** Compiladores, pricipios, técnicas y herramientas. 1990.
2. **Gálvez Rojas, Sergio y Mora Mata, Miguel Ángel.** Traductores y Compiladores con LEX/YACC, JFLEX/CUP Y JAVACC. 2005.
3. **Rodríguez Avila, Eduardo René.** Diseño de Compiladores. 2000.

4.11 Anexos

A. Asignaciones de laboratorio

Asignación de laboratorio No. 1

Objetivo:

Instalar las herramientas JLex y Cup en un ambiente GNU / Linux.

Especificaciones:

- Instalación de un distribución GNU / Linux.
- Instalar JDK Java 2 SE Development kit
- Descargar los componentes de JLex y Cup
- Documentar el proceso de instalación de las herramientas.
- Deberá entregar un resumen del proceso realizado
- Deberá ser realizado en forma individual para un mejor aprendizaje.

Asignación de laboratorio No.2

Objetivo:

Ejercitar el uso del analizador JLex y CUP.

Especificaciones:

- Construcción de un analizador léxico y sintáctico.
- Evaluar un archivo de entrada por medio de dicho analizador.
- Generar una salida en donde se identifique el análisis.
- Proporcionar información detallada de los errores que se encuentren.

Asignación de laboratorio No. 3

Objetivo:

Ejercitar la construcción del analizador descendente LL(1)

Especificaciones:

- Realice e implementa el funcionamiento estándar del método LL(1)
- Establecer dos ejercicios sencillos proporcionando la gramática y luego que el estudiante cree su propia gramática para analizar.
- Realizado de forma individual.

Asignación de laboratorio No. 4

Objetivo:

Ejercitar la construcción del método LALR.

Especificaciones:

- Establecer dos ejercicios sencillos proporcionando la gramática y luego que el estudiante cree su propia gramática para analizar.
- Establecer algunas cadenas de entrada para verificar el analizador LALR.
- Realizado de forma individual.
- Debe mandar screenshots del procedimiento ejecutándose y el script del ejercicio completo.

B. Sección de proyectos

Proyecto No. 1

Objetivo:

Que el estudiante ponga en práctica los conocimientos adquiridos sobre las fases de compilación de análisis léxico y análisis sintáctico, por medio de herramientas que le sean de ayuda para la construcción de estos análisis. Poniendo en práctica los conocimientos para procesar documentos de texto, la utilización de Jlex y Cup.

Especificaciones:

- Se utilizará la arquitectura capas.
- Definiendo una capa de presentación que permita al usuario el ingreso, modificación y eliminación de datos. Se debe tomar en cuenta una interfaz gráfica que sea amigable para el usuario. Es importante también proporcionar ayuda al usuario para la construcción de las entradas.
- Definiendo una capa de negocio con todos los proceso para la manipulación de la información, llevando la lógica del programa.
- Y la capa de datos almacenando y llevando un control minucioso del análisis desarrollado en la capa de negocio (análisis léxico, sintáctico).
- Se debe hacer un instalador interactivo, para la aplicación
- Realizar documentación técnica, de usuario y manual de instalación.
- Trabajarlo sobre sistema operativo Linux, cualquier distribución.
- Implementarlo en Java o QT. (*a criterio del encargado*)

Proyecto No. 2

Objetivo:

Construcción de un intérprete para un lenguaje formal, debiendo utilizar los conceptos aprendidos en el curso con la utilización de JLex yCup.

Especificaciones:

- Se utilizará la arquitectura capas.
- Definiendo una capa de presentación que permita al usuario el ingreso, modificación y eliminación de datos. Se debe tomar en cuenta una interfaz gráfica que sea amigable para el usuario. Es importante también proporcionar ayuda al usuario para la construcción de las entradas.
- Definiendo una capa de negocio con todos los proceso para la manipulación de la información, llevando la lógica del programa.
- Y la capa de datos almacenando y llevando un control minucioso del análisis desarrollado en la capa de negocio (análisis léxico, sintáctico).
- Se debe hacer un instalador interactivo, para la aplicación
- Realizar documentación técnica, de usuario y manual de instalación.
- Trabajarlo sobre sistema operativo Linux, cualquier distribución.
- Implementarlo en Java o QT. (*a criterio del encargado*)

5. GUÍA DE INSTRUCTOR DEL CURSO ORGANIZACIÓN DE LENGUAJES Y COMPILADORES 2

A continuación se presenta el desarrollo de la Guía del curso de Organización de Lenguajes y Compiladores 2, de acuerdo al formato sugerido en el curso de "Estructuración de laboratorios", impartido por el señor. Mrutunjaya Panda.

5.1 Datos Generales

NOMBRE DE CURSO:	Organización de Lenguajes y Compiladores 2 (781)
PRE- REQUISITOS:	Organización de Lenguajes y Compiladores 1 (777) Estructuras de Datos (772)
POST – REQUISITOS:	Sistemas Operativos 1 (281)
OBJETIVO:	Al finalizar el curso, el estudiante tendrá una base teórica y práctica que permita diseñar e implementar de una manera completa un compilador o intérprete para un lenguaje de las nuevas generaciones, haciendo énfasis en la fase de síntesis.

5.2 Distribución del curso

Tabla XVI. Distribución Organización de Lenguajes y Compiladores 2

MÓDULO	TEORÍA	PRÁCTICA	TOTAL	EXAMEN	ASIGNACIÓN DE LABORATORIO	PROYECTOS
LENGUAJES Y COMPILADORES PRINCIPIANTES	19 horas	6 horas	25 horas	1 ½ horas	6	2
			25 horas			

5.3 Evaluación

Ponderación de tareas, prácticas y proyectos.

Tabla XVII. Evaluación Organización de Lenguajes y Compiladores 2

ACTIVIDAD	NÚMERO	PORCENTAJE	TOTAL
CASO DE ESTUDIO	1	4.0 %	4.00 %
ASIGNACIÓN DE LABORATORIO	6	1.0 %	6.00%
PROYECTO	2	30.0 % 55.0 %	85.00 %
EXAMEN	1	5.0 %	5.00 %
		TOTAL	**100.00 %**

5.4 Detalle de sesiones

Tabla XVIII. Sesiones curso Organización de Lenguajes y Compiladores 2

No	TEORIA	No	PRÁCTICA	TUTORIAL / PROYECTO / ASIGNACIÓN DE LABORATORIO
1	Lenguajes y Compiladores Avanzado- 1	2	Laboratorio – 1	
3	Lenguajes y Compiladores Avanzado- 2	4	Laboratorio – 2	Caso de estudio – 1
5	Lenguajes y Compiladores Avanzado- 3	6	Laboratorio – 3	Asignación de laboratorio – 1
7	Lenguajes y Compiladores Avanzado– 4	8	Laboratorio – 4	Proyecto – 1
9	Lenguajes y Compiladores Avanzado– 5	10	Laboratorio – 5	Asignación de laboratorio – 2
11	Lenguajes y Compiladores Avanzado– 6			
12	Lenguajes y Compiladores Avanzado– 7	13	Laboratorio – 6	Asignación de laboratorio – 3

14	Lenguajes y Compiladores Avanzado– 8			Asignación de laboratorio – 4 Proyecto – 2
15	Lenguajes y Compiladores Avanzado– 9	16	Laboratorio - 7	Asignación de laboratorio – 5
16	Lenguajes y Compiladores Avanzado– 10	17	Laboratorio – 8	Asignación de laboratorio – 6
18	Lenguajes y Compiladores Avanzado– 11			
19	Lenguajes y Compiladores Avanzado– 12			

5.5 Distribución de sesiones

Tabla XIX. Sesiones curso Organización de Lenguajes y Compiladores 2

SESIÓN NO. 1:	**Duración: 1 horas**
OBJETIVO:	**Establecer los principios básicos del análisis semántico, los atributos y la utilización de los mismos, dentro del análisis.**
CONTENIDO:	1. Análisis semántico 2. Traducción dirigida por la sintaxis 2.1. Atributos 2.2. Árboles de sintaxis 3. Gramáticas atribuidas 3.1. Gramáticas s-atribuidas

	3.2. Gramáticas l-atribuidas 3.3. Gramáticas lc-atribuidas
SESIÓN NO. 2:	**Duración: 1 horas**
OBJETIVO:	**Ejercitar la construcción de gramáticas descendentes predictivas, y la trasformación de otras a este tipo.**
CONTENIDO:	Ejercicio de laboratorio No. 1
SESIÓN NO. 3:	**Duración: 90 minutos**
OBJETIVO:	**Realizar una traducción sencilla utilizando esquemas de traducción dirigida por la sintaxis.**
CONTENIDO:	1. Traducción dirigida por la sintaxis 1.1. Definición dirigida por la sintaxis 1.2. Esquemas de traducción dirigidos por la sintaxis 2. Grafos de dependencia 2.1. Definición 2.2. Algoritmo de construcción
SESIÓN NO. 4:	**Duración: 30 horas**
OBJETIVO:	**Utilizar dentro de definiciones dirigidas por la sintaxis los atributos necesarios para una traducción.**
CONTENIDO:	Ejercicio de laboratorio No. 2 Desarrollo del caso de estudio No.1

SESIÓN NO. 5:	**Duración: 90 minutos**
OBJETIVO:	**Introducir la funcionalidad y estructura de la tabla de símbolos para la construcción de un compilador.**
CONTENIDO:	1. Tabla de símbolos 1.1. Definición 1.2. Símbolo 1.3. Atributos de los símbolos 1.4. Operaciones de la tabla de símbolos 1.5. Estructuras de datos para la tabla de símbolos 2. Interfaz de la tabla de símbolos 3. Datos a almacenar en la tabla de símbolos
SESIÓN NO. 6:	**Duración: 30 minutos**
OBJETIVO:	**Utilizar dentro de definiciones dirigidas por la sintaxis las funciones necesarias para implementar y almacenar datos en la tabla de símbolos.**
CONTENIDO:	Ejercicio de laboratorio No. 3 Asignación de laboratorio No. 1
SESIÓN NO. 7:	**Duración: 1 horas**
OBJETIVO:	**Asimilar los conceptos fundamentales para diseñar e implementar un comprobador de tipos en diferentes tipos de lenguajes, incluidos aquellos que soportan sobrecarga y polimorfismo.**

CONTENIDO:	1. Utilización de la tabla de símbolos en el análisis semántico 2. Sobre carga de funciones y operadores 3. Funciones polifórmicas
SESIÓN NO. 8:	**Duración: 1 horas**
OBJETIVO:	**Ejercitar los conceptos de tabla de símbolos, gestión de tipos y sobrecarga de operaciones o funciones.**
CONTENIDO:	Ejercicio de laboratorio No. 4
SESIÓN NO. 9	**Duración: 90 minutos**
OBJETIVO:	**Determinar la funcionalidad de la comprobación de tipos al momento de construir un compilador.**
CONTENIDO:	1. Descripción funcional. 2. Expresiones de tipos. 3. Sistemas de tipos. 4. Especificación de un comprobador de tipos. 5. Conversiones de tipos. 6. Otros aspectos de un comprobador de tipos: equivalencia, sobrecarga, polimorfismo.

SESIÓN NO. 10:	**Duración: 30 minutos**
OBJETIVO:	**Practicar la construcción de un comprobador de tipos para un lenguaje específico.**
CONTENIDO:	Ejercicio de laboratorio No. 5 Asignación de laboratorio No. 2
SESIÓN NO. 11:	**Duración: 2 horas**
OBJETIVO:	**Establecer el funcionamiento de la memoria en tiempo de ejecución, y la estructura que debe crearse para el manejo por medio del compilador.**
CONTENIDO:	1. Técnicas para la asignación dinámica de la memoria. 2. Organización de la memoria: tipos de entornos de ejecución. 3. Gestión de la memoria en tiempo de ejecución. 4. Aspectos del lenguaje fuente: procedimientos, tipos de paso de parámetros, recursividad, activación de procedimientos, ámbitos, enlace de nombres. 5. Organización de la memoria: registros de activación. 6. Pila de control: acceso a los objetos locales y no locales. 7. Montículo: objetos dinámicos.

SESIÓN NO. 12:	**Duración: 2 horas**
OBJETIVO:	**Comprender el funcionamiento estándar del código intermedio y sus esquemas.**
CONTENIDO:	1. Funcionalidad de la generación de código intermedio 2. Representación de código intermedio por medio de: 2.1. Árboles, 2.2. GDA, 2.3. Código de tres direcciones 2.4. Reutilización de temporales
SESIÓN NO. 13:	**Duración: 1 horas**
OBJETIVO:	**Traducir por medio de tripletas la asignación y las operaciones aritméticas de un código de alto nivel a código intermedio.**
CONTENIDO:	1. Generación de código para expresiones 2. Proposiciones de asignación. 3. Expresiones aritméticas. 4. Reutilización de temporales
SESIÓN NO. 14:	**Duración: 1 horas**
OBJETIVO:	**Ejercitar el proceso de traducción de código por medio de tripletas, optimizando temporales.**
CONTENIDO:	Asignación de laboratorio No. 4

SESIÓN NO. 15:	**Duración: 90 minutos**
OBJETIVO:	**Traducir por medio de tripletas expresiones booleanas y sentencias de control de un código de alto nivel a código intermedio.**
CONTENIDO:	1. Expresiones booleanas 2. Sentencias condicionales 2.1. IF, case 3. Sentencias cíclicas 3.1. For, while, do 3.2. Procedimientos y funciones
SESIÓN NO. 16:	**Duración: 30 minutos**
OBJETIVO:	**Ejercitar el proceso de traducción de código tres direcciones haciendo uso de sentencias if, case, for, while, do.**
CONTENIDO:	Ejercicio de laboratorio No. 5
SESIÓN NO. 17:	**Duración: 1 horas**
OBJETIVO:	**Establecer los diferentes procesos para optimizar código intermedio.**
CONTENIDO:	1. Optimización de código: 1.1. Bloques básicos y optimización local.

	1.2. Eliminación de subexpresiones comunes. 1.3. Eliminación de código muerto. 1.4. Transformaciones aritméticas. 1.5. Empaquetamiento de variables temporales. 2. Mejoras en lazos.
SESIÓN NO. 18:	**Duración: 1 horas**
OBJETIVO:	**Traducir y optimizar código intermedio de un código de alto nivel.**
CONTENIDO:	Asignación de laboratorio No. 6
SESIÓN NO. 19:	**Duración: 2 horas**
OBJETIVO:	**Establecer los pasos para la optimización de código intermedio utilizando el método de mirilla**
CONTENIDO:	1. Representación de bloques básicos por medio de GDA 2. Optimización por medio del método de mirilla 2.1. Eliminación de instrucciones redundantes 2.2. Optimización del flujo de control 2.3. Simplificaciones algebraicas 2.4. Uso de instrucciones especiales de máquina.

SESIÓN NO. 20:	Duración: 2 horas
OBJETIVO:	**Generación de código de máquina**
CONTENIDO:	1. Arquitectura de máquinas objeto 1.1. CISC, RISC 1.2. Código absoluto, código relocalizable 2. Generación de código de máquina 2.1. Modos de direccionamiento 2.2. Instrucciones básicas de ensamblador 2.3. Construcción de árboles de flujo

5.6 Detalle del proyecto

PROYECTO:

Criterio de evaluación:

- Funcionamiento - 70
- Conocimiento técnico - 10
- Documentación - 20

Documentación a entregar: SI

Enunciado

Tabla XX. Proyecto curso Organización de Lenguajes y Compiladores 2

PROYECTO 1:	**Poner en práctica los conocimientos del curso, sobre la fase de análisis de un compilador (análisis léxico, sintáctico y semántico)**

OBJETIVO:	**Que el estudiante ponga en práctica los conocimientos adquiridos sobre las fases de compilación de análisis léxico, análisis sintáctico y análisis semántico por medio de herramientas que le sean de ayuda para la construcción de estos análisis.**
Iniciada en sesión	**7**
Entregada en sesión	**14**
Viva / Demostración	**NO**

PROYECTO 2:	**Poner en práctica los conocimientos adquiridos en traducciones dirigidas por la sintaxis ascendentes y descendentes en la construcción de un intérprete para una función específico.**
OBJETIVO:	**Que el estudiante aplique los conocimientos adquiridos sobre el proceso de compilación e interpretación en el desarrollo de un sistema computacional.**
Iniciada en sesión	**14**
Entregada en sesión	**19**
Viva / Demostración	**NO**

5.7 Detalle del examen

EXAMEN:

Número total de exámenes	1
Examen realizado después de sesión	18
Temas para el examen	Lenguajes y Compiladores Avanzado
Duración	1 ½ hrs.

Criterio de evaluación:

- Objetivo - 40%
- Subjetivo - 60%

5.8 Examen de ejemplo

Figura 3: Ejemplo de examen del curso Compiladores 2

Serie I (40 pts): Resuelva los siguientes casos

1. Considere el siguiente código

```
Proc A
      Var i=3
      Proc B (n)
      Proc D (n)
      Begin D
      i:= i-2
      If n>0 Then Call B(n-1)
End D
Begin B
      i:= i+1
      Call D (n-1)
End B
Proc C
      Begin C
      Call B (i)
      If i>0 Then Call C
      End C
Begin A
Call C
End A
```

(a) Escriba el código de tres direcciones asociado
(b) Especifique los registros de activación
(c) Construya los Bloques básicos
(d) Genere un árbol de flujo
(e) Eliminación de subexpresiones comunes.
(f) Eliminación de código muerto.
(g) Transformaciones aritméticas.
(h) Empaquetamiento de variables temporales.

Serie II (60 pts): Desarrollo los siguientes temas.

- Escriba una definición dirigida por la sintaxis que permia la generación de código intermedio, para el manejo y asignación de matrices numéricas.
- Escriba un Esquema de traducción para la Comprobación de tipos.

5.9 Bibliografía sugerida

1. **Aho, Alfred V., Sethi, Ravi y Ullman, Jeffrey D.** Compiladores, pricipios, técnicas y herramientas. 1990.
2. **Gálvez Rojas, Sergio y Mora Mata, Miguel Ángel.** Traductores y Compiladores con LEX/YACC, JFLEX/CUP Y JAVACC. 2005.
3. **Rodríguez Avila, Eduardo René.** Diseño de Compiladores. 2000.

5.10 Anexos

A. Asignaciones de laboratorio

Asignación de laboratorio No. 1

Objetivo:

Utilizar dentro de definiciones dirigidas por la sintaxis las funciones necesarias para implementar y almacenar datos en la tabla de símbolos.

Especificaciones:

- Diseñar definiciones dirigidas por la sintaxis que reconozca la utilización y asignación de variables.
- Utilización de uno o más diseños de tabla de símbolos
- Desarrollo en pseudo código.
- Deberá ser realizado en forma individual para un mejor aprendizaje.

Asignación de laboratorio No.2

Objetivo:

Practicar la construcción de un comprobador de tipos para un lenguaje específico.

Especificaciones:

- Construcción de un analizador que compruebe tipos de datos.
- Evaluar un archivo de entrada por medio de dicho analizador.
- Generar una salida en donde se identifique el análisis.
- Proporcionar información detallada de los errores que se encuentren.

Asignación de laboratorio No. 3

Objetivo:

Ejercitar el proceso de traducción de código por medio de tripletas, optimizando temporales.

Especificaciones:

- Realice e implementa el funcionamiento de un traductor de código intermedio, utilizando las tripletas como forma de representación.
- A través de un archivo de entrada con un formato específico, se genera el código intermedio asociado.
- Puede realizarse en forma individual para mejorar el aprendizaje.

Asignación de laboratorio No. 5

Objetivo:

Traducir y optimizar código intermedio de un código de alto nivel.

Especificaciones:

- Generar una herramienta que permita la traducción de un código de alto nivel hacia un código intermedio.
- Optimizar dicho código generado a través de diferentes métodos o técnicas.
- Realizado de forma individual.

B. Sección de Proyectos

Proyecto No. 1

Objetivo:

Que el estudiante ponga en práctica los conocimientos adquiridos sobre las fases de compilación de análisis léxico, análisis sintáctico y análisis semántico por medio de herramientas que le sean de ayuda para la construcción de estos análisis.

Especificaciones:

- Se utilizará la arquitectura capas.
- Definiendo una capa de presentación que permita al usuario el ingreso, modificación y eliminación de datos. Se debe tomar en cuenta una interfaz gráfica que sea amigable para el usuario. Es importante también proporcionar ayuda al usuario para la construcción de las entradas.
- Definiendo una capa de negocio con todos los proceso para la manipulación de la información, llevando la lógica del programa.
- Y la capa de datos almacenando y llevando un control minucioso del análisis desarrollado en la capa de negocio (análisis léxico, sintáctico).
- Se debe hacer un instalador interactivo, para la aplicación
- Realizar documentación técnica, de usuario y manual de instalación.
- Enfatizar el análisis y recuperación de errores en todos los análisis.
- Implementado para el análisis de un lenguaje de programación o de otro tipo.

Proyecto No. 2

Objetivo:

Que el estudiante aplique los conocimientos adquiridos sobre el proceso de compilación e interpretación en el desarrollo de un sistema computacional.

Especificaciones:

- Se utilizará la arquitectura capas.
- Definiendo una capa de presentación que permita al usuario el ingreso, modificación y eliminación de datos. Se debe tomar en cuenta una interfaz gráfica que sea amigable para el usuario. Es importante también proporcionar ayuda al usuario para la construcción de las entradas.
- El análisis y traducción debe basarse en las fases de un compilador o intérprete.
- Generación de código intermedio preferentemente tripletas, para su posterior comprobación o compilación.

6. DOCUMENTACIÓN DE APOYO DEL CURSO DE LENGUAJES FORMALES Y DE PROGRAMACIÓN

A continuación se presenta el desarrollo de la documentación de apoyo del curso de Lenguajes Formales y de Programación, el cual se realizó en formato de un libro conteniendo las unidades más importantes impartidas a lo largo del curso

6.1 Datos Generales

NOMBRE DE CURSO: Lenguajes Formales y de Programación (796)

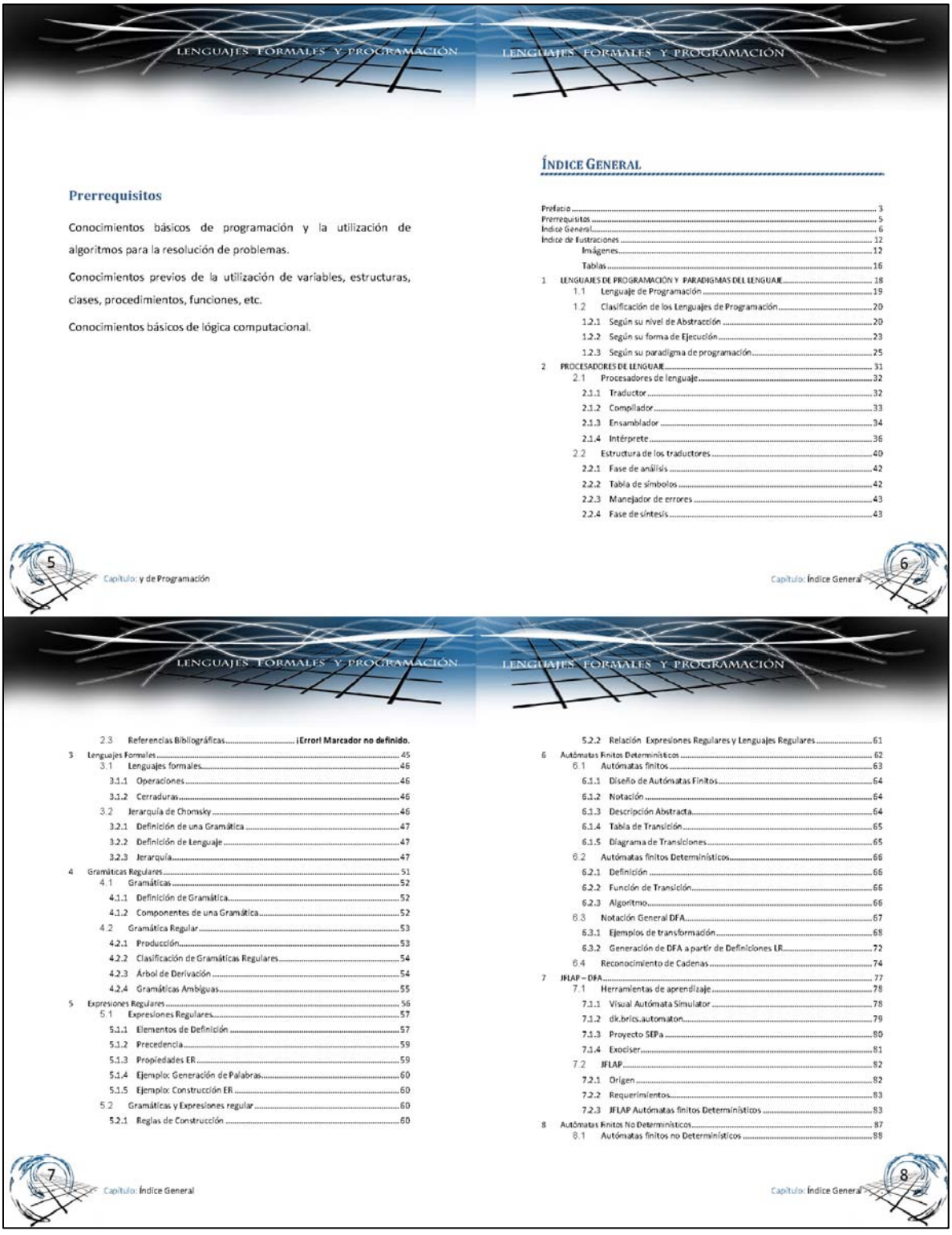
PRE- REQUISITOS: Introducción a la Programación 1 (770)
Lógica de Sistemas (795)
Matemática de Cómputo 1 (960)

POST – REQUISITOS: Organización de Lenguajes y Compiladores 1 (777)
Organización Computacional (964)

OBJETIVO: Brindar a los estudiantes del curso de Lenguajes Formales y de Programación, contiendo teoría, ejemplos, diagramas, resumen, glosario y artículos de interés relacionados a cada una de las unidades del curso.

Figura 4. Páginas 1-4 Libro “Compiladores Principiantes”

Figura 5. Páginas 5-8 Libro “Compiladores Principiantes”



Prerrequisitos

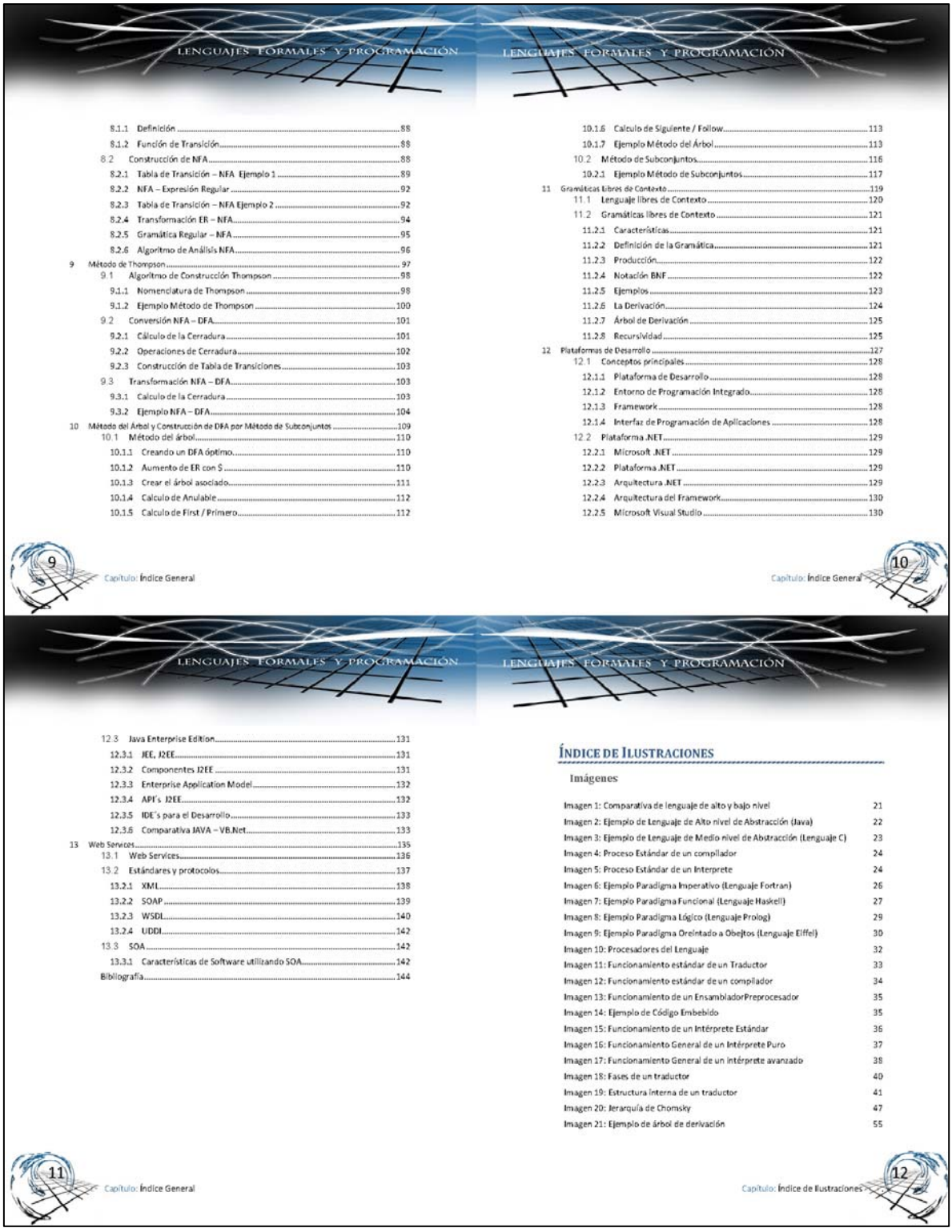
Conocimientos básicos de programación y la utilización de algoritmos para la resolución de problemas.

Conocimientos previos de la utilización de variables, estructuras, clases, procedimientos, funciones, etc.

Conocimientos básicos de lógica computacional.

ÍNDICE GENERAL

Figura 6. Páginas 9-12 Libro “Compiladores Principiantes”



ÍNDICE DE ILUSTRACIONES

Imágenes

Figura 7. Páginas 13-16 Libro “Compiladores Principiantes”

LENGUAJES FORMALES Y PROGRAMACIÓN

LENGUAJES FORMALES Y PROGRAMACIÓN

LENGUAJES FORMALES Y PROGRAMACIÓN

Tablas

6.2 *Lenguajes de Programación y Paradigmas del Lenguaje*

Figura 8. Páginas 17-20 Libro "Compiladores Principiantes"

1 LENGUAJES DE PROGRAMACIÓN Y PARADIGMAS DEL LENGUAJE

NOMBRE DE CURSO: Lenguajes Formales y de Programación
LABORATORIO

MODULO: Lenguajes y Compiladores Principiantes -1

OBJETIVOS:

- Describir que es un lenguaje de programación.
- Identificar las características por las cuales se clasifican los lenguajes de programación.
- Identificar los diferentes paradigmas del lenguaje.
- Reconocer las características principales de cada paradigma.

1.1 Lenguaje de Programación

Los Lenguajes de Programación nacen con el propósito de estandarizar la programación de las computadoras, no importando su sistema operativo o los dispositivos que esta posee. El principal objetivo de los Lenguajes de Programación es controlar el comportamiento de una máquina.

Los Lenguajes de Programación poseen una serie de símbolos, además de reglas que definen la estructura y el funcionamiento de cada uno de ellos. Esta representación simbólica varía de lenguaje a lenguaje, pero proporciona un funcionamiento estándar entre computadora y computadora, permitiendo a un programador almacenar, trasmitir y procesar información, recursos o dispositivos.

Los orígenes de los Lenguajes de Programación parten del cálculo lambda y de la máquina de Turing, desarrollados en la década de 1930, es considerado ser uno de los primeros lenguajes de programación del mundo, incluso ahora muchos de los lenguajes de programación actuales en apariencia utilizan el cálculo de lambda .

El primer lenguaje formal fue diseñado por Konrad Zuse, Plankalkül en los años 40, pero no fue conocido. El primer lenguaje de programación conocido y publicado fue desarrollado por investigadores de la IBM en la década de 1950, FORTRAN, y posterior a este con el propósito de estandarizar los lenguajes de programación un comité de científicos desarrollo ALGOL 58.

Entre los principales lenguajes que dieron origen a los actuales lenguajes de programación está FORTRAN (FORmula TRANslation) creado por IBM en 1957; COBOL (COmmon Business Oriented Language) desarrollado por un comité de científicos en 1959; y por último LISP (LIST Preocessing) creado por John McCarty en 1959.

Todos los lenguajes de programación creados hasta la fecha, es parte del interés constante de empresas, organizaciones y personas particulares de poder controlar y administrar de una manera eficiente la tecnología que constantemente esta en desarrollo e innovación y requieren de lenguajes de programación aptos para el desarrollo de software y aplicaciones.

1.2 Clasificación de los Lenguajes de Programación

Desde que hacen su aparición los lenguajes de programación aproximadamente a finales de los años 40 hasta ahora, aproximadamente 70 años de evolución. Han dado como resultado aproximadamente 115 lenguajes de programación conocidos y utilizados alrededor del mundo.

Es por ello que surge la necesidad de clasificarlo de acuerdo a 3 características importantes: Según su nivel de abstracción; Según su forma de ejecución; y por último según el paradigma de programación. A continuación se detalla más estas características y se identifican algunos lenguajes que pertenecen a cada uno de ellos.

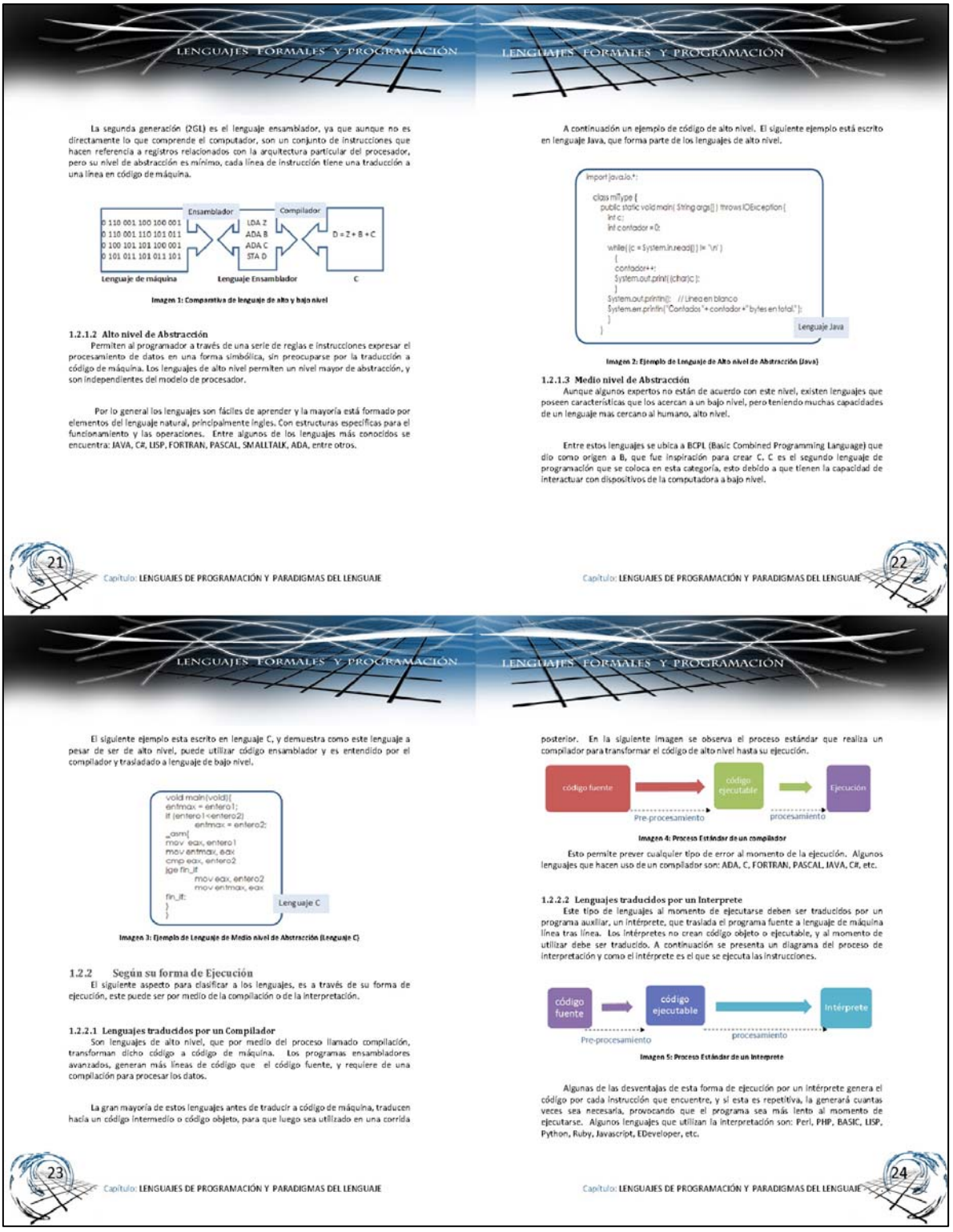
1.2.1 Según su nivel de Abstracción

Los programas escritos en un lenguaje programación deben ser "entendidos" por los circuitos físicos de la máquina para poder ser ejecutados. Es por ello que el nivel de abstracción de un lenguaje nos indica que tan semejante al lenguaje de una computadora es el lenguaje de programación. Este puede ser de alto, bajo o medio nivel de abstracción.

1.2.1.1 Bajo nivel de Abstracción

Este tipo de lenguaje es el más cercano al funcionamiento real de una computadora. Existen dos generaciones en este nivel. La primera generación (1GL) es el lenguaje de código máquina, compuesto por ceros y unos que son lo que realmente comprende el computador. Es el más cercano al funcionamiento real de un computador, por lo que es difícil para ser escrito o leído.

Figura 9. Páginas 21-24 Libro “Compiladores Principiantes”



La segunda generación (2GL) es el lenguaje ensamblador, ya que aunque no es directamente lo que comprende el computador, son un conjunto de instrucciones que hacen referencia a registros relacionados con la arquitectura particular del procesador, pero su nivel de abstracción es mínimo, cada línea de instrucción tiene una traducción a una línea en código de máquina.

Imagen 1: Comparativa de lenguaje de alto y bajo nivel

1.2.1.2 Alto nivel de Abstracción

Permiten al programador a través de una serie de reglas e instrucciones expresar el procesamiento de datos en una forma simbólica, sin preocuparse por la traducción a código de máquina. Los lenguajes de alto nivel permiten un nivel mayor de abstracción, y son independientes del modelo de procesador.

Por lo general los lenguajes son fáciles de aprender y la mayoría está formado por elementos del lenguaje natural, principalmente Ingles. Con estructuras específicas para el funcionamiento y las operaciones. Entre algunos de los lenguajes más conocidos se encuentra: JAVA, C#, LISP, FORTRAN, PASCAL, SMALLTALK, ADA, entre otros.

A continuación un ejemplo de código de alto nivel. El siguiente ejemplo está escrito en lenguaje Java, que forma parte de los lenguajes de alto nivel.

```
import java.io.*;

class miType {
  public static void main( String args[] ) throws IOException {
    int c;
    int contador = 0;

    while( (c = System.in.read()) != '\n' )
      {
      contador++;
      System.out.print( (char)c );
      }
    System.out.println();   // Linea en blanco
    System.err.println("Contados "+ contador +" bytes en total.");
    }
}
```

Lenguaje Java

Imagen 2: Ejemplo de Lenguaje de Alto nivel de Abstracción (Java)

1.2.1.3 Medio nivel de Abstracción

Aunque algunos expertos no están de acuerdo con este nivel, existen lenguajes que poseen características que los acercan a un bajo nivel, pero teniendo muchas capacidades de un lenguaje mas cercano al humano, alto nivel.

Entre estos lenguajes se ubica a BCPL (Basic Combined Programming Language) que dio como origen a B, que fue inspiración para crear C. C es el segundo lenguaje de programación que se coloca en esta categoría, esto debido a que tienen la capacidad de interactuar con dispositivos de la computadora a bajo nivel.

El siguiente ejemplo esta escrito en lenguaje C, y demuestra como este lenguaje a pesar de ser de alto nivel, puede utilizar código ensamblador y es entendido por el compilador y trasladado a lenguaje de bajo nivel.

```
void main(void){
entmax = entero1;
if (entero1<entero2)
        entmax = entero2;
_asm{
mov eax, entero1
mov entmax, eax
cmp eax, entero2
jge fin_if
        mov eax, entero2
        mov entmax, eax
fin_if:
}
}
```

Lenguaje C

Imagen 3: Ejemplo de Lenguaje de Medio nivel de Abstracción (Lenguaje C)

1.2.2 Según su forma de Ejecución

El siguiente aspecto para clasificar a los lenguajes, es a través de su forma de ejecución, este puede ser por medio de la compilación o de la interpretación.

1.2.2.1 Lenguajes traducidos por un Compilador

Son lenguajes de alto nivel, que por medio del proceso llamado compilación, transforman dicho código a código de máquina. Los programas ensambladores avanzados, generan más líneas de código que el código fuente, y requiere de una compilación para procesar los datos.

La gran mayoría de estos lenguajes antes de traducir a código de máquina, traducen hacia un código intermedio o código objeto, para que luego sea utilizado en una corrida

posterior. En la siguiente imagen se observa el proceso estándar que realiza un compilador para transformar el código de alto nivel hasta su ejecución.

Imagen 4: Proceso Estándar de un compilador

Esto permite prever cualquier tipo de error al momento de la ejecución. Algunos lenguajes que hacen uso de un compilador son: ADA, C, FORTRAN, PASCAL, JAVA, C#, etc.

1.2.2.2 Lenguajes traducidos por un Interprete

Este tipo de lenguajes al momento de ejecutarse deben ser traducidos por un programa auxiliar, un intérprete, que traslada el programa fuente a lenguaje de máquina línea tras línea. Los intérpretes no crean código objeto o ejecutable, y al momento de utilizar debe ser traducido. A continuación se presenta un diagrama del proceso de interpretación y como el intérprete es el que se ejecuta las instrucciones.

Imagen 5: Proceso Estándar de un Interprete

Algunas de las desventajas de esta forma de ejecución por un Intérprete genera el código por cada instrucción que encuentre, y si esta es repetitiva, la generará cuantas veces sea necesaria, provocando que el programa sea más lento al momento de ejecutarse. Algunos lenguajes que utilizan la interpretación son: Perl, PHP, BASIC, LISP, Python, Ruby, Javascript, EDeveloper, etc.

Figura 10. Páginas 25-28 Libro “Compiladores Principiantes”

1.2.3 Según su paradigma de programación

Los paradigmas de programación son filosofías o enfoques de construcción de software, es por ello que dependiendo de la situación cada uno tiene ventajas y desventajas.

1.2.3.1 Paradigma Imperativo

Este paradigma estructura la programación por medio de estados, que varían dependiendo de sentencias definidas. Los programas desarrollados bajo este paradigma poseen un conjunto de instrucciones que le indican al computador que operaciones, tareas y procesos debe de realizar.

Las mismas computadoras están diseñadas bajo este paradigma, trabajando bajo un conjunto de instrucciones de bajo nivel ya definidas. Es por ello que este tipo de paradigma fácilmente se puede traducir, a través de un compilador, a lenguaje de máquina. Dependiendo del nivel de abstracción puede ser complicado construir código en lenguaje imperativo. Pero debido a la cercanía con el lenguaje de máquina la eficiencia es muy alta en cuanto a su ejecución.

Las principales características del paradigma imperativo son:

- Los datos son almacenados en un tipo específico con propiedades y comportamiento bien definido.
- Utiliza el cambio de estados a través de los valores de las variables y por las entradas y salidas de los dispositivos.
- El flujo de control es bien definido a través de elementos como le secuencia, selección, repetición y rutinas.
- Utilizan jerarquía por medio de bloques, rutinas, módulos y programas.
- La asignación de valores a una posición de memoria.
- Utiliza los algoritmos como fuente de implementación.


```
PROGRAM pares
* Suma de los números pares menores de 102
      i=2
      suma=0
10    suma=suma + i
      i=i+2
      IF (i-100)10,10,20
20    STOP
      END
```

Lenguaje Fortran

Imagen 6: Ejemplo Paradigma Imperativo (Lenguaje Fortran)

Aunque existen muchos lenguajes que implementan estas características, en este paradigma claramente se identifican: BASIC, C, C++, Fortran, PASCAL, etc. tomando en cuenta que algunos lenguajes son multiparadigma.

1.2.3.2 Paradigma Funcional

Este paradigma plasma las ideas y filosofías a partir del cálculo de lambda, desarrollado por Alonzo Church. El cálculo de lambda, a través de la utilización de funciones ha demostrado resolver de forma equivalente un problema, como si fuera la máquina de Turing.

El principal elemento de este paradigma es la función, todo se maneja a través de funciones: variables, expresiones y estas no son modificables luego de asignado su valor. Una función está asociada a un conjunto de elementos de origen y otro conjunto de elementos de destino, dominio y rango. Estas funciones aplican operaciones sobre algunos valores, argumentos, para obtener un resultado.

Los lenguajes que pertenecen a este paradigma funcional pueden ser puros o híbridos, dependiendo si admiten algunas propiedades del paradigma imperativo como lo es la asignación. Las principales características del paradigma funcional son:

- Definición y evaluación de funciones como forma de abstracción, y la evaluación de expresiones como forma de computación.
- Uso de la recursión dentro de las funciones en lugar de realizar iteraciones.
- Funciones como datos primitivos, como todos los elementos de este lenguaje son funciones posibilita el uso de variables como el valor de un procedimiento, una función toma argumentos de otra función, retorna funciones o procedimientos, o estructuras.

Entre los lenguajes funcionales puros, los más relevantes son los lenguajes Haskell y Miranda. Y entre los funcionales híbridos más conocidos son Scala, Lisp, Scheme, SAP y Standard ML. A continuación un ejemplo de este.

```
module Pila (TPila, mete, saca, vacia, muestra) where
data TPila a = PilaVacia | Apila a (TPila a)
mete::a -> TPila a -> TPila a
mete x p = Apila x p
saca::TPila a ->TPila a
saca (Apila _ p) = p
saca PilaVacia = error "Sacando elementos de pila vacia"
vacia::TPila a
vacia = PilaVacia
muestra::(Show a)=>TPila a ->String
muestra PilaVacia = ""
muestra (Apila x p) = " " ++ show x ++ muestra p
```

Lenguaje Haskell

Imagen 7: Ejemplo Paradigma Funcional (Lenguaje Haskell)

1.2.3.3 Paradigma Lógico

La programación lógica comprende dos paradigmas de programación: la programación declarativa y la programación funcional. Ya que a través del uso de la lógica de predicativo y la relación de estos elementos dentro del ambiente, se encuentran soluciones a problemas dependientes del contexto.

Este paradigma trabaja de una forma descriptiva, estableciendo relaciones entre entidades, definiendo que es y que no es. A través de identificadores llamados functores y las reglas o sentencias, se establecen relaciones entre varios elementos indicándoles la lógica necesaria para resolver un problema. Este tipo de lenguaje es utilizado en inteligencia artificial para dar cierta lógica a la resolución de problemas. (3)

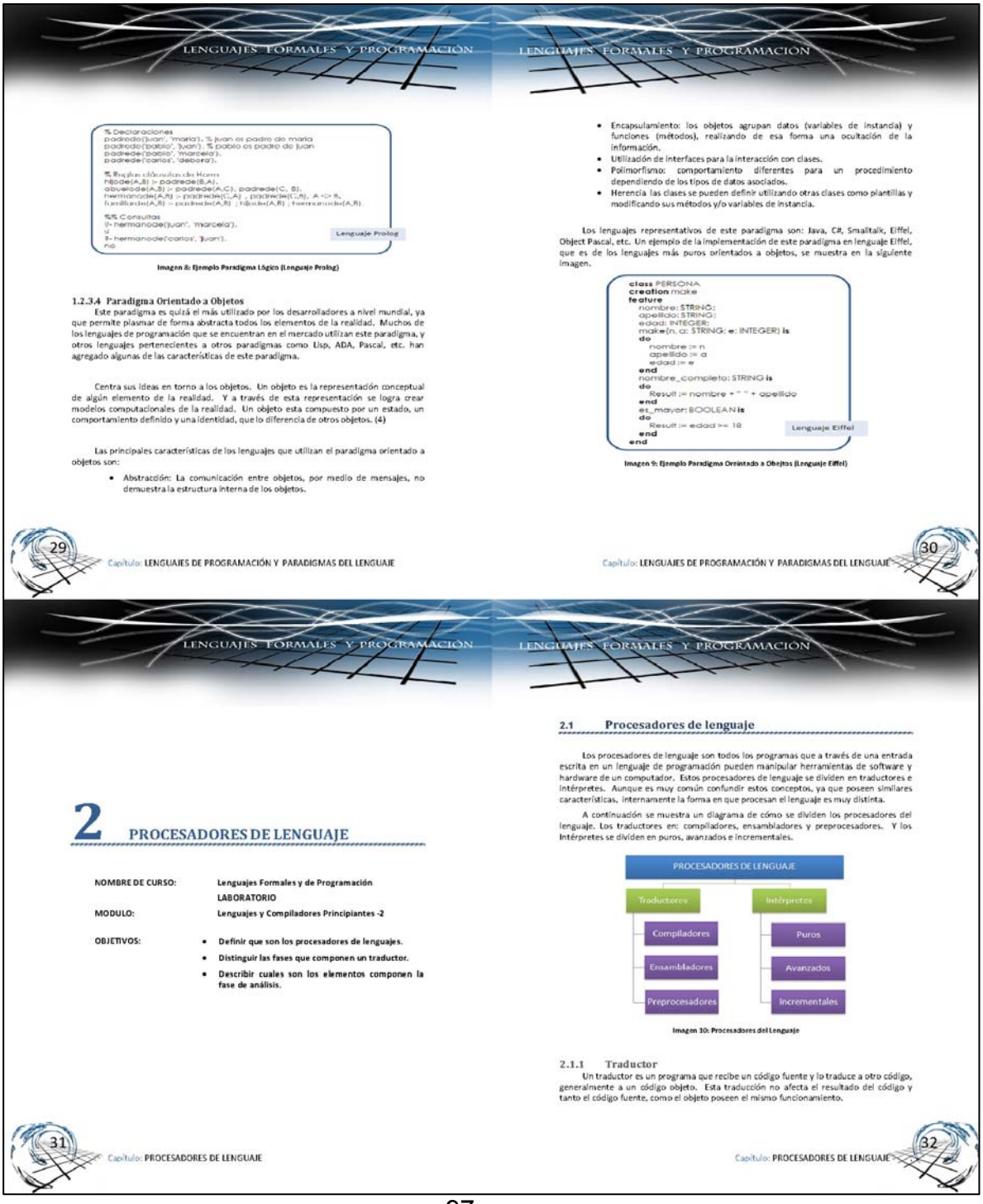
Las principales características de este paradigma son:

- Están construidos únicamente por expresiones lógicas (verdadero – falso)
- El orden de ejecución de instrucciones no se realiza por una llamada o de forma secuencial.
- Utiliza reglas o cláusulas de Horn, que cuando se de un conjunto de restricciones, se ejecutará automáticamente dicha regla.

El lenguaje que utiliza este paradigma es Prolog (Programming in Logic), que permite definir los elementos y su relación, para luego resolver cuestionamientos de verdadero o falso. Es utilizado generalmente en el campo de la inteligencia artificial: en sistemas expertos, procesamiento de lenguaje natural.

6.3 Procesadores de lenguaje

Figura 11. Páginas 29-32 Libro "Compiladores Principiantes"




```
% Declaraciones
padrede('juan', 'maria'). % juan es padre de maria
padrede('pablo', 'juan'). % pablo es padre de juan
padrede('pablo', 'marcela').
padrede('carlos', 'deborah').

% Reglas cláusulas de Horn
hijode(A,B) :- padrede(B,A).
abuelode(A,B) :- padrede(A,C), padrede(C, B).
hermanode(A,B) :- padrede(C,A) , padrede(C,B), A <> B.
familiarde(A,B) :- padrede(A,B) ; hijode(A,B) ; hermanode(A,B).

%% Consultas
?- hermanode('juan', 'marcela').
y
?- hermanode('carlos', 'juan').
no
```

Lenguaje Prolog

Imagen 8: Ejemplo Paradigma Lógico (Lenguaje Prolog)

1.2.3.4 Paradigma Orientado a Objetos

Este paradigma es quizá el más utilizado por los desarrolladores a nivel mundial, ya que permite plasmar de forma abstracta todos los elementos de la realidad. Muchos de los lenguajes de programación que se encuentran en el mercado utilizan este paradigma, y otros lenguajes pertenecientes a otros paradigmas como Lisp, ADA, Pascal, etc. han agregado algunas de las características de este paradigma.

Centra sus ideas en torno a los objetos. Un objeto es la representación conceptual de algún elemento de la realidad. Y a través de esta representación se logra crear modelos computacionales de la realidad. Un objeto esta compuesto por un estado, un comportamiento definido y una identidad, que lo diferencia de otros objetos. (4)

Las principales características de los lenguajes que utilizan el paradigma orientado a objetos son:

- Abstracción: La comunicación entre objetos, por medio de mensajes, no demuestra la estructura interna de los objetos.

- Encapsulamiento: los objetos agrupan datos (variables de instancia) y funciones (métodos), realizando de esa forma una ocultación de la información.
- Utilización de interfaces para la interacción con clases.
- Polimorfismo: comportamiento diferentes para un procedimiento dependiendo de los tipos de datos asociados.
- Herencia las clases se pueden definir utilizando otras clases como plantillas y modificando sus métodos y/o variables de instancia.

Los lenguajes representativos de este paradigma son: Java, C#, Smalltalk, Eiffel, Object Pascal, etc. Un ejemplo de la implementación de este paradigma en lenguaje Eiffel, que es de los lenguajes más puros orientados a objetos, se muestra en la siguiente imagen.

```
class PERSONA
creation make
feature
  nombre: STRING;
  apellido: STRING;
  edad: INTEGER;
  make(n, a: STRING; e: INTEGER) is
  do
    nombre := n
    apellido := a
    edad := e
  end
  nombre_completo: STRING is
  do
    Result := nombre + " " + apellido
  end
  es_mayor: BOOLEAN is
  do
    Result := edad >= 18
  end
end
```

Lenguaje Eiffel

Imagen 9: Ejemplo Paradigma Orientado a Objetos (Lenguaje Eiffel)

2 PROCESADORES DE LENGUAJE

NOMBRE DE CURSO:	**Lenguajes Formales y de Programación LABORATORIO**
MODULO:	**Lenguajes y Compiladores Principiantes -2**
OBJETIVOS:	• **Definir que son los procesadores de lenguajes.** • **Distinguir las fases que componen un traductor.** • **Describir cuales son los elementos componen la fase de análisis.**

2.1 Procesadores de lenguaje

Los procesadores de lenguaje son todos los programas que a través de una entrada escrita en un lenguaje de programación pueden manipular herramientas de software y hardware de un computador. Estos procesadores de lenguaje se dividen en traductores e intérpretes. Aunque es muy común confundir estos conceptos, ya que poseen similares características, internamente la forma en que procesan el lenguaje es muy distinta.

A continuación se muestra un diagrama de cómo se dividen los procesadores del lenguaje. Los traductores en: compiladores, ensambladores y preprocesadores. Y los Intérpretes se dividen en puros, avanzados e incrementales.

Imagen 10: Procesadores del Lenguaje

2.1.1 Traductor

Un traductor es un programa que recibe un código fuente y lo traduce a otro código, generalmente a un código objeto. Esta traducción no afecta el resultado del código y tanto el código fuente, como el objeto poseen el mismo funcionamiento.

Figura 12. Páginas 33-36 Libro “Compiladores Principiantes”

Los traductores utilizan un lenguaje fuente LF que utiliza el traductor para generar a través de un lenguaje de implementación LI o lenguaje host un lenguaje objeto LO. Es decir, un LF por medio de un LI genera un LO. En la siguiente imagen se puede comprender esta secuencia.

Imagen 11: Funcionamiento estándar de un Traductor

2.1.2 Compilador

Los compiladores son programas que reciben un lenguaje fuente de nivel medio o superior y lo transforman a un programa objeto, que puede ser ensamblador o de máquina. Un compilador traduce, pero no es capaz de ejecutar pues está no es su función. Además de esto los compiladores pueden reconocer errores al momento de compilar y no generará un código objeto hasta no tener el lenguaje fuente sin ningún error.

Imagen 12: Funcionamiento estándar de un compilador

El programa objeto luego debe de ser ejecutado, y a través de la ejecución se le introducen los datos para obtener los resultados esperados.

2.1.3 Ensamblador

Los ensambladores son programas que traducen de lenguaje ensamblador a lenguaje de máquina. A través de los ensambladores se integran los módulos de un programa, identifica y resuelve direcciones de memoria, especifica la memoria necesaria para la ejecución, incorpora datos y códigos necesarios para la futura ejecución.

Un ensamblador está relacionado internamente con la arquitectura del computador, esto debido a que debe generar código de máquina específico para el modelo, lee cada línea en lenguaje ensamblador y la traduce a una línea de código de máquina. Existen algunos ensambladores llamados macroensambladores que utilizan macroinstrucciones, ésta macroinstrucción se traduce a varias instrucciones de máquina.

Imagen 13: Funcionamiento de un Ensamblador/Preprocesador

El preprocesador o precompilador, es un programa que se ejecuta antes del compilador. Este programa es capaz de reconocer lenguaje embebido o incrustado dentro del lenguaje principal o anfitrión, y luego convertir este lenguaje embebido hacia el lenguaje anfitrión.

El lenguaje anfitrión es cualquier tipo de lenguaje de alto nivel, y puede contener un lenguaje embebido de un nivel superior, como una consulta hacia una base de datos, un cálculo estadístico, una simulación, etc. Este tipo de traductores es utilizado para la incorporación de nuevas herramientas de propósito especial dentro de lenguajes ya establecidos.

En la siguiente imagen se puede observar un código escrito en lenguaje C, que tiene embebido lenguaje ensamblador. Para que sea ejecutado un traductor leerá este código y lo transformará de forma que lo comprenda el compilador de C.

Imagen 14: Ejemplo de Código Embebido

2.1.3.1 Pseudocompilador

Es un programa muy similar a un compilador, con la diferencia principal que el código objeto que genera no puede ser ejecutado en un computador, es indispensable la ejecución en una máquina virtual. El código generado se le denomina código de byte.

La principal ventaja es que permite utilizar emuladores de los microprocesadores existentes, y al momento de compilar se produce el código necesario para cada uno de ellos. Esto permite que el código sea independiente de la plataforma y del hardware.

2.1.4 Intérprete

Un intérprete es un programa diseñado para ejecutar instrucciones de un código determinado, no genera ningún tipo de código adicional, y es completamente ejecutable. Los intérpretes permiten ejecutar rutinas, aunque generan mayor número de líneas de código que un compilador.

El funcionamiento de un intérprete se basa en el código fuente, la interpretación de acuerdo a los datos ingresados y los resultados obtenidos. En la siguiente figura se muestra dicho proceso.

Imagen 15: Funcionamiento de un Intérprete Estándar

Los intérpretes generan código más pesado, sin embargo son más flexibles al momento de compilar y depurar. Además la mayoría de los intérpretes permiten la portabilidad del lenguaje y la sencillez en su implementación.

Figura 13. Páginas 37-40 Libro "Compiladores Principiantes"

El proceso de interpretación de un lenguaje puede ser iterativo o recursivo. La interpretación iterativa es por medio de la ejecución línea tras línea, donde cada línea se analiza y luego se ejecuta; hasta no encontrar otra instrucción. Mientras que la interpretación recursiva se hace en dos fases, una de especificación semántica y la otra de implementación, este tipo de interpretación no es muy adecuado debido ineficiencia al momento de ejecutar.

Los intérpretes se dividen en puros, avanzados e incrementales.

2.1.4.1 Intérpretes puros

Este tipo de intérpretes surgen en la primera generación de computadores, pues era la forma en que trabajaban estos. El proceso es a través del análisis de cada sentencia y su ejecución hasta finalizar todo el programa fuente. Al momento de encontrar algún error se debe de comenzar nuevamente el proceso.

En la siguiente figura se observa el esquema general de un intérprete puro, donde a través de un lenguaje fuente, el intérprete comienza a procesar instrucción por instrucción, colocándola temporalmente en la memoria para crear la tabla de etiquetas donde se almacenan los procedimientos y funciones del programa. Al momento de evaluar el código almacenado en memoria, se ingresan los datos necesarios y si es correcta la sintaxis de esta se ejecuta, y se continua el proceso, buscando dentro de la memoria la siguiente instrucción.

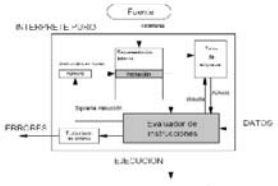


Imagen 15: Funcionamiento General de un Intérprete Puro

2.1.4.2 Intérpretes avanzados

Los intérpretes avanzados completan el proceso de interpretación con un análisis previo del programa fuente, para establecer si existe algún tipo de error sintáctico previo a la traducción, esto permite a demás que el intérprete genere código intermedio que será ejecutado en la memoria, código de byte. Esto facilita la ejecución de este código en cualquier sistema operativo que contenga un intérprete de código de byte.

Los lenguajes como Quick-BASIC, Turbo Pascal y el lenguaje Java son ejemplos claros de este tipo de intérprete. Por ejemplo, en lenguaje Java, cuando se utiliza un programa con extensión .java se compila produciendo varios ficheros de extensión *.class* (escritos en *bytecode*) que luego pueden ser utilizados en aplicaciones de internet.

En la siguiente imagen se muestra el esquema general de un intérprete avanzado, la fase previa de análisis y la utilización de un intérprete puro para la ejecución de este.

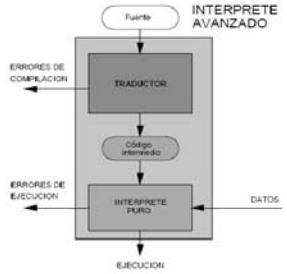


Imagen 17: Funcionamiento General de un intérprete avanzado

2.1.4.3 Intérpretes Incrementales

Debido a la existencia de algunos lenguajes que permiten la creación y el manejo de funciones u objetos luego del tiempo de compilación, son creados durante la ejecución. Existen traductores, denominados intérpretes incrementales o evaluadores parciales. En los que a medida que se va ejecutando y se ingresan los datos, se van creando las estructuras necesarias para poder procesar otro subconjunto de instrucciones o datos dependientes del código ya analizado.

2.2 Estructura de los traductores

La estructura interna de los traductores de lenguaje consiste en dos fases importantes el análisis y la síntesis. En ellas el traductor, ya sea un compilador o un intérprete logra reconocer un lenguaje específico, analiza el código fuente genera un reporte de errores de este análisis. Si es adecuado el código fuente pasa a la fase de síntesis en la genera el código objeto o el código destino, mientras no exista ningún tipo de error. En la siguiente imagen se puede observar este proceso.

Imagen 18: Fases de un traductor

La fase de análisis se encarga de verificar el código fuente, a través de la descomposición del código y el análisis minucioso a través de los componentes léxicos o *tokens*. Luego de agrupar todos los elementos en *tokens* se verifica su estructura sintáctica y semántica de cada sentencia o instrucción del lenguaje. A través de este análisis construye una tabla de símbolos en los que se almacenan todos los identificadores encontrados. Si existe algún error dentro del código, se deben mostrar los errores para su posterior manejo.

Figura 14. Páginas 41-44 Libro "Compiladores Principiantes"

La fase de síntesis se encarga de la generación del lenguaje intermedio, o lenguaje objeto, que servirá como apoyo en la generación de código de máquina final. En está fase de análisis se realiza un reporte de los errores al momento de general el código y preservar la funcionalidad adecuada del código generado.

En la siguiente imagen se muestra la estructura interna de los traductores, y todos los pasos necesarios que debe llevar este a partir de un programa fuente para analizar y sintetizar hacia un programa objeto.

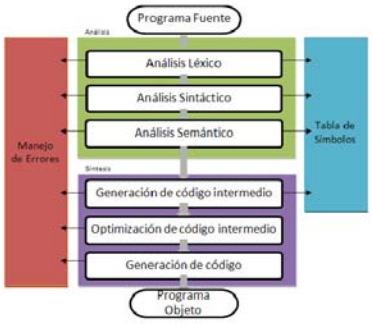


Imagen 19: Estructura interna de un traductor

2.2.1 Fase de análisis

2.2.1.1 Análisis léxico

En esta fase del análisis se toman todos los elementos del programa fuente, agrupando cada uno de los caracteres para formar *tokens* o componentes léxicos. Por medio de estos *tokens* se logra identificar: constantes, identificadores, variables, funciones, clases, palabras reservadas, operadores, etiquetas, etc. Este análisis se conoce también como explorador o *scanner*.

2.2.1.2 Análisis sintáctico

Esta fase también es conocida como *parsing*, en la que a través del agrupamiento de los *tokens* se establece la estructura de las sentencias del programa a través de clases sintácticas. Este análisis crea un árbol sintáctico que es construido a partir de unas reglas llamadas gramática. Existen dos clases de analizadores sintácticos los ascendentes y los descendentes dependiendo de su dirección de análisis.

2.2.1.3 Análisis semántico

En esta fase del análisis se verifica el contexto en el que se encuentra cada sentencia sintáctica, por lo general se realiza simultáneamente con el análisis sintáctico. Dentro de este análisis se encuentra la evaluación de tipos o comprobación de tipos, en la que se verifica el contexto de cada uno de los tipos utilizados y su adecuado manejo.

2.2.2 Tabla de símbolos

La tabla de símbolos permite almacenar todos los identificadores que se encuentran al momento del análisis, pueden ser variables, constantes, procedimientos o tipos. Esta información es de utilidad al momento de realizar un análisis semántico y en la comprobación de tipos. La tabla de símbolos puede ser almacena en una tabla de hash, un árbol binario o cualquier otra estructura que permita hacer búsquedas rápidas.

Los principales atributos que poseen las tablas de símbolos son:

- Detalle del identificador: variable, función, constante, procedimiento, etc.
- Tipo: real, carácter, cadena, entero, etc.
- Tamaño: puede ser definido por dimensiones o numero de bytes a utilizar.
- Ubicación de inicio dentro de la memoria, para procedimientos y funciones.
- Lista de detalles, si se necesita algún tipo de información adicional de campos, estructuras, registros o parámetros, etc.

2.2.3 Manejador de errores

El manejador de errores permite establecer si el programa fuente posee algún error ya sea de tipo léxico (caracteres no definidos), sintáctico (estructura inadecuada) o semántico (contexto no apropiado) permitiendo así la depuración de los errores. Es indispensable que el manejador de errores, genere un reporte específico para poder ajustar el código en forma adecuada.

2.2.4 Fase de síntesis

2.2.4.1 Generación de código intermedio

La generación de código intermedio permite generar código que es portable e independiente de la máquina, ya que es entendible solamente por una máquina abstracta. En muchas ocasiones este código puede ser lenguaje C, y puede utilizar cuartetos o tercetos para representar el código, pero se usa comúnmente la notación polaca; tomando en cuenta que esta es más fácil para traducir a código de máquina.

2.2.4.2 Optimización de código

La optimización de código es la verificación que se hace del código ya generado para mejora la velocidad al momento de la ejecución. Existen algunas reglas para optimizar que elimina la reescritura de código, la eliminación de código inalcanzable, la optimización de temporales, etc. que permiten un código mas eficiente.

La optimización de código en algunos traductores se hace en el código intermedio, aunque hay algunos que hacen la optimización en el código de máquina.

2.2.4.3 Generación de código

Posterior a la generación de código intermedio se procede a la traducción de este código a código ensamblador o para el funcionamiento dentro de una pila. Cada una de las instrucciones pasa a ser instrucciones en código de máquina donde se involucran posiciones reales de memoria por medio de los registros.

7. DOCUMENTACIÓN DE APOYO DEL CURSO DE ORGANIZACIÓN DE LENGUAJES Y COMPILADORES 1

A continuación se presenta el desarrollo de la documentación de apoyo del curso de Organización de Lenguajes y Compiladores 1, el cual se realizó en formato de un libro conteniendo las unidades más importantes impartidas a lo largo del curso

7.1 Datos generales

NOMBRE DE CURSO:	**Organización de Lenguajes y Compiladores 1 (777)**
PRE- REQUISITOS:	Introducción a la Programación 2 (771) Lenguajes Formales y de Programación (796) Matemática de Cómputo 2 (962)
POST – REQUISITOS:	Organización de Lenguajes y Compiladores 2 (781)
OBJETIVO:	Brindar a los estudiantes del curso de Compiladores, contiendo teórico necesario para la construcción de compiladores e intérpretes, utilizando ejemplos, diagramas y artículos de interés relacionados a cada una de las unidades del curso.

Figura 15. Páginas 1-4 Libro “Compiladores Intermedio”

Figura 16. Páginas 5-8 Libro “Compiladores Intermedio”

Prerrequisitos

Conocimientos básicos de programación y la utilización de algoritmos para la resolución de problemas.

Conocimientos básicos de lógica computacional.

Conocimientos básicos de procesadores de lenguaje manejo y manipulación de expresiones regulares, construcción de gramáticas y teoría de autómatas.

ORGANIZACIÓN DE LENGUAJES Y COMPILADORES 1

ÍNDICE GENERAL

ORGANIZACIÓN DE LENGUAJES Y COMPILADORES 1

ORGANIZACIÓN DE LENGUAJES Y COMPILADORES 1

Figura 17. Páginas 9-12 Libro “Compiladores Intermedio”

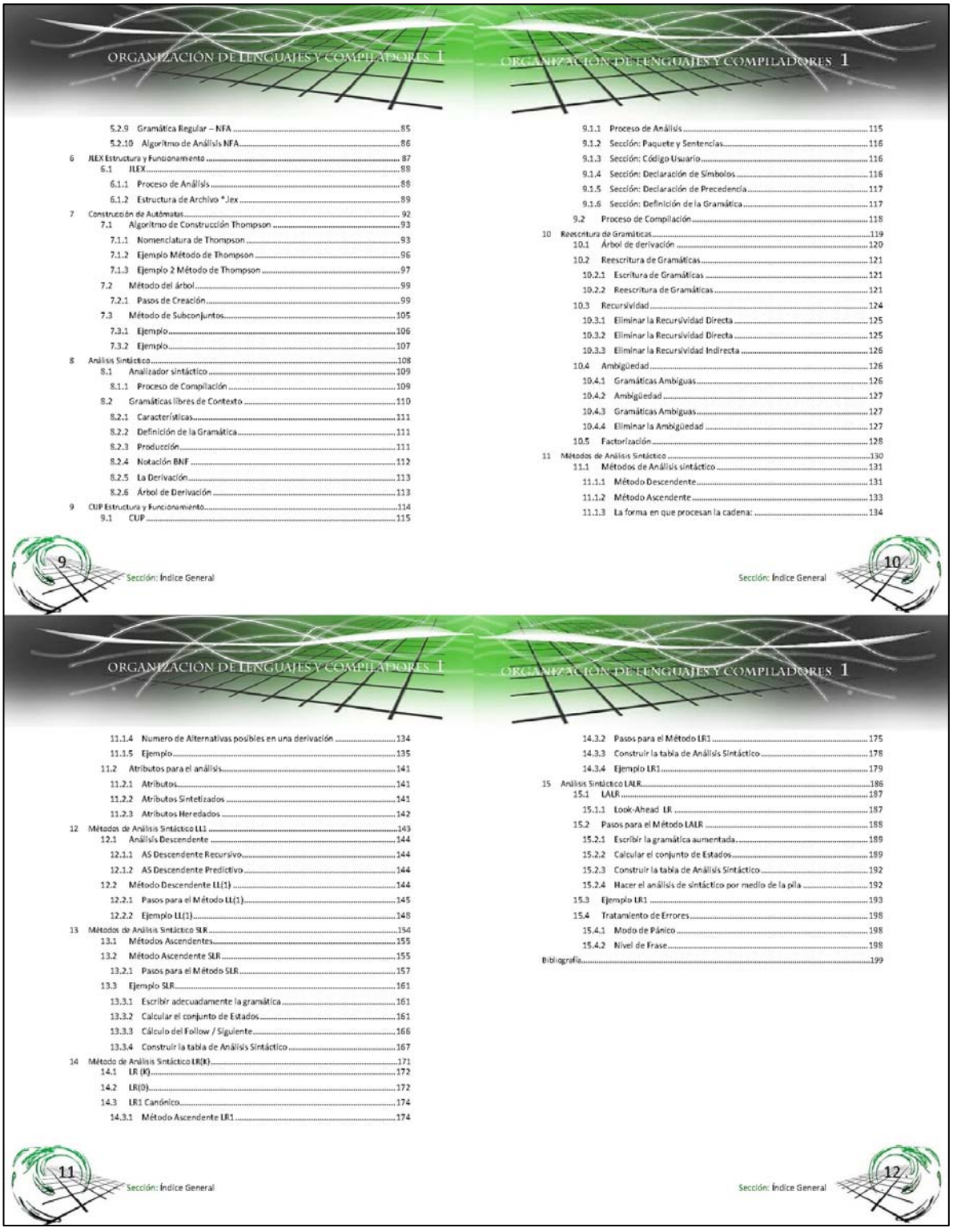
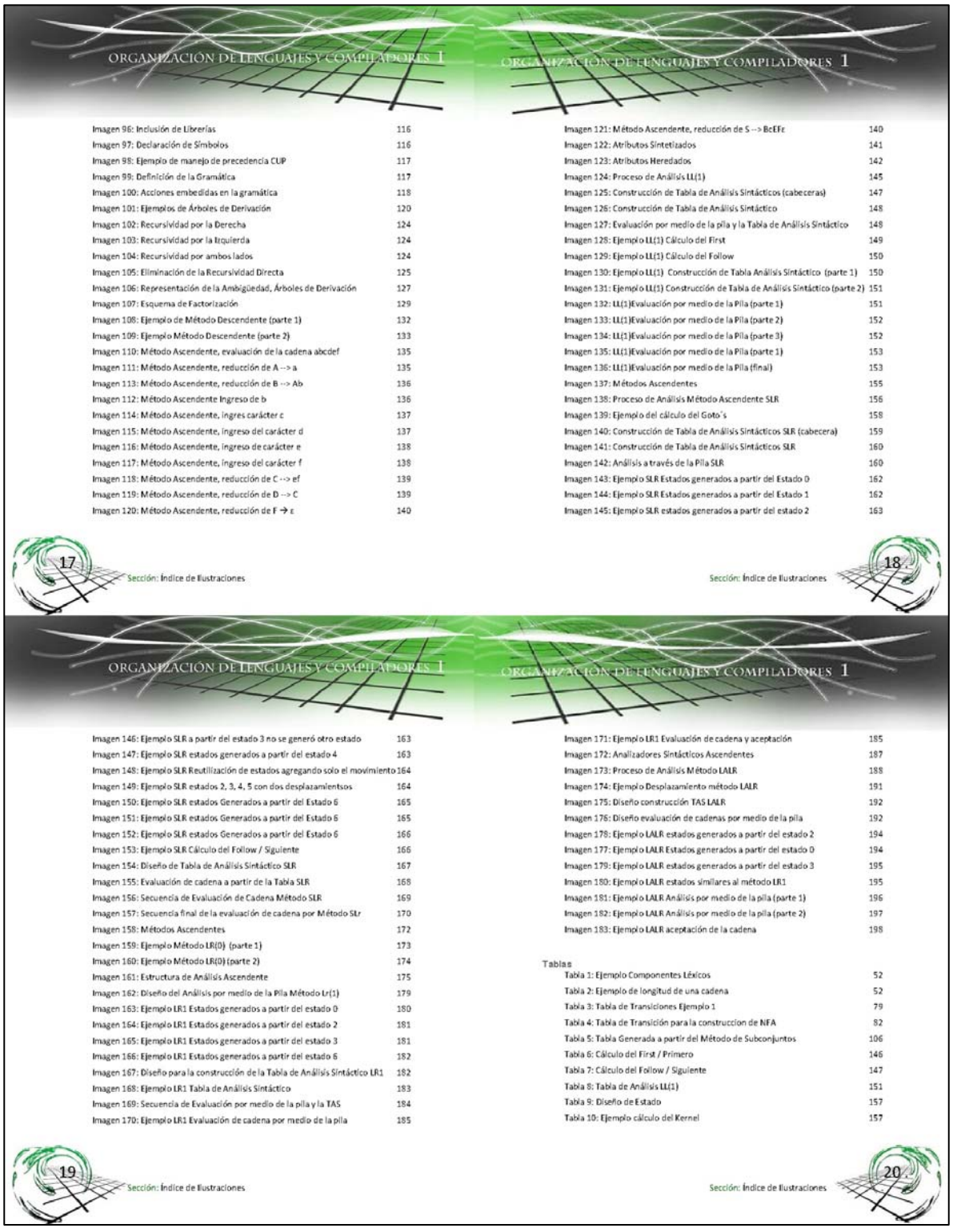


Figura 18. Páginas 13-16 Libro “Compiladores Intermedio”

ÍNDICE DE ILUSTRACIONES

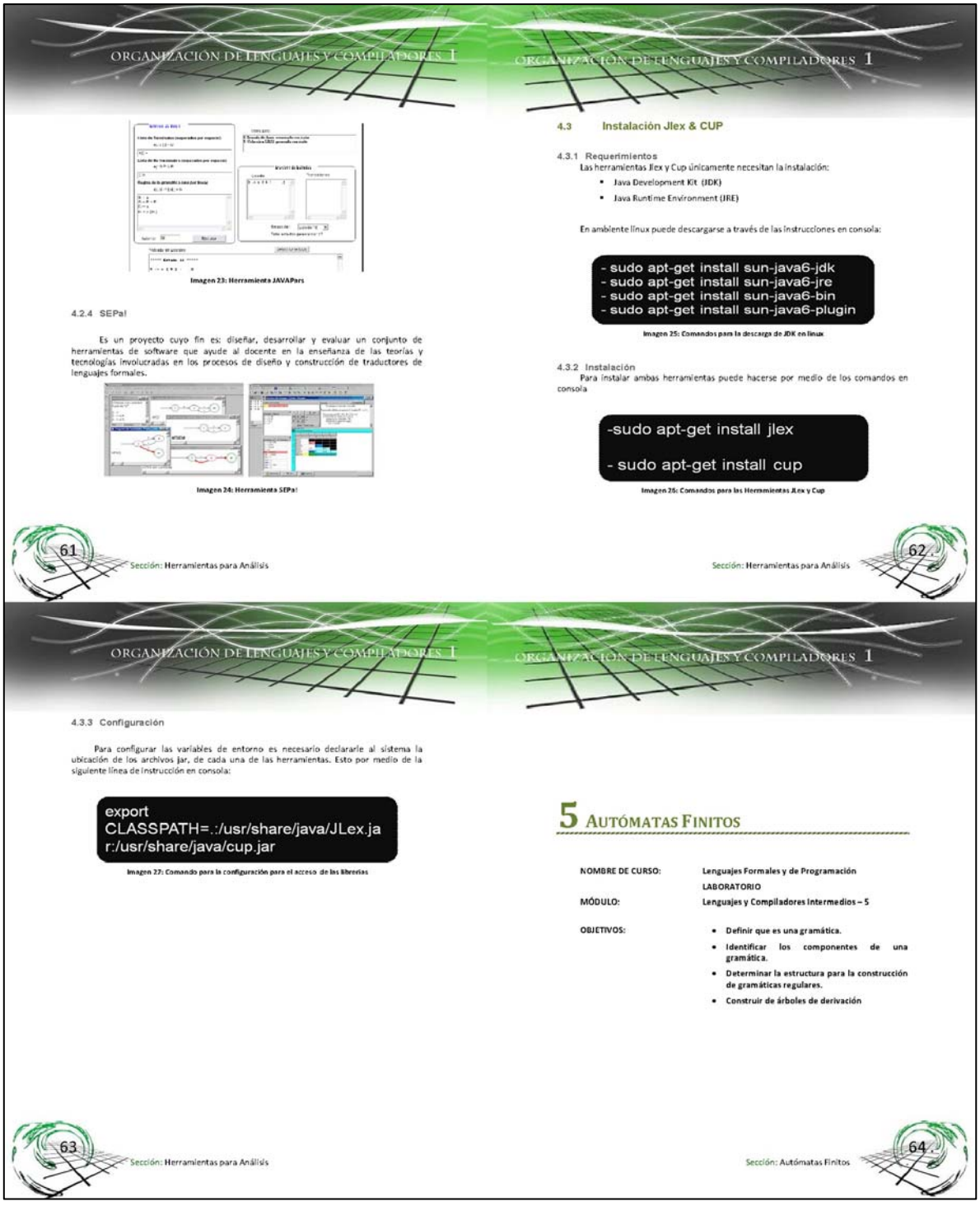
Imagenes

Figura 19. Páginas 17-20 Libro “Compiladores Intermedio”



7.2 Autómatas finitos

Figura 20. Páginas 61-64 Libro "Compiladores Intermedio"



ORGANIZACIÓN DE LENGUAJES Y COMPILADORES 1

Imagen 23: Herramienta JAVAPars

4.2.4 SEPal

Es un proyecto cuyo fin es: diseñar, desarrollar y evaluar un conjunto de herramientas de software que ayude al docente en la enseñanza de las teorías y tecnologías involucradas en los procesos de diseño y construcción de traductores de lenguajes formales.

Imagen 24: Herramienta SEPal

61 Sección: Herramientas para Análisis

ORGANIZACIÓN DE LENGUAJES Y COMPILADORES 1

4.3 Instalación Jlex & CUP

4.3.1 Requerimientos

Las herramientas Jlex y Cup únicamente necesitan la instalación:

- Java Development Kit (JDK)
- Java Runtime Environment (JRE)

En ambiente linux puede descargarse a través de las instrucciones en consola:

```
- sudo apt-get install sun-java6-jdk
- sudo apt-get install sun-java6-jre
- sudo apt-get install sun-java6-bin
- sudo apt-get install sun-java6-plugin
```

Imagen 25: Comandos para la descarga de JDK en linux

4.3.2 Instalación

Para instalar ambas herramientas puede hacerse por medio de los comandos en consola

```
-sudo apt-get install jlex

- sudo apt-get install cup
```

Imagen 26: Comandos para las Herramientas JLex y Cup

Sección: Herramientas para Análisis 62

ORGANIZACIÓN DE LENGUAJES Y COMPILADORES 1

4.3.3 Configuración

Para configurar las variables de entorno es necesario declararle al sistema la ubicación de los archivos jar, de cada una de las herramientas. Esto por medio de la siguiente línea de instrucción en consola:

```
export
CLASSPATH=.:/usr/share/java/JLex.ja
r:/usr/share/java/cup.jar
```

Imagen 27: Comando para la configuración para el acceso de las librerías

63 Sección: Herramientas para Análisis

ORGANIZACIÓN DE LENGUAJES Y COMPILADORES 1

5 AUTÓMATAS FINITOS

NOMBRE DE CURSO:	Lenguajes Formales y de Programación LABORATORIO
MÓDULO:	Lenguajes y Compiladores Intermedios – 5
OBJETIVOS:	• Definir que es una gramática. • Identificar los componentes de una gramática. • Determinar la estructura para la construcción de gramáticas regulares. • Construir de árboles de derivación

Sección: Autómatas Finitos 64

Figura 21. Páginas 65-68 Libro "Compiladores Intermedio"

5.1 Autómatas finitos

También llamado máquina de estado finito, es un modelo matemático que recibe una cadena de símbolos de un alfabeto y determina la cadena pertenece al lenguaje que el autómata reconoce. Es una máquina que analiza cadenas de entrada bajo cierto lenguaje regular, puede reconocer o traducir una cadena de símbolos

La cadena se divide en Celdas de memoria para almacenar, los estados son la memoria de los autómatas. Cuando finaliza la cadena, acepta o rechaza la cadena, dependiendo de procesamiento.

Imagen 28: Proceso de Análisis de un Autómata

Los autómatas finitos se representan a través de diagramas de Moore, donde cada estado representa una etapa y por medio de un símbolo de la cadena de entrada se traslada a otro estado, formando un arco. Si logra alcanzar un estado de aceptación, se reconoce, de lo contrario se rechaza la cadena.

Inspecciona cada símbolo de la cadena comparán-dola con una tabla de transiciones y el estado actual en que se encuen-tra, si es reconocido cambia su estado y se mueve una celda hacia la derecha, hasta que reconoce todos los elementos de la cadena.

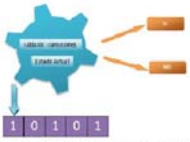


Imagen 29: Análisis de cadena por medio de un Autómata

5.1.1 Tipos de Autómatas Finitos

Autómatas Finitos Deterministicos

- No permite ambigüedades
- Cada arco posee un solo símbolo de transición entre un estado y otro.

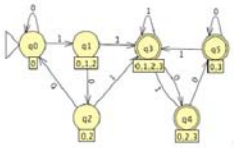


Imagen 30: Ejemplo de Autómata Finitos Deterministicos

Autómatas Finitos No Deterministicos

Los estados pueden tener una o más transiciones con diferentes símbolos del lenguaje.

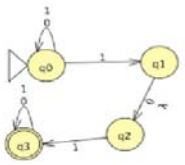


Imagen 31: Ejemplo de Autómata Finito No Determinístico

5.1.2 Diseño de Autómatas Finitos

Pasos para el diseño de un Autómata Finito

- Establecer un conjunto de estados que "memoricen" condiciones importantes
- Establecer correctamente cual es el estado inicial y los estados de aceptación
- Establecer las transiciones que permiten pasar de un estado a otro

Errores al momento de construir cualquier autómata

- Aceptar cadenas que no se debería aceptar, pues no están definidas.
- No aceptar todas las cadenas que debe de aceptar.

5.1.3 Construcción de Autómata

Un autómata puede partir de una expresión regular, una tabla de transiciones o de una gramática. Además de que existen métodos para la escritura y transformación de los autómatas.

5.2 Autómatas finitos Deterministicos

También es llamado AFD o en sus siglas en inglés DFA Deterministic finite automata. Puede detectar símbolos conforme llegan y se basa en el estado actual y el símbolo recibido, ejecutar una transición.

Estos tienen un número finito de estados además los Autómata Finito Determinista no debe tener ambigüedades. El diagrama de transición cada arco posee solo un símbolo del alfabeto, al salir de un estado.

5.2.1 Definición

Se define por medio del quinteto (Q, Σ, q0, A, δ) donde:

- Q es un conjunto finito (de estados)
- Σ es un alfabeto (finito) de símbolos
- $q0 \in Q$ (El estado inicial)
- $A \subset Q$ (El conjunto de estados aceptores)
- δ es una función de transición Q x δ a Q

Figura 22. Páginas 69-72 Libro "Compiladores Intermedio"

5.2.2 Función de Transición

Para todo q ∈ Q & a ∈ Σ, δ(q, a) = p es decir, existe un estado p al que se mueve al estar en el estado q, y lee el carácter a. Una transición: δ(q, a, q0) ∈ Q normalmente se describe q →q0

5.2.3 Algoritmo

Inicia en el estado Inicial, y en el primer símbolo de entrada
q = estado actual
s = símbolo actual en la cinta
Mientras (st = blanco) haga
q = δ (q, s)
s = siguiente símbolo a la derecha en la cinta entonces
si q_n = Estado Final
Acepta
Sino
No Acepta

5.2.4 Notación General DFA

Imagen 32: Notación General DFA

5.2.5 Ejemplos de transformación

Transformación ER - DFA. Ejemplo 1

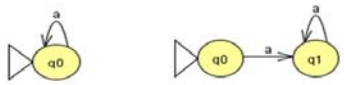


Imagen 33: Ejemplo DFA a* y a+

Transformación ER - DFA. Ejemplo 2

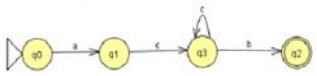


Imagen 34: Ejemplo DFA ac*b

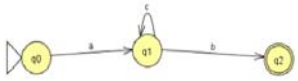


Imagen 35: Ejemplo DFA ac*b

Transformación ER - DFA. Ejemplo 3

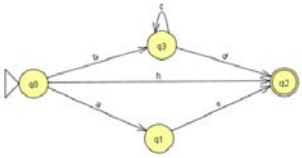


Imagen 36: Ejemplo DFA bc*d | h | ae

Transformación ER - DFA. Ejemplo 4

A partir de una expresión regular: bb|(a(ba)*)* Iniciando por: **bb**|(a(ba)*)*. Tomando en cuenta que al completar la cadena **bb** puede llegar a unestado final.

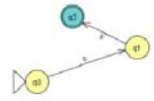


Imagen 37: Ejemplo DFA bb | (a(ba)*)* parte 1

Tomando en cuenta la utilización de paréntesis se define la secuencia **(ba)***. Por estar contenida en una cerradura, indica que puede o no venir, por lo que q_1 es un estado de aceptación .Por Ejemplo bb|(a(ba)*)* de define como:

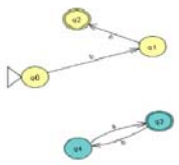


Imagen 38: Ejemplo DFA bb | (a(ba)*)* parte 2

Figura 23. Páginas 73-76 Libro "Compiladores Intermedio"

Para finalizar, la secuencia **(a (..))***, parte del estado inicial con una transición a, y dando la recursividad en q3. Pero tomando en cuenta que puede recibir cadena vacía q0 puede ser estado final. bb|**(a(ba)*)***

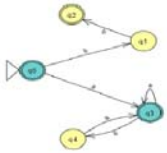


Imagen 39: Ejemplo DFA bb | (a(ba)*)* parte 3

NFA – Tabla de Transición

A partir del DFA se obtiene la siguiente tabla de transición

Q \ Σ	a	b
0	3	1
1	2	-
2	-	-
3	3	4
4	3	-

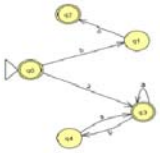


Imagen 40: DFA generado a partir de una Tabla de Transición

A partir de la definición {Q, Σ, q0, A, δ} donde

:

$Q = \{0,1,2,3\}$

$\Sigma = \{a, b\}$

$q_0 = 0$

$A = \{0, 1, 2\}$

Q \ Σ	a	b
0	0	1
1	0	2
2	0	3
3	3	3

Imagen 41: Tabla de Transición para la Construcción de DFA

5.2.5.1.1 Transformación

Parte del estado inicial $q_0 = 0$ Y se mueve con a al estado q_0 y con b al estado q_1

Σ \ Q	a	b
0	0	1
1	0	2
2	0	3
3	3	3

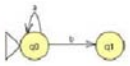


Imagen 42: Construcción de DFA a partir de TT parte 1

Del estado q_1 con a pasa al estado 0 y con b al estado q_2

Σ \ Q	a	b
0	0	1
1	0	2
2	0	3
3	3	3

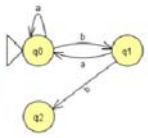


Imagen 43: Construcción de DFA a partir de TT parte 2

Del estado q_2 con a pasa al estado 0 y con b al estado q_3

Σ \ Q	a	b
0	0	1
1	0	2
2	0	3
3	3	3

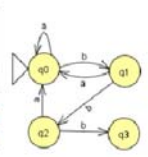


Imagen 44: Construcción de DFA a partir de TT parte 3

Del estado q_3 con a pasa al estado 0 y con b al estado q_3

Σ \ Q	a	b
0	0	1
1	0	2
2	0	3
3	3	3

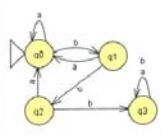


Imagen 45: Construcción de DFA a partir de TT parte 4

Los estados de aceptación son q_0, q_1 y q_2

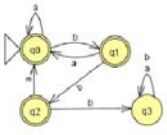


Imagen 46: DFA resultante

5.1 Generación de DFA a partir de Definiciones LR

Al ser los DFA reconocedores de LR, puede generarse un DFA a partir de definiciones de lenguaje como:

- Cadenas terminadas en 00
- Cadenas con dos "unos" consecutivos

Figura 24. Páginas 77-80 Libro “Compiladores Intermedio”

Donde el alfabeto = {0,1}

6.1.1 Transformación

Cadenas terminadas en 00

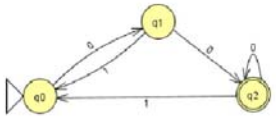


Imagen 47: DFA cadenas terminadas en 00

Cadenas con dos "unos" consecutivos

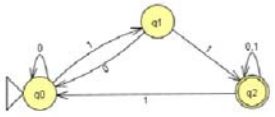


Imagen 48_ DFA Cadenas con dos "unos" consecutivos

5.2 Autómatas finitos no Deterministicos

También es llamado AFN o en sus siglas en inglés NFA *Non-Deterministic Finite Automata*. Es decir, el autómata acepta aquellos lenguajes en los que al procesar la cadena se obtenga un conjunto en el que haya al menos un estado de Aceptación.

Los Autómata Finito No Determinista puede tener ambigüedades. Los estados pueden tener una o más transiciones con diferentes símbolos del lenguaje. El autómata acepta una palabra si existe al menos un camino desde el estado q_0 a un estado final con la palabra de entrada.

5.2.1 Definición

Se define por medio del quinteto (Q, Σ, q0, A, δ) donde:

- Q es un conjunto finito (de estados)
- Σ es un alfabeto (finito) de símbolos
- q0 ∈ Q (El estado inicial)
- A ⊆ Q (El conjunto de estados aceptores)
- δ es una función de transición $\delta^*: Q \times \Sigma^* \rightarrow 2^Q$

5.2.2 Función de Transición

Se define como sigue: $\delta^*: Q \times \Sigma^* \rightarrow 2^Q$

- Para todo q ∈ Q, $\delta^*(q,\Lambda) = \{q\}$
- Para todo q ∈ Q, y ∈ Σ* & a ∈ Σ

Tomando en cuenta que 2^Q es el conjunto de potencias de Q, el conjunto de todos los subconjuntos de Q

5.2.3 Construcción de NFA

- Puede partir desde una tabla de transiciones, generando un diagrama de Moore.
- A través de una expresión regular, generando un diagrama de Moore.
- Por medio de una gramática regular.

5.2.4 Tabla de Transición – NFA Ejemplo 1

A partir de la definición (Q, Σ, q0, A, δ) donde:

- Q= {0,1,2,3}
- Σ ={a, b}
- q_0 =0
- A={3}

Genere el Diagrama de Moore que representa al NFA

Q \ Σ	a	b
0	1,2	-
1	-	1,3
2	2	3
3	3	-

Tabla 3: Tabla de Transiciones Ejemplo 1

Construcción NFA

Parte del estado inicial q_0 =0 y se mueve con **a** al estado q_1 y q_2 y con **b** no tiene movimiento.

Q \ Σ	a	b
0	1,2	-
1	-	1,3
2	2	3
3	3	-

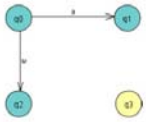


Imagen 49: Construcción de NFA parte 1

Del estado q_1 con **b** pasa al estado q_3 y q_4, con **a** no hay movimiento

Q \ Σ	a	b
0	1,2	-
1	-	1,3
2	2	3
3	3	-

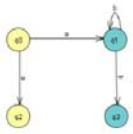


Imagen 50: Construcción de NFA parte 2

Del estado q_2 con **a** pasa al estado q_3 y con **b** al estado q_4

Q \ Σ	a	b
0	1,2	-
1	-	1,3
2	2	3
3	3	-

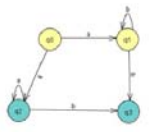
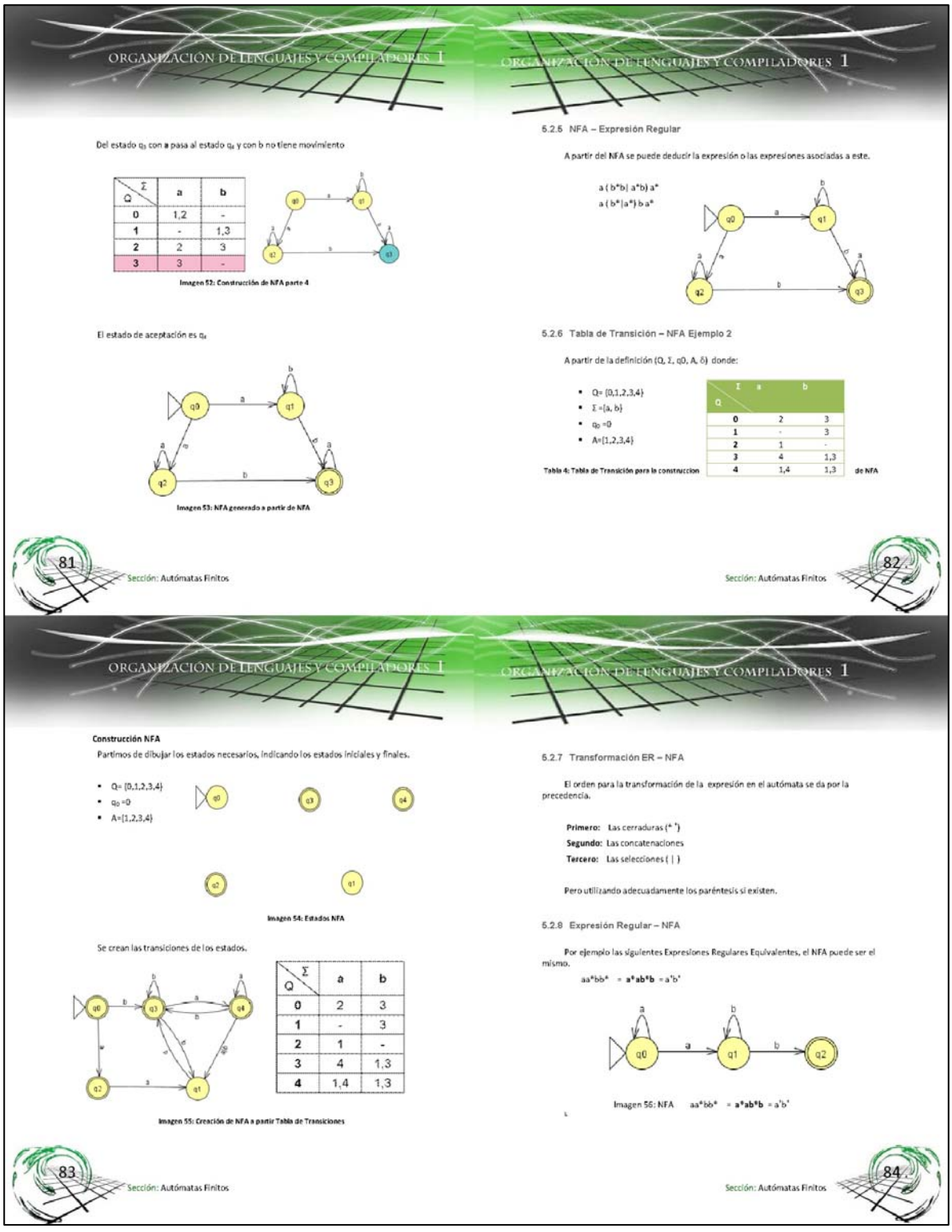


Imagen 51: Construcción de NFA parte 3

Figura 25. Páginas 81-84 Libro "Compiladores Intermedio"



Del estado q_3 con a pasa al estado q_4 y con b no tiene movimiento

Σ / Q	a	b
0	1,2	-
1	-	1,3
2	2	3
3	3	-

Imagen 52: Construcción de NFA parte 4

El estado de aceptación es q_4

Imagen 53: NFA generado a partir de NFA

5.2.5 NFA – Expresión Regular

A partir del NFA se puede deducir la expresión o las expresiones asociadas a este.

a (b*b| a*b) a*
a (b*|a*) b a*

5.2.6 Tabla de Transición – NFA Ejemplo 2

A partir de la definición (Q, Σ, q0, A, δ) donde:

- Q= {0,1,2,3,4}
- Σ ={a, b}
- q_0 =0
- A={1,2,3,4}

Σ / Q	a	b
0	2	3
1	-	3
2	1	-
3	4	1,3
4	1,4	1,3

Tabla 4: Tabla de Transición para la construccion de NFA

Construcción NFA

Partimos de dibujar los estados necesarios, indicando los estados iniciales y finales.

- Q= {0,1,2,3,4}
- q_0 =0
- A={1,2,3,4}

Imagen 54: Estados NFA

Se crean las transiciones de los estados.

Σ / Q	a	b
0	2	3
1	-	3
2	1	-
3	4	1,3
4	1,4	1,3

Imagen 55: Creación de NFA a partir Tabla de Transiciones

5.2.7 Transformación ER – NFA

El orden para la transformación de la expresión en el autómata se da por la precedencia.

Primero: Las cerraduras (* +)
Segundo: Las concatenaciones
Tercero: Las selecciones (|)

Pero utilizando adecuadamente los paréntesis si existen.

5.2.8 Expresión Regular – NFA

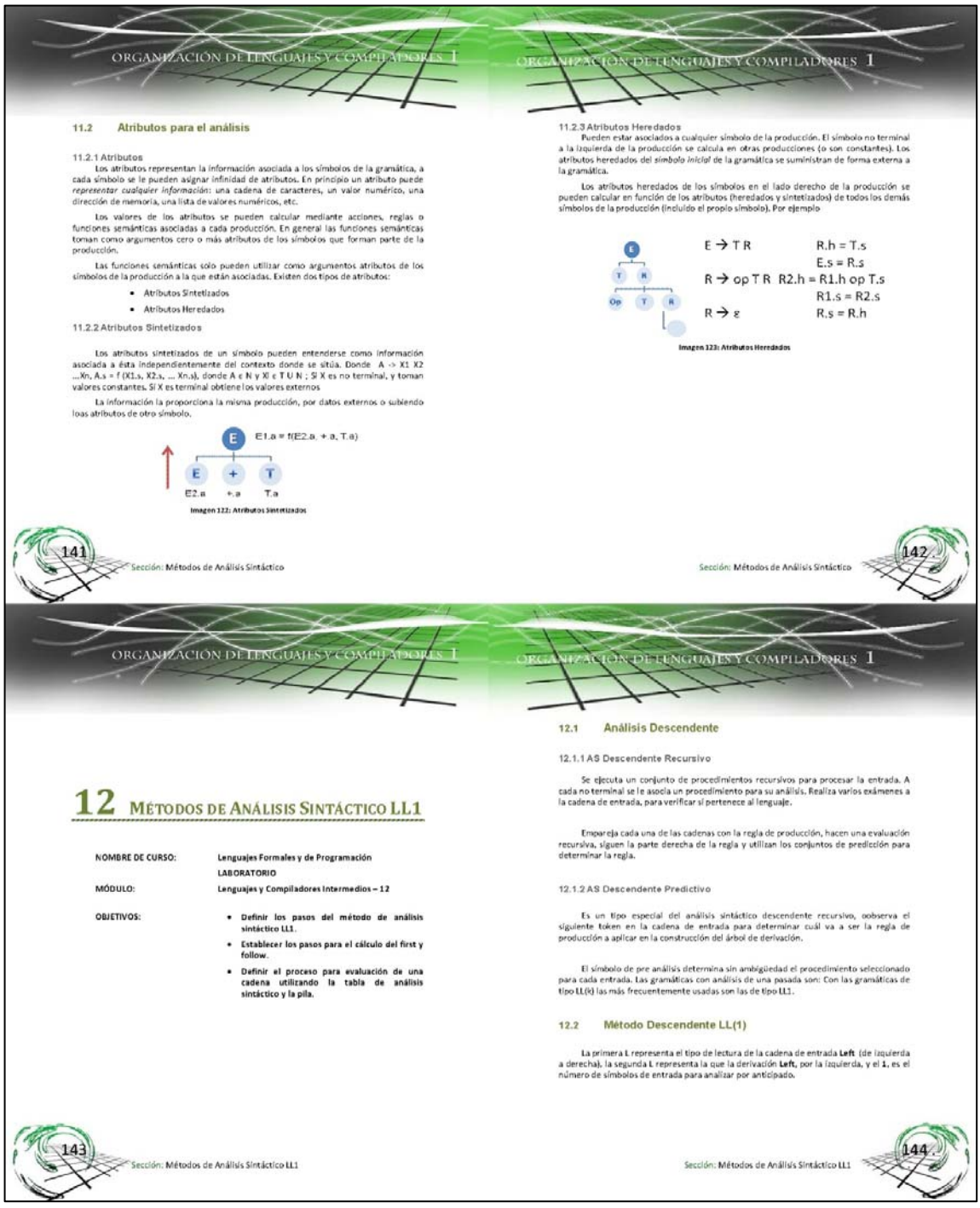
Por ejemplo las siguientes Expresiones Regulares Equivalentes, el NFA puede ser el mismo.

aa*bb* = **a*ab*b** = a'b'

Imagen 56: NFA aa*bb* = **a*ab*b** = a'b'

7.3 Métodos de análisis sintáctico LL1

Figura 26. Páginas 141-144 Libro "Compiladores Intermedio"



ORGANIZACIÓN DE LENGUAJES Y COMPILADORES 1

11.2 Atributos para el análisis

11.2.1 Atributos

Los atributos representan la información asociada a los símbolos de la gramática, a cada símbolo se le pueden asignar infinidad de atributos. En principio un atributo puede *representar cualquier información*: una cadena de caracteres, un valor numérico, una dirección de memoria, una lista de valores numéricos, etc.

Los valores de los atributos se pueden calcular mediante acciones, reglas o funciones semánticas asociadas a cada producción. En general las funciones semánticas toman como argumentos cero o más atributos de los símbolos que forman parte de la producción.

Las funciones semánticas solo pueden utilizar como argumentos atributos de los símbolos de la producción a la que están asociadas. Existen dos tipos de atributos:

- Atributos Sintetizados
- Atributos Heredados

11.2.2 Atributos Sintetizados

Los atributos sintetizados de un símbolo pueden entenderse como información asociada a ésta independientemente del contexto donde se sitúa. Donde A -> X1 X2 ...Xn, A.s = f (X1.s, X2.s, ... Xn.s), donde A ∈ N y Xi ∈ T U N ; Si X es no terminal, y toman valores constantes. Si X es terminal obtiene los valores externos

La información la proporciona la misma producción, por datos externos o subiendo loas atributos de otro símbolo.

Imagen 122: Atributos Sintetizados

ORGANIZACIÓN DE LENGUAJES Y COMPILADORES 1

11.2.3 Atributos Heredados

Pueden estar asociados a cualquier símbolo de la producción. El símbolo no terminal a la izquierda de la producción se calcula en otras producciones (o son constantes). Los atributos heredados del *símbolo inicial* de la gramática se suministran de forma externa a la gramática.

Los atributos heredados de los símbolos en el lado derecho de la producción se pueden calcular en función de los atributos (heredados y sintetizados) de todos los demás símbolos de la producción (incluido el propio símbolo). Por ejemplo

Imagen 123: Atributos Heredados

ORGANIZACIÓN DE LENGUAJES Y COMPILADORES 1

12 MÉTODOS DE ANÁLISIS SINTÁCTICO LL1

NOMBRE DE CURSO:	**Lenguajes Formales y de Programación LABORATORIO**
MÓDULO:	**Lenguajes y Compiladores Intermedios – 12**
OBJETIVOS:	• **Definir los pasos del método de análisis sintáctico LL1.** • **Establecer los pasos para el cálculo del first y follow.** • **Definir el proceso para evaluación de una cadena utilizando la tabla de análisis sintáctico y la pila.**

ORGANIZACIÓN DE LENGUAJES Y COMPILADORES 1

12.1 Análisis Descendente

12.1.1 AS Descendente Recursivo

Se ejecuta un conjunto de procedimientos recursivos para procesar la entrada. A cada no terminal se le asocia un procedimiento para su análisis. Realiza varios exámenes a la cadena de entrada, para verificar si pertenece al lenguaje.

Empareja cada una de las cadenas con la regla de producción, hacen una evaluación recursiva, siguen la parte derecha de la regla y utilizan los conjuntos de predicción para determinar la regla.

12.1.2 AS Descendente Predictivo

Es un tipo especial del análisis sintáctico descendente recursivo, oobserva el siguiente token en la cadena de entrada para determinar cuál va a ser la regla de producción a aplicar en la construcción del árbol de derivación.

El símbolo de pre análisis determina sin ambigüedad el procedimiento seleccionado para cada entrada. Las gramáticas con análisis de una pasada son: Con las gramáticas de tipo LL(k) las más frecuentemente usadas son las de tipo LL1.

12.2 Método Descendente LL(1)

La primera L representa el tipo de lectura de la cadena de entrada **Left** (de izquierda a derecha), la segunda L representa la que la derivación **Left**, por la izquierda, y el **1**, es el número de símbolos de entrada para analizar por anticipado.

Figura 27. Páginas 145-148 Libro "Compiladores Intermedio"

Buffer de Entrada

Cadena de entrada a analizar, finaliza con el carácter $

Pila

Símbolos gramaticales que se van utilizando

Tabla de Análisis Sintáctico

Matriz bidimensional que sirve para el análisis

Cadena de Salida

Cadena de Salida posterior al análisis

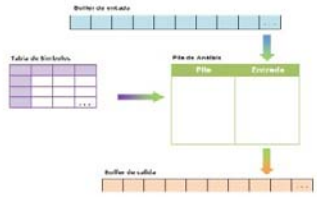


Imagen 124: Proceso de Análisis LL(1)

12.2.1 Pasos para el Método LL(1)

- Escribir adecuadamente la gramática
- Calcular el First y el Follow
- Construir la tabla de Análisis Sintáctico
- Hacer el análisis de sintáctico por medio de la pila y la tabla de análisis

Escribir adecuadamente la gramática

Para poder utilizar un analizador descendente no recursivo la gramática debe cumplir con:

- No debe tener ambigüedad
- No debe ser recursiva por la izquierda
- Debe estar factorizada

Calcular el First / Primero

Símbolo	First/Primero
Si X es terminal	First(x) = {x}
Si X → ε producción	Añadir ε al First(x)
Si X es No Terminal y X →YZW	1. Si Y→ ε añadir el First(Z) 2. Si Z → ε añadir el First(W) 3. Si todos generan ε entonces añadir →ε

Tabla 6: Cálculo del First / Primero

Calcular el Follow / Siguiente

Símbolo	Follow/Siguiente
Si B es símbolo inicial	Follow (B) = { $}
Si A → α B M Producción	1. Follow (B) = First (M) excepto ε. 2. Si el First(M) contiene ε entonces añadir el Follow(A) a Follow (B)
Si A → B Producción	Añadir el Follow(A) a Follow(B)

Tabla 7: Cálculo del Follow / Siguiente

Construir la tabla de Análisis Sintáctico

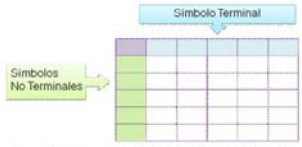


Imagen 125: Construcción de Tabla de Análisis Sintácticos (cabeceras)

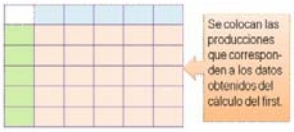


Imagen 126: Construcción de Tabla de Análisis Sintáctico

Hacer el análisis de sintáctico por medio de la pila y la tabla de análisis

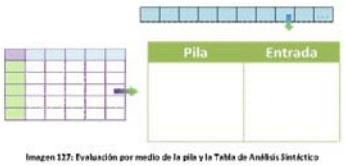
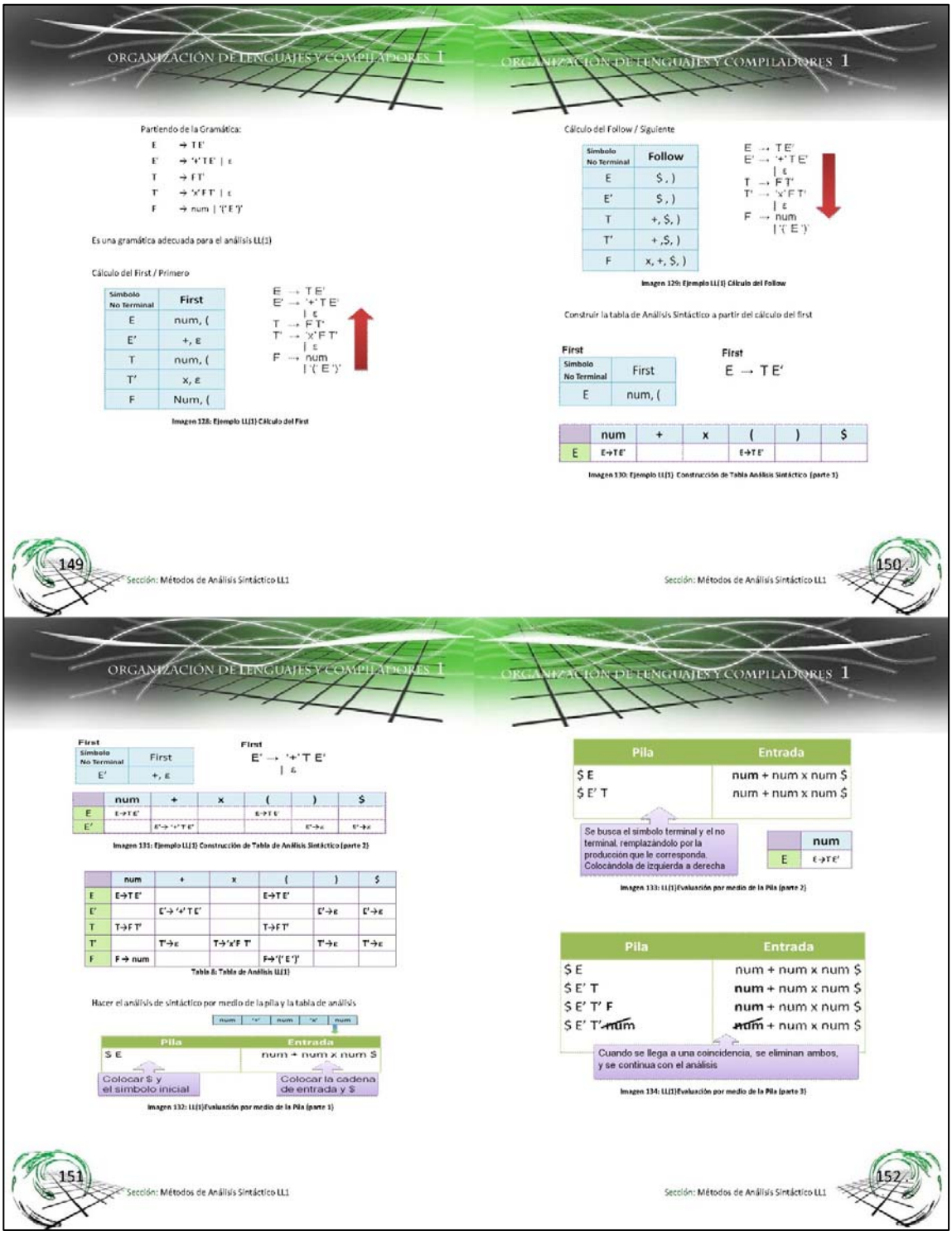


Imagen 127: Evaluación por medio de la pila y la Tabla de Análisis Sintáctico

12.2.2 Ejemplo LL(1)

Figura 28. Páginas 149-152 Libro “Compiladores Intermedio”



ORGANIZACIÓN DE LENGUAJES Y COMPILADORES 1

Partiendo de la Gramática:

E → T E'
E' → '+' T E' | ε
T → F T'
T' → 'x' F T' | ε
F → num | '(' E ')'

Es una gramática adecuada para el análisis LL(1)

Cálculo del First / Primero

Símbolo No Terminal	First
E	num, (
E'	+, ε
T	num, (
T'	x, ε
F	Num, (

Imagen 128: Ejemplo LL(1) Cálculo del First

149 Sección: Métodos de Análisis Sintáctico LL1

ORGANIZACIÓN DE LENGUAJES Y COMPILADORES 1

Cálculo del Follow / Siguiente

Símbolo No Terminal	Follow
E	$,)
E'	$,)
T	+, $,)
T'	+, $,)
F	x, +, $,)

Imagen 129: Ejemplo LL(1) Cálculo del Follow

Construir la tabla de Análisis Sintáctico a partir del cálculo del first

First

Símbolo No Terminal	First
E	num, (

First
E → T E'

	num	+	x	(	)	$
E	E→T E'			E→T E'		

Imagen 130: Ejemplo LL(1) Construcción de Tabla Análisis Sintáctico (parte 1)

150 Sección: Métodos de Análisis Sintáctico LL1

ORGANIZACIÓN DE LENGUAJES Y COMPILADORES 1

First

Símbolo No Terminal	First
E'	+, ε

First
E' → '+' T E' | ε

	num	+	x	(	)	$
E	E→T E'			E→T E'		
E'		E'→ '+' T E'			E'→ε	E'→ε

Imagen 131: Ejemplo LL(1) Construcción de Tabla de Análisis Sintáctico (parte 2)

	num	+	x	(	)	$
E	E→T E'			E→T E'		
E'		E'→ '+' T E'			E'→ε	E'→ε
T	T→F T'			T→F T'		
T'		T'→ε	T→'x' F T'		T'→ε	T'→ε
F	F→ num			F→'(' E ')'		

Tabla 8: Tabla de Análisis LL(1)

Hacer el análisis de sintáctico por medio de la pila y la tabla de análisis

num + num x num $

Pila	Entrada
$ E	num + num x num $

Colocar $ y el símbolo inicial

Colocar la cadena de entrada y $

Imagen 132: LL(1)Evaluación por medio de la Pila (parte 1)

151 Sección: Métodos de Análisis Sintáctico LL1

ORGANIZACIÓN DE LENGUAJES Y COMPILADORES 1

Pila	Entrada
$ E	num + num x num $
$ E' T	num + num x num $

Se busca el símbolo terminal y el no terminal, remplazándolo por la producción que le corresponda. Colocándola de izquierda a derecha

	num
E	E→T E'

Imagen 133: LL(1)Evaluación por medio de la Pila (parte 2)

Pila	Entrada
$ E	num + num x num $
$ E' T	num + num x num $
$ E' T' F	num + num x num $
$ E' T' num	num + num x num $

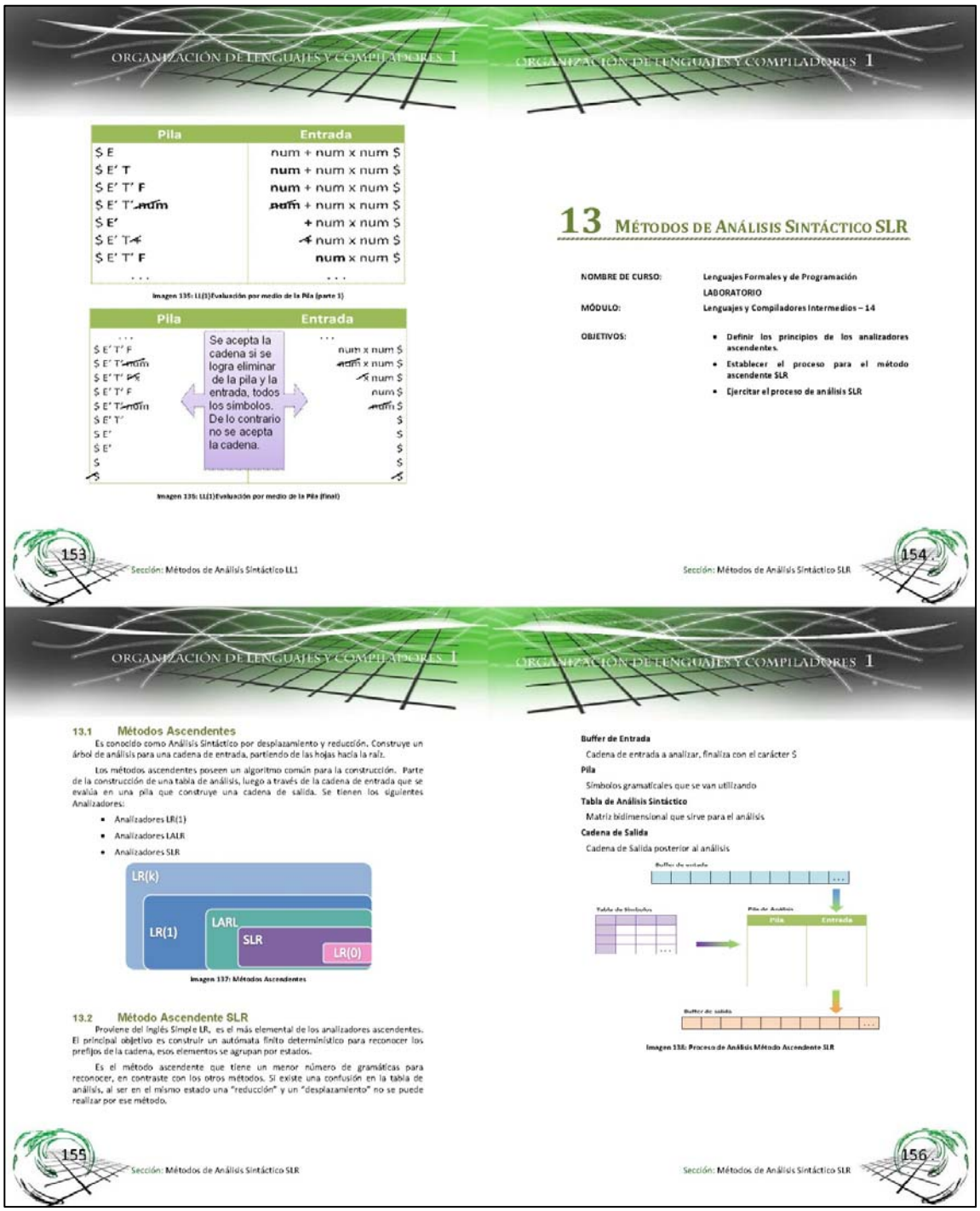
Cuando se llega a una coincidencia, se eliminan ambos, y se continua con el análisis

Imagen 134: LL(1)Evaluación por medio de la Pila (parte 3)

152 Sección: Métodos de Análisis Sintáctico LL1

7.4 Métodos de análisis sintáctico SLR

Figura 29. Páginas 153-156 Libro "Compiladores Intermedio"



ORGANIZACIÓN DE LENGUAJES Y COMPILADORES 1

Pila	Entrada
$ E	num + num x num $
$ E' T	num + num x num $
$ E' T' F	num + num x num $
$ E' T' num	num + num x num $
$ E'	+ num x num $
$ E' T +	+ num x num $
$ E' T' F	num x num $
...	...

Imagen 135: LL(1)Evaluación por medio de la Pila (parte 1)

Pila	Entrada
...	...
$ E' T' F	num x num $
$ E' T' num	num x num $
$ E' T' *	* num $
$ E' T' F	num $
$ E' T' num	num $
$ E' T'	$
$ E'	$
$ E'	$
$	$
$	$

Se acepta la cadena si se logra eliminar de la pila y la entrada, todos los simbolos. De lo contrario no se acepta la cadena.

Imagen 136: LL(1)Evaluación por medio de la Pila (final)

153 Sección: Métodos de Análisis Sintáctico LL1

ORGANIZACIÓN DE LENGUAJES Y COMPILADORES 1

13 Métodos de Análisis Sintáctico SLR

NOMBRE DE CURSO: Lenguajes Formales y de Programación
LABORATORIO

MÓDULO: Lenguajes y Compiladores Intermedios – 14

OBJETIVOS:

- Definir los principios de los analizadores ascendentes.
- Establecer el proceso para el método ascendente SLR
- Ejercitar el proceso de análisis SLR

154 Sección: Métodos de Análisis Sintáctico SLR

ORGANIZACIÓN DE LENGUAJES Y COMPILADORES 1

13.1 Métodos Ascendentes

Es conocido como Análisis Sintáctico por desplazamiento y reducción. Construye un árbol de análisis para una cadena de entrada, partiendo de las hojas hacia la raíz.

Los métodos ascendentes poseen un algoritmo común para la construcción. Parte de la construcción de una tabla de análisis, luego a través de la cadena de entrada que se evalúa en una pila que construye una cadena de salida. Se tienen los siguientes Analizadores:

- Analizadores LR(1)
- Analizadores LALR
- Analizadores SLR

Imagen 137: Métodos Ascendentes

13.2 Método Ascendente SLR

Proviene del Inglés Simple LR, es el más elemental de los analizadores ascendentes. El principal objetivo es construir un autómata finito determinístico para reconocer los prefijos de la cadena, esos elementos se agrupan por estados.

Es el método ascendente que tiene un menor número de gramáticas para reconocer, en contraste con los otros métodos. Si existe una confusión en la tabla de análisis, al ser en el mismo estado una "reducción" y un "desplazamiento" no se puede realizar por ese método.

155 Sección: Métodos de Análisis Sintáctico SLR

ORGANIZACIÓN DE LENGUAJES Y COMPILADORES 1

Buffer de Entrada
Cadena de entrada a analizar, finaliza con el carácter $

Pila
Símbolos gramaticales que se van utilizando

Tabla de Análisis Sintáctico
Matriz bidimensional que sirve para el análisis

Cadena de Salida
Cadena de Salida posterior al análisis

Imagen 138: Proceso de Análisis Método Ascendente SLR

156 Sección: Métodos de Análisis Sintáctico SLR

8. DOCUMENTACIÓN DE APOYO DEL CURSO DE ORGANIZACIÓN DE LENGUAJES Y COMPILADORES 2

A continuación se presenta el desarrollo de la documentación de apoyo del curso de Organización de Lenguajes y Compiladores 2, el cual se realizó en formato de un libro conteniendo las unidades más importantes impartidas a lo largo del curso

8.1 Datos Generales

NOMBRE DE CURSO:	**Organización de Lenguajes y Compiladores 2 (781)**
PRE- REQUISITOS:	Organización de Lenguajes y Compiladores 1 (777) Estructuras de Datos (772)
POST – REQUISITOS:	Sistemas Operativos 1 (281)
OBJETIVO:	Al finalizar el curso, el estudiante tendrá una base teórica y práctica que permita diseñar e implementar de una manera completa un compilador o intérprete para un lenguaje de las nuevas generaciones, haciendo énfasis en la fase de síntesis.
MÓDULO:	**Lenguajes y Compiladores Avanzado**

Figura 30. Páginas 1-4 Libro “Compiladores Avanzado”

Figura 31. Páginas 5-8 Libro “Compiladores Avanzado”

Prerrequisitos

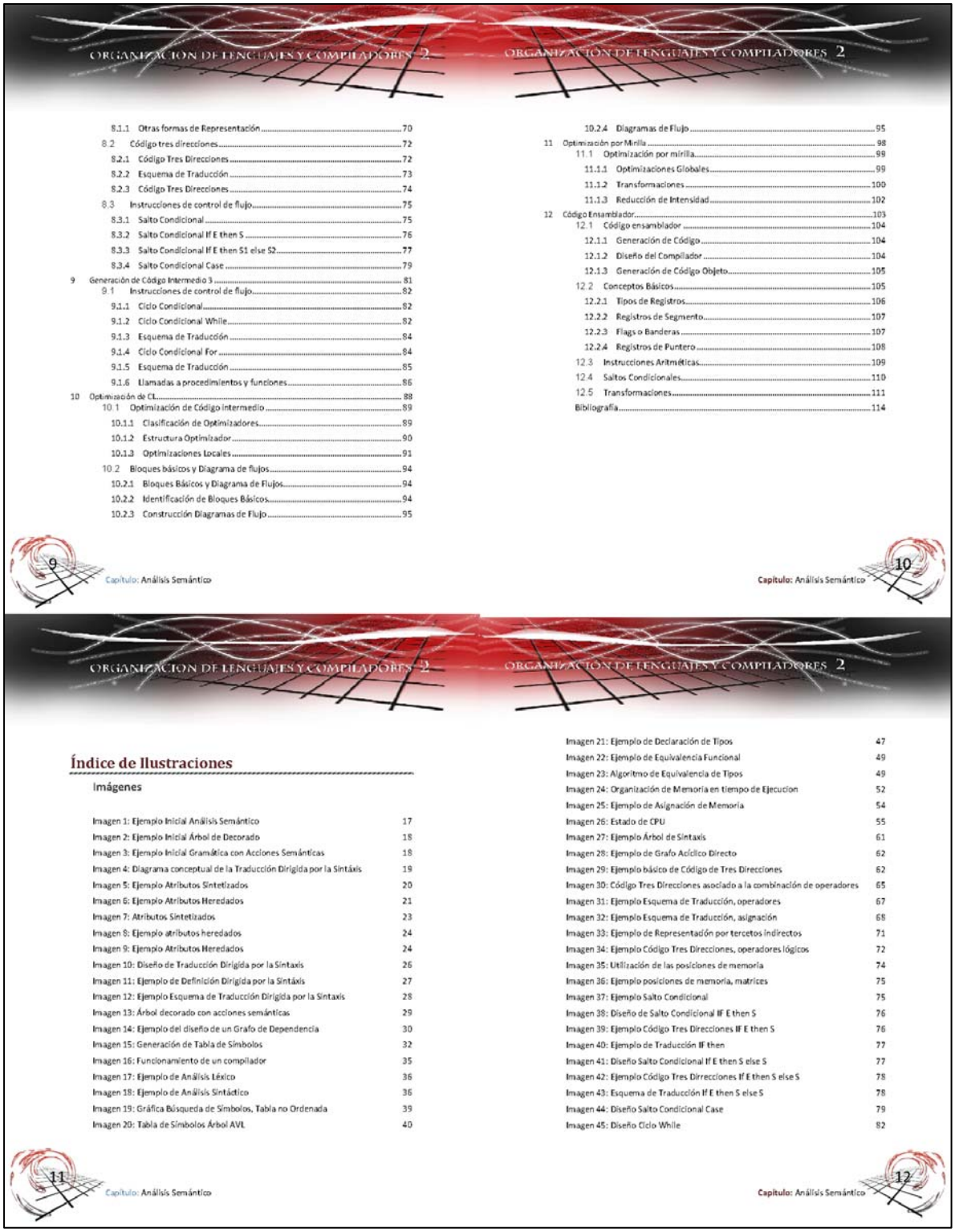
Conocimientos básicos de programación y la utilización de algoritmos para la resolución de problemas.

Conocimientos previos de la utilización de variables, estructuras, clases, procedimientos, funciones, etc.

Conocimientos básicos de lógica computacional.

Índice General

Figura 32. Páginas 9-12 Libro “Compiladores Avanzado”

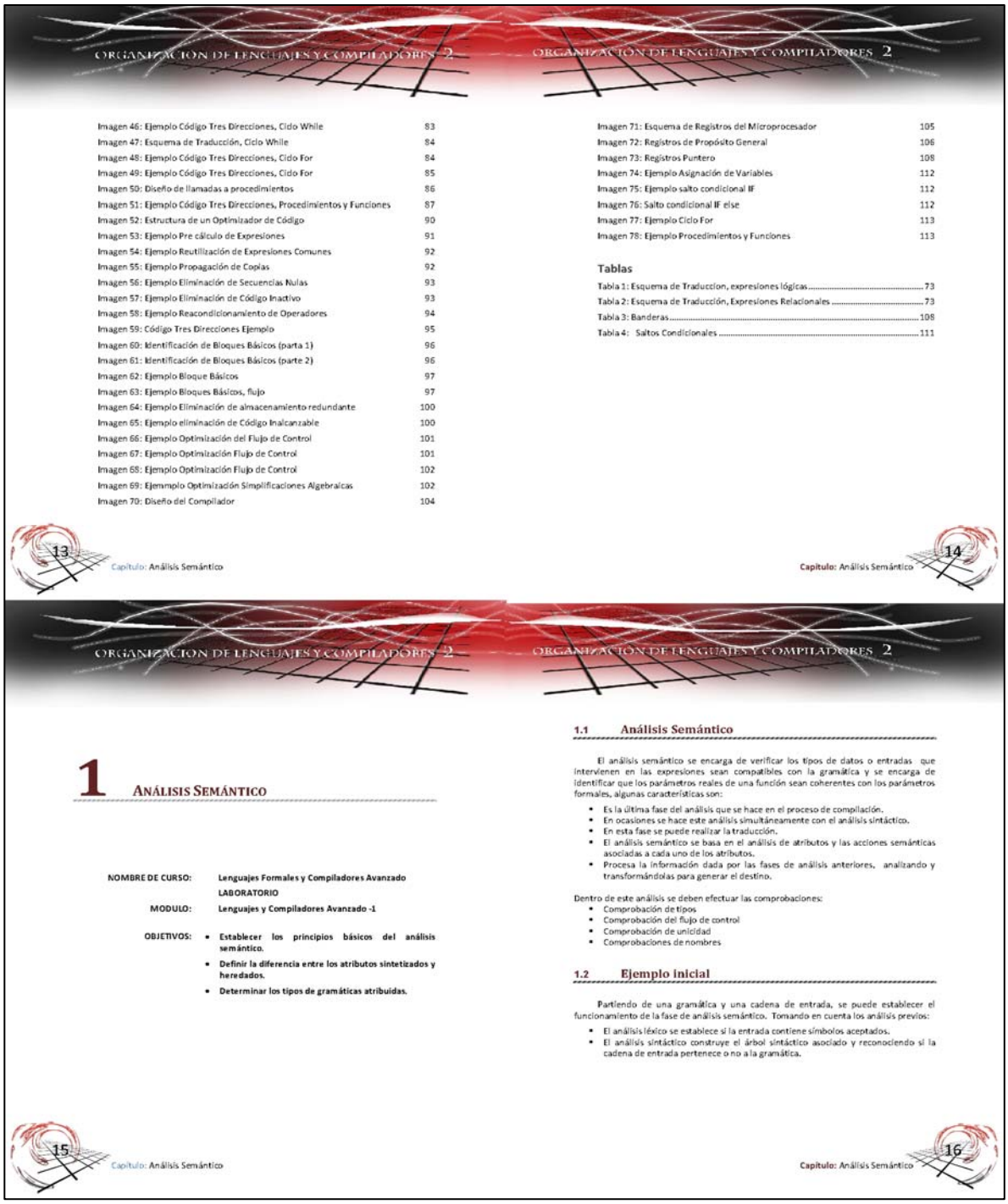


Índice de Ilustraciones

Imágenes

8.2 Análisis semántico

Figura 33. Páginas 13-16 Libro "Compiladores Avanzado"



Tablas

1 ANÁLISIS SEMÁNTICO

NOMBRE DE CURSO: Lenguajes Formales y Compiladores Avanzado
LABORATORIO

MODULO: Lenguajes y Compiladores Avanzado -1

OBJETIVOS:

- **Establecer los principios básicos del análisis semántico.**
- **Definir la diferencia entre los atributos sintetizados y heredados.**
- **Determinar los tipos de gramáticas atribuidas.**

1.1 Análisis Semántico

El análisis semántico se encarga de verificar los tipos de datos o entradas que intervienen en las expresiones sean compatibles con la gramática y se encarga de identificar que los parámetros reales de una función sean coherentes con los parámetros formales, algunas características son:

- Es la última fase del análisis que se hace en el proceso de compilación.
- En ocasiones se hace este análisis simultáneamente con el análisis sintáctico.
- En esta fase se puede realizar la traducción.
- El análisis semántico se basa en el análisis de atributos y las acciones semánticas asociadas a cada uno de los atributos.
- Procesa la información dada por las fases de análisis anteriores, analizando y transformándolas para generar el destino.

Dentro de este análisis se deben efectuar las comprobaciones:

- Comprobación de tipos
- Comprobación del flujo de control
- Comprobación de unicidad
- Comprobaciones de nombres

1.2 Ejemplo inicial

Partiendo de una gramática y una cadena de entrada, se puede establecer el funcionamiento de la fase de análisis semántico. Tomando en cuenta los análisis previos:

- El análisis léxico se establece si la entrada contiene símbolos aceptados.
- El análisis sintáctico construye el árbol sintáctico asociado y reconociendo si la cadena de entrada pertenece o no a la gramática.

Figura 34. Páginas 17-20 Libro "Compiladores Avanzado"

Imagen 1: Ejemplo Inicial Análisis Semántico

Los análisis previos nos determinan un primer paso para el análisis semántico, pero el análisis léxico y la construcción del árbol sintáctico, no quiere decir que sea semánticamente correcto, para ello se debe proceder a decorar el árbol sintáctico asociando valores a los terminales y no terminales que en él aparecen.

Dentro del análisis léxico se debe proceder a asociar una acción léxica que permita retornar el valor al siguiente análisis. Para asociarlo se hace por medio de: **[0-9]+ { yytext(); return NUM; }**. Para el análisis sintáctico podemos proceder a la decoración del árbol sintáctico, propagando los atributos a través de las hojas hacia la raíz. Esto se hace por medio de acciones semánticas.

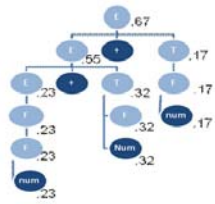


Imagen 2: Ejemplo Inicial Árbol de Decorado

Una acción o regla semántica es un algoritmo que puede acceder a los atributos de los terminales y/o no terminales. Como acción semántica no sólo se puede poner una asignación a un atributo, además puede añadirse código, estas acciones se deben ejecutar cada vez que se reduce sintácticamente por la regla asociada. Las reglas de producción o acciones semánticas asociadas son:

S →	E	{ printf("%d\n", E); }
E1→	E2 + T	{E1 = E2 + T;}
E →	T	{E = T;}
T1→	T2 * F	{T1 = T2 * F;}
T →	F	{T = F;}
F →	(E)	{F = E;}
F →	num	{F = NUM;}

Imagen 3: Ejemplo Inicial Gramática con Acciones Semánticas

Este ejemplo demuestra el proceso que lleva en análisis semántico, más elemental. A continuación se establece la traducción dirigida por la sintaxis, que es el proceso que se utiliza al realizar el análisis semántico.

1.3 Traducción dirigida por la sintaxis

Consiste en asociar atributos a cada símbolo y una acción semántica a una regla de producción, pero no se impone el orden de ejecución de las reglas.

En una primera fase se construye el árbol sintáctico completo, incluyendo los atributos y, luego se ejecutan las acciones semánticas en una secuencia tal que permita el cálculo de todos los atributos de los nodos del árbol. El siguiente diagrama representa el aspecto conceptual de la traducción dirigida por la sintaxis.

Imagen 4: Diagrama conceptual de la Traducción Dirigida por la Sintaxis

Existen dos tipos de traducciones por las cuales se puede hacer al análisis semántico:

- DDS Definición Dirigida por la Sintaxis
- EDT Esquemas de Traducción

1.3.1 Atributos

Los atributos representan cualquier cosa: cadena, número, tipo, posición de memoria, etc. Cada atributo está asociado a un símbolo, pero un símbolo puede tener asociada una infinidad de atributos.

Los valores para los atributos de los terminales son proporcionados generalmente por el analizador léxico, el proceso de calcular los valores de los atributos en los nodos se denomina decorar el árbol de análisis sintáctico.

El valor de un atributo se define mediante la regla semántica asociada a la regla de producción utilizada en ese nodo, existen dos tipos de atributos:

- Sintácticos
- Heredados

Atributos Sintetizados

El valor se calcula únicamente a partir de los valores de los atributo hijos en el árbol de análisis, no importando si estos atributos son sintetizados o heredados, A.a = f(A1.a1,A2.a2, ...,Ak.ak).

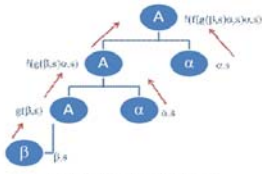


Imagen 5: Ejemplo Atributos Sintetizados

Una gramática con atributos sintetizados:

E → E + T	{ E0.valor := E1.valor + T.valor }
E → T	{ E.valor := T.valor }
T → T * F	{ T0.valor := T1.valor * F.valor }
T → F	{ T.valor := F.valor }
F → (E)	{ F.valor := E.valor }
F → entero	{ F.valor := entero.valor }

Figura 35. Páginas 21-24 Libro "Compiladores Avanzado"

F → ident [F.valor := (ident.texto) }

Atributos Heredados

El valor se calcula a partir de los valores de los atributos que pertenecen al padre o a los hermanos de ese nodo en el árbol de análisis, Al.al = f(A1.a1,A2.a2, ...,Al-1.al-1,Al-1.al-1, ...,Ak.ak)

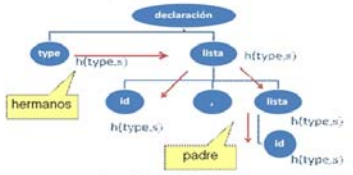


Imagen 6: Ejemplo Atributos Heredados

Una gramática con atributos sintetizados:

Declaracion → Tipo : ListaIDs;
[ListaIDS.tipo := NomTipo.tipo }

ListaIDs → ident
[insertarTDS(ident.texto, ListaIDs.tipo) }

ListaIDs → ListaIDs, ident
[ListaIDs1.tipo := ListaIDs0.tipo
insertarTDS(ident.texto, ListaIDs0.tipo) }

NomTipo → float [NomTipo.tipo := "real" }

NomTipo → int [NomTipo.tipo := "entero" }

1.3.2 Árboles con Atributos

Los árboles decorados que tienen solamente atributos sintetizados se dice que son árboles sintetizados, si el árbol contiene uno o más atributos heredados se denominan como un árbol heredado. Los árboles decorados con acciones sintácticas se pueden clasificar según la estructura de sus acciones, estos se clasifican en:

- Árbol de Sintaxis Concreta
- Árbol de Sintaxis Abstracta

Árboles de Sintaxis Concreta

Es un árbol que representa la estructura sintáctica de una cadena identificada en una gramática formal, un programa que produce tales árboles se llama parser.

El árbol de sintaxis concreta puede ser generado para sentencias en lenguajes naturales (PLN, procesado de lenguaje natural), o para el procesamiento de lenguajes de programación. Las acciones semánticas consisten en la asignación y funciones básicas de programación.

Árbol de Sintaxis Abstracta

Es un árbol finito, etiquetado, dirigido, donde los nodos internos son etiquetados por operadores, y los nodos hoja representan los operandos de estos, así, las hojas son operadores NULL y representan solamente variables o constantes.

Se utiliza en un parser como un intermedio entre un árbol y una estructura de datos, que permite una representación interna de un programa para un compilador o intérprete mientras esta siendo optimizado y se está realizando la generación de código.

Las acciones semánticas consisten en la asignación y el uso de clases elaboradas y diseñadas para un manejo abstracto de los datos, por lo general se utilizan arboles, tablas de hash, etc.

1.4 Gramáticas Atribuidas

Las Gramáticas Atribuidas se clasifican según el tipo de atributos que tienen asociadas. Estas se pueden clasificar en:

- Gramáticas S-Atribuidas
- Gramáticas L-Atribuidas
- Gramáticas LC-Atribuidas

1.4.1 Gramáticas S-Atribuidas

Se puede decorar el árbol sintáctico de forma ascendente: desde las hojas hasta la raíz, este tipo de gramáticas utiliza únicamente atributos sintetizados.

Las definiciones S-atribuidas pueden transformarse en esquemas de traducción sin más que poner todas las acciones semánticas al final de las reglas de producción. Un ejemplo de gramática S-Atribuida es el siguiente tipo de gramática.

Producción	*Acción Semántica*
S → E	{ Imprimir (E.valor) }
E → T	{ E.valor = T.valor }
E → E1 + T	{ E.valor = E1.valor + T.valor }
T → F	{ T.valor = F.valor }
T → T1 * F	{ T.valor = T1.valor * F.valor }
F → (E)	{ F.valor = E.valor }
F → número	{ F.valor = número.valor_léxico }
F → identificador	{ F.valor = identificador. valor }

Imagen 7: Atributos Sintetizados

1.4.2 Gramáticas L-Atribuida

Puede representarse en un árbol sintáctico decorado con atributos heredados y sintetizados, las acciones semánticas pueden colocarse en cualquier posición del lado derecho de la producción, no puede ser utilizado en esquemas de traducción. Un ejemplo de gramática S-Atribuida es el siguiente tipo de gramática.

Producción	*Acción semántica*
D → T L ;	L.h = T.tipo
T → int	T.tipo = entero
T → float	T.tipo = real
L → id	Añadir(id.TS,L.h)
L → L1 , id	L1.h = L.h Añadir(id.TS,L.h)

Atributo Sintetizado
Atributo Heredado

Imagen 8: Ejemplo atributos heredados

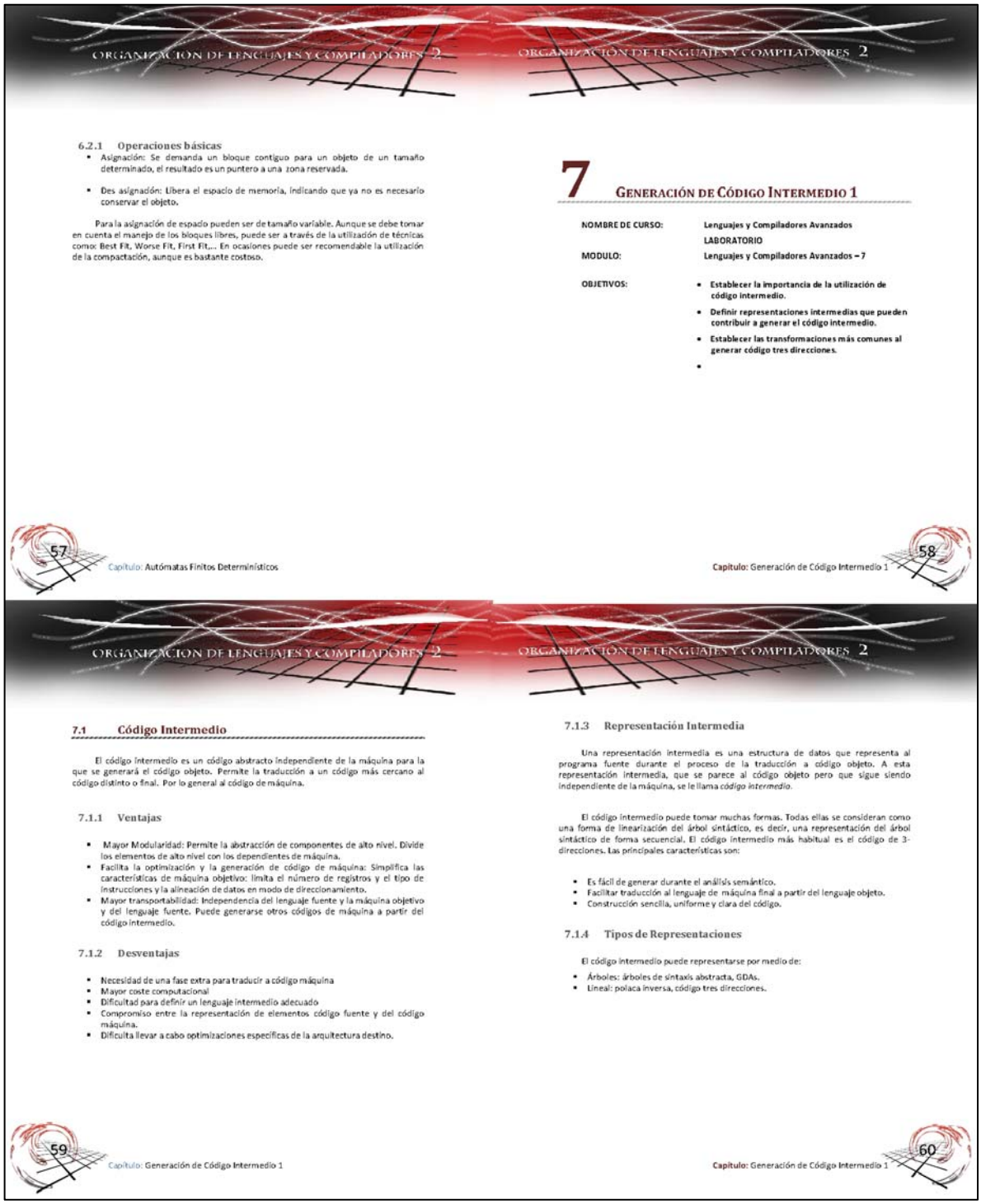
Las acciones semánticas pueden colocarse en cualquier posición del lado derecho de la producción.

D → T { L.h = T.tipo } L ;
T → int { T.tipo = entero }
T → float { T.tipo = real }
L → id { Añadir(id.TS,L.h) }
L → L1 {L1.h = L.h} , id { Añadir(id.TS,L.h) }

Imagen 9: Ejemplo Atributos Heredados

8.3 Generación de código intermedio 1

Figura 36. Páginas 57-60 Libro "Compiladores Avanzado"



ORGANIZACIÓN DE LENGUAJES Y COMPILADORES 2

6.2.1 Operaciones básicas

- Asignación: Se demanda un bloque contiguo para un objeto de un tamaño determinado, el resultado es un puntero a una zona reservada.
- Des asignadón: Libera el espacio de memoria, indicando que ya no es necesario conservar el objeto.

Para la asignación de espado pueden ser de tamaño variable. Aunque se debe tomar en cuenta el manejo de los bloques libres, puede ser a través de la utilizadón de técnicas como: Best Fit, Worse Fit, First Fit,... En ocasiones puede ser recomendable la utilización de la compactación, aunque es bastante costoso.

57 Capítulo: Autómatas Finitos Determinísticos

ORGANIZACIÓN DE LENGUAJES Y COMPILADORES 2

7 GENERACIÓN DE CÓDIGO INTERMEDIO 1

NOMBRE DE CURSO:	Lenguajes y Compiladores Avanzados LABORATORIO
MODULO:	Lenguajes y Compiladores Avanzados – 7
OBJETIVOS:	• Establecer la importancia de la utilización de código intermedio. • Definir representaciones intermedias que pueden contribuir a generar el código intermedio. • Establecer las transformaciones más comunes al generar código tres direcciones. •

58 Capítulo: Generación de Código Intermedio 1

ORGANIZACIÓN DE LENGUAJES Y COMPILADORES 2

7.1 Código Intermedio

El código intermedio es un código abstracto independiente de la máquina para la que se generará el código objeto. Permite la traducción a un código más cercano al código distinto o final. Por lo general al código de máquina.

7.1.1 Ventajas

- Mayor Modularidad: Permite la abstracción de componentes de alto nivel. Divide los elementos de alto nivel con los dependientes de máquina.
- Facilita la optimización y la generación de código de máquina: Simplifica las características de máquina objetivo: limita el número de registros y el tipo de instrucciones y la alineación de datos en modo de direccionamiento.
- Mayor transportabilidad: Independencia del lenguaje fuente y la máquina objetivo y del lenguaje fuente. Puede generarse otros códigos de máquina a partir del código intermedio.

7.1.2 Desventajas

- Necesidad de una fase extra para traducir a código máquina
- Mayor coste computacional
- Dificultad para definir un lenguaje intermedio adecuado
- Compromiso entre la representación de elementos código fuente y del código máquina.
- Dificulta llevar a cabo optimizaciones específicas de la arquitectura destino.

59 Capítulo: Generación de Código Intermedio 1

ORGANIZACIÓN DE LENGUAJES Y COMPILADORES 2

7.1.3 Representación Intermedia

Una representación intermedia es una estructura de datos que representa al programa fuente durante el proceso de la traducción a código objeto. A esta representación intermedia, que se parece al código objeto pero que sigue siendo independiente de la máquina, se le llama *código intermedio*.

El código intermedio puede tomar muchas formas. Todas ellas se consideran como una forma de linearización del árbol sintáctico, es decir, una representación del árbol sintáctico de forma secuencial. El código intermedio más habitual es el código de 3-direcciones. Las principales características son:

- Es fácil de generar durante el análisis semántico.
- Facilitar traducción al lenguaje de máquina final a partir del lenguaje objeto.
- Construcción sencilla, uniforme y clara del código.

7.1.4 Tipos de Representaciones

El código intermedio puede representarse por medio de:

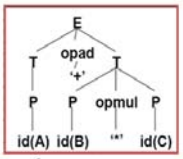
- Árboles: árboles de sintaxis abstracta, GDAs.
- Lineal: polaca inversa, código tres direcciones.

60 Capítulo: Generación de Código Intermedio 1

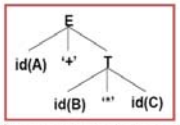
Figura 37. Páginas 61-64 Libro "Compiladores Avanzado"

Árboles de Sintaxis Abstracta

Es un árbol cuyos nodos son de tipo semánticos y no incluye símbolos terminales.



Árbol de Sintaxis



Árbol de Sintaxis Abstracta

Imagen 27: Ejemplo Árbol de Sintaxis

7.1.4.1.1 Ventajas:
Unificación de los pasos de compilación

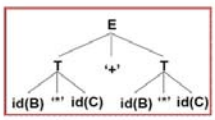
- Creación del árbol
- La tabla de símbolos
- Análisis semántico
- Optimización
- Generación de código objeto

7.1.4.1.2 Desventaja:
- Utiliza más espacio para el almacenamiento

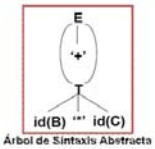
Grafo Acíclico Directo

Árbol Sintáctico Conciso, es un grafo que define un grafo para la representación de código intermedio.

7.1.4.1.3 Ventajas:
- Ahorran espacio en memoria.
- Mejoran el rendimiento, eliminando operaciones duplicadas en el código.
- Es complicada la construcción del grafo.



Árbol de Sintaxis



Árbol de Sintaxis Abstracta

Imagen 28: Ejemplo de Grafo Acíclico Directo

Código Tres Direcciones

Es una secuencia de instrucciones donde únicamente se accede a un máximo de 3 posiciones de memoria.

Imagen 29: Ejemplo básico de Código de Tres Direcciones

Las proposiciones más comunes son:

1. **x = y op z** donde **op** es un operador binario, aritmético, lógico o relacional.
2. **x = op y** donde el **op** es una negación lógica, menos unario, operadores de desplazamiento o conversión de tipos.
3. **x = y** asignación de **y** a **x.**
4. **x = y[i]** asignación con índices, con i posiciones de memoria, o de la forma **x[i] = y**
5. **goto etiq** Salto incondicional a la siguiente instrucción que se ejecutar.
6. **if false x goto etiq** Salto condicional
7. **param x y call f** llamadas a funciones o procedimientos con el uso de parámetros.
8. **x = *y** asignación o utilización de punteros.

Proposición de la forma **x = y op z,** donde los operandos x, y representan cualquier valor; ya sean variables, temporales generadas por el compilador, constantes, etc. Un operador puede ser un operador binario, aritmético, lógico o relacional.

7.1.5 Asignación de Temporales

Debido a que únicamente se pueden utilizar tres posiciones de memoria para realizar una operación, se hace necesaria la utilización de variables temporales. Por lo general se utiliza la variable t acompañada de un número correlativo de la forma **t[Numeral]**. Para asignara el número que le corresponde a un temporal puede llevarse de dos formas:

- Una variable global que se aumenta a medida que se necesita un nuevo temporal.
- Una función **nuevoTemp()** que nos permita manejar de mejor forma la adquisición del valor.
- Al crear una temporal es importante verificar de que tipo debe ser la temporal.
- Ya que se puede crear temporales de tipo entero, decimal, carácter, cadena, booleana, etc.

7.1.6 Código Tres Direcciones

Proposición de la forma **x = y op z,** partiendo de la siguiente asignación x = 14 - 9 + 7 (40 / 8).

```
t1 = 14 – 9
t2 = 40 / 8
t3 = 7 * t2
x  = t1 + t3
```

Proposición de la forma **x = y op z,** al momento de traducir al código intermedio, es importante tomar en cuenta la precedencia de operadores. Pues esto puede influir en el resultado. Precedencia de operadores aritméticos:

( ) [ ]	Agrupación
* /	Multiplicación y División
+ -	Adición y Substracción
(-)	Negativo

Proposición de la forma **x = y op z,** precedencia de operadores lógicos de agrupación y de operaciones relacionales:

( ) [ ]
AND
OR XOR
< menor
≤ menor igual
= igual
> mayor
≥ mayor igual
≠ no igual

Figura 38. Páginas 65-68 Libro "Compiladores Avanzado"

Operaciones Aritméticas

Proposición de la forma **y = 2+5*4-3**, se debe utilizar adecuadamente la precedencia

- t1 = 5 * 4 — t1 = 2 + 5
- t2 = 2 + t1 — t2 = t1 * 4
- t3 = t2 – 3 — t3 = t2 – 3
- **y = 19 OK** — **y = 25 X**

Combinación de operadores

Proposición de la forma **x = 3 * 4 + 2 > 20 AND True**

t1 = 3 * 4
t2 = t1 + 2
t3 = 20
b1 = t2 > t3
b2 = True
x = b1 AND b2

Temporales de tipo entero

Temporales de tipo lógico

Imagen 30: Código Tres Direcciones asociado a la combinación de operadores

7.1.7 Reutilización de Temporales

Las temporales ingresan a la tabla de símbolos, por lo que aumenta la utilización de memoria. Para ellos se puede reutilizar las temporales, únicamente verificando que ya haya sido utilizado.

Se puede utilizar la función **nuevoTemp()** genera un nombre temporal cada vez que se necesita, pero si ya existe una libre, puede utilizarla sin declarar una nueva. Partiendo de la operación **x = a * b + c * d – e * f.**

Creación de Nuevas Temporales	Reutilización de Temporales
t1 = a * b	t1 = a * b
t2 = c * d	t2 = c * d
t3 = t1 + t2	t1 = t1 + t2
t4 = e * f	t2 = e * f
t5 = t3 – t4	t1 = t1 – t2
x = t5	x = t1

7.1.8 Esquema de Traducción

Para representar este proceso, se puede hacer uso de un esquema de traducción, para generar el código intermedio asociado a cada operación. Un ejemplo sencillo de esto se puede observar en la siguiente traducción.

Producción	Regla Semántica
E → E1 **op** E2	E.lugar = nuevoTemp(); E.cod = generarCI(E.lugar = E1.cod OP E2.cod)
E → (E1)	E.lugar = E1.lugar E.Cod = E1.cod
E → id	E.lugar = lexema(id) E.Cod = generarCI(E.lugar = lexema(id))
E → num	E.lugar = lexema(num) E.Cod = ""

E.lugar = nombre de la temporal o variable
E.cod = secuencia de todas las temporales asociadas.
nuevoTemp() = obtención de un temporal
generarCI(parms) = genera el código intermedio

Imagen 31: Ejemplo Esquema de Traducción, operadores

7.1.9 Código Tres Direcciones

La asignación de una variable hacia otra, tomando en cuenta que las variables se encuentran almacenadas por lo generar en la pila, es necesario hacer una asignación previa. Proposición de la forma **x = y.** X = 20 / 2 + 4 donde x = Global. Generamos el código intermedio asociado

- t1 = 20/2 — x = Global
- t1 = t1 + 4
- x = t1

7.1.10 Esquema de Traducción

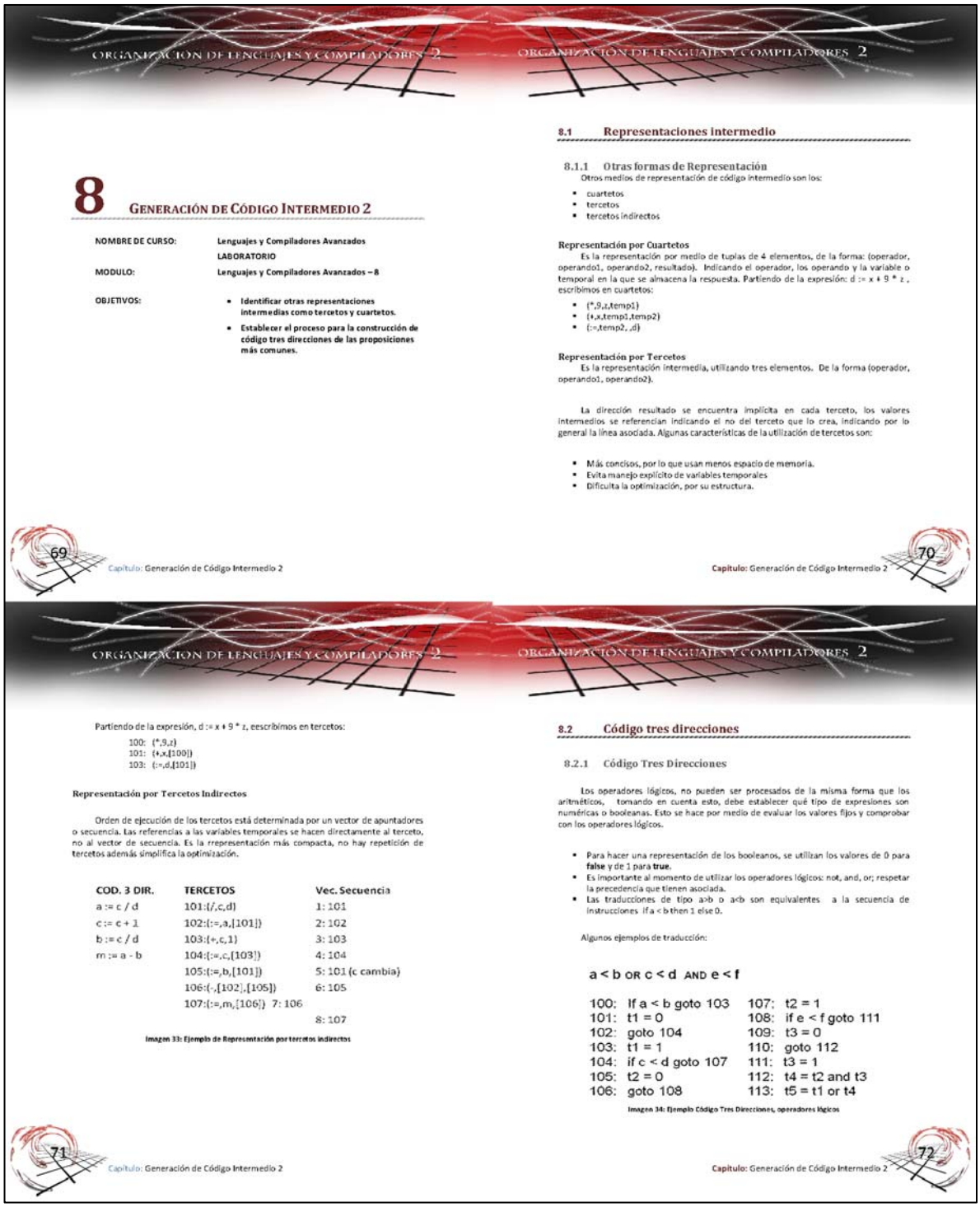
Producción	Regla Semántica
S → id = E	S.Cod = E.cod {lexema(id) = E.lugar}
E → **op** E2	E.lugar = nuevoTemp(); E.cod = generarCI(E.lugar = OP E1.cod)
E → (E1)	E.lugar = E1.lugar E.Cod = E1.cod
E → id	E.lugar = lexema(id) E.Cod = generarCI(E.lugar = lexema(id))
E → num	E.lugar = lexema(num) E.Cod = ""

E.lugar = nombre de la temporal o variable
E.cod = secuencia de todas las temporales asociadas.
nuevoTemp() = obtención de un temporal
generarCI(parms) = genera el código intermedio

Imagen 32: Ejemplo Esquema de Traducción, asignación

8.4 Generación de código intermedio 2

Figura 39. Páginas 69-72 Libro "Compiladores Avanzado"



8 GENERACIÓN DE CÓDIGO INTERMEDIO 2

NOMBRE DE CURSO: Lenguajes y Compiladores Avanzados
LABORATORIO

MODULO: Lenguajes y Compiladores Avanzados – 8

OBJETIVOS:

- **Identificar otras representaciones intermedias como tercetos y cuartetos.**
- **Establecer el proceso para la construcción de código tres direcciones de las proposiciones más comunes.**

8.1 Representaciones intermedio

8.1.1 Otras formas de Representación

Otros medios de representación de código intermedio son los:

- cuartetos
- tercetos
- tercetos indirectos

Representación por Cuartetos

Es la representación por medio de tuplas de 4 elementos, de la forma: (operador, operando1, operando2, resultado). Indicando el operador, los operando y la variable o temporal en la que se almacena la respuesta. Partiendo de la expresión: d := x + 9 * z , escribimos en cuartetos:

- (*,9,z,temp1)
- (+,x,temp1,temp2)
- (:=,temp2, ,d)

Representación por Tercetos

Es la representación intermedia, utilizando tres elementos. De la forma (operador, operando1, operando2).

La dirección resultado se encuentra implícita en cada terceto, los valores intermedios se referencian indicando el no del terceto que lo crea, indicando por lo general la línea asociada. Algunas características de la utilización de tercetos son:

- Más concisos, por lo que usan menos espacio de memoria.
- Evita manejo explícito de variables temporales
- Dificulta la optimización, por su estructura.

Partiendo de la expresión, d := x + 9 * z, escribimos en tercetos:

100: (*,9,z)
101: (+,x,[100])
103: (:=,d,[101])

Representación por Tercetos Indirectos

Orden de ejecución de los tercetos está determinada por un vector de apuntadores o secuencia. Las referencias a las variables temporales se hacen directamente al terceto, no al vector de secuencia. Es la rrepresentación más compacta, no hay repetición de tercetos además simplifica la optimización.

COD. 3 DIR.	TERCETOS	Vec. Secuencia
a := c / d	101:(/,c,d)	1: 101
c := c + 1	102:(:=,a,[101])	2: 102
b := c / d	103:(+,c,1)	3: 103
m := a - b	104:(:=,c,[103])	4: 104
	105:(:=,b,[101])	5: 101 (c cambia)
	106:(-,[102],[105])	6: 105
	107:(:=,m,[106]) 7: 106	
		8: 107

Imagen 33: Ejemplo de Representación por tercetos indirectos

8.2 Código tres direcciones

8.2.1 Código Tres Direcciones

Los operadores lógicos, no pueden ser procesados de la misma forma que los aritméticos, tomando en cuenta esto, debe establecer qué tipo de expresiones son numéricas o booleanas. Esto se hace por medio de evaluar los valores fijos y comprobar con los operadores lógicos.

- Para hacer una representación de los booleanos, se utilizan los valores de 0 para **false** y de 1 para **true.**
- Es importante al momento de utilizar los operadores lógicos: not, and, or; respetar la precedencia que tienen asociada.
- Las traducciones de tipo a>b o a<b son equivalentes a la secuencia de instrucciones if a < b then 1 else 0.

Algunos ejemplos de traducción:

a < b OR c < d AND e < f

100: If a < b goto 103	107: t2 = 1
101: t1 = 0	108: if e < f goto 111
102: goto 104	109: t3 = 0
103: t1 = 1	110: goto 112
104: if c < d goto 107	111: t3 = 1
105: t2 = 0	112: t4 = t2 and t3
106: goto 108	113: t5 = t1 or t4

Imagen 34: Ejemplo Código Tres Direcciones, operadores lógicos

Figura 40. Páginas 73-76 Libro "Compiladores Avanzado"

8.2.2 Esquema de Traducción

Expresiones Lógicas

Producción	Regla Semántica
E → E op E (and, or)	E0.dir = crearTemporal() E0.codigo = E1.codigo + E2.codigo + generarCI('E0.dir = E1.dir OP E2.dir')
E → op E (not)	E0.dir = crearTemporal() E0.codigo = E1.codigo + generarCI('E0.dir = OP E1.dir')
E → true	E.dir = crearTemporal() E.codigo = generarCI('E.dir = true')
E → false	E.dir = crearTemporal() E.codigo = generarCI('E.dir = false')

Tabla 1: Esquema de Traducción, expresiones lógicas

Expresiones Relacionales

Producción	Regla Semántica
E → E op_rel E (<,>,≤,≥,=,≠)	E0.dir = crearTemporal() ETQ_TRUE = nuevaEtiqueta() ETQ_FIN = nuevaEtiqueta() E0.codigo = E1.codigo + E2.codigo + generarCI('if (E1.dir OPREL E2.dir) goto ETQ_TRUE') generarCI('E0.dir = 0')+ generarCI('goto ETQ_FIN)+ generarCI('ETQ_TRUE:')+ generarCI('E0.dir = 1')+ generarCI('ETQ_FIN:')

Tabla 2: Esquema de Traducción, Expresiones Relacionales

8.2.3 Código Tres Direcciones

Asignación de la forma **x = y[i]** o **x[i] = y**, la asignación o utilización de posiciones de memoria, arreglos, vectores o matrices. Se hace por medio del desplazamiento de i posiciones de memoria a partir de la declaración de la variable.

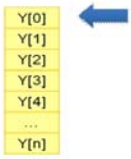


Imagen 35: Utilización de las posiciones de memoria

La transformadón a partir de las siguientes expresiones

x[10] := a+b	y[3] = z[3]
t1 := a+b	t1 = z[3]
x[10] := t1	y[3] = t1

El uso de arreglos limita el uso de posiciones de memoria, ya que se debe transformar a un arreglo lineal de una única dimensión. Por medio de la siguiente fórmula se obtiene la posición. vec[a,b]. **Posicion = base + ((a-li_a)*n2 + (b-li_b))*a**

Asignación de forma **x = y[i,j]**. Buscando **y[3,2]** donde y[a,b]:

pos = (a-li_a)*(+ (b-li_b)+1
pos = (3-1)*4 + (2-1) +1
pos = 8 + 2 = 10

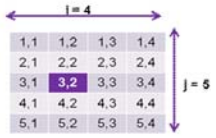


Imagen 36: Ejemplo posiciones de memoria, matrices

Utilizadón de punteros a direcciones de memoria.

- "x := &y" (asigna a x la dirección de y)
- "x := *y" (asigna a x el contenido apuntado por la dir. guardada en y)
- "*x := y" (guarda en la dir. guardada en la var. x el valor de y)

8.3 Instrucciones de control de flujo

8.3.1 Salto Condicional

Se refiere al uso de if, else o case bajo alguna condición. Para el traslado hacia código de tres direcciones se hace por medio de la asignación de Labels y sentencias goto.

Imagen 37: Ejemplo Salto Condicional

8.3.2 Salto Condicional If E then S

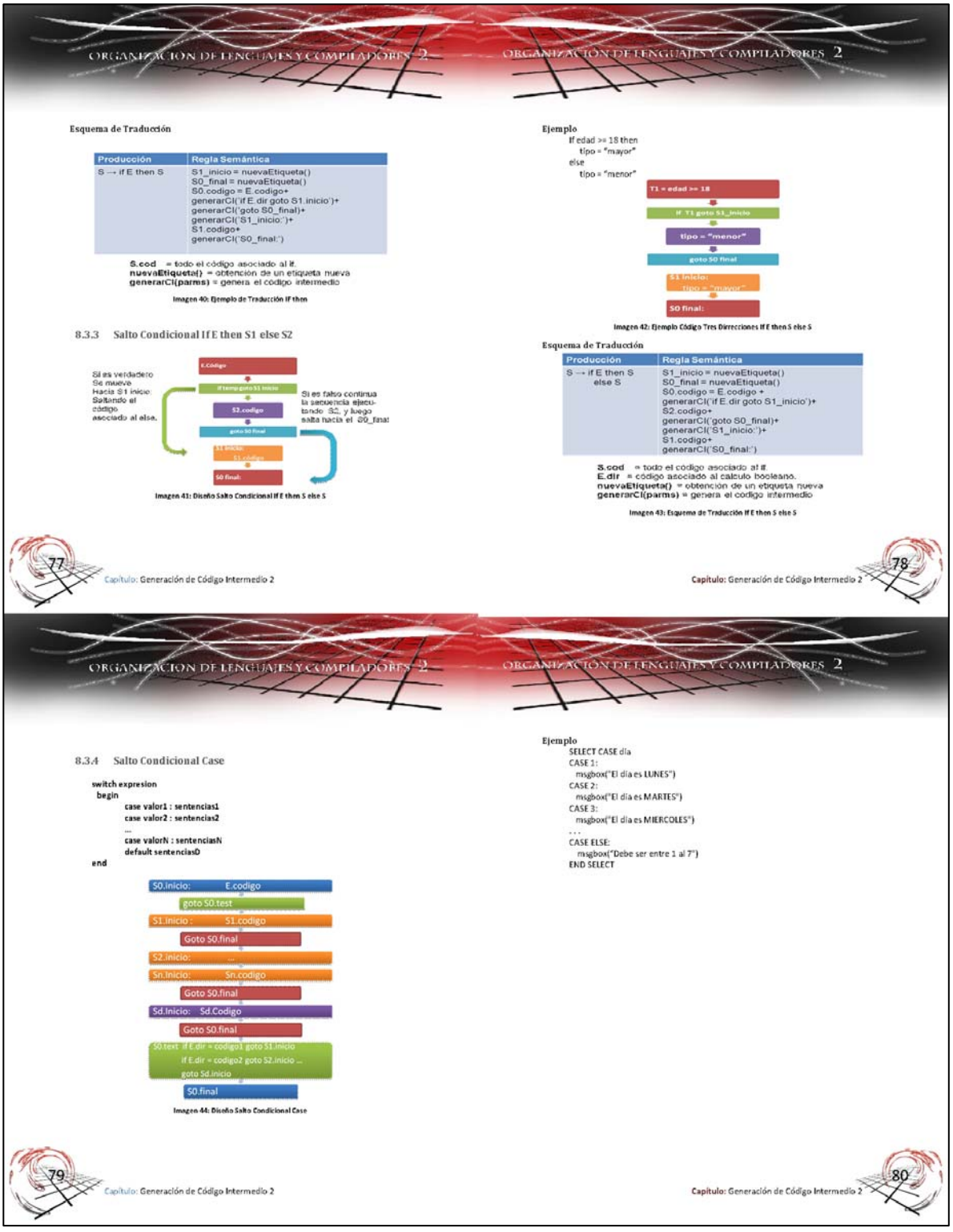
Partiendo de la instrucción **if E then código**

Imagen 38: Diseño de Salto Condicional If E then S

Ejemplo
If val > 3 + 4 then
val = 8 - 2

Imagen 39: Ejemplo Código Tres Direcciones If E then S

Figura 41. Páginas 77-80 Libro "Compiladores Avanzado"



ORGANIZACIÓN DE LENGUAJES Y COMPILADORES 2

Esquema de Traducción

Producción	Regla Semántica
S → if E then S	S1_inicio = nuevaEtiqueta() S0_final = nuevaEtiqueta() S0.codigo = E.codigo+ generarCI('if E.dir goto S1.inicio')+ generarCI('goto S0_final)+ generarCI('S1_inicio:')+ S1.codigo+ generarCI('S0_final:')

S.cod = todo el código asociado al if.
nuevaEtiqueta() = obtención de un etiqueta nueva
generarCI(parms) = genera el código intermedio

Imagen 40: Ejemplo de Traducción IF then

8.3.3 Salto Condicional If E then S1 else S2

Imagen 41: Diseño Salto Condicional If E then S else S

ORGANIZACIÓN DE LENGUAJES Y COMPILADORES 2

Ejemplo

```
If edad >= 18 then
    tipo = "mayor"
else
    tipo = "menor"
```


Imagen 42: Ejemplo Código Tres Direcciones If E then S else S

Esquema de Traducción

Producción	Regla Semántica
S → if E then S else S	S1_inicio = nuevaEtiqueta() S0_final = nuevaEtiqueta() S0.codigo = E.codigo + generarCI('if E.dir goto S1_inicio')+ S2.codigo+ generarCI('goto S0_final)+ generarCI('S1_inicio:')+ S1.codigo+ generarCI('S0_final:')

S.cod = todo el código asociado al if.
E.dir = código asociado al calculo booleano.
nuevaEtiqueta() = obtención de un etiqueta nueva
generarCI(parms) = genera el código intermedio

Imagen 43: Esquema de Traducción If E then S else S

ORGANIZACIÓN DE LENGUAJES Y COMPILADORES 2

8.3.4 Salto Condicional Case

```
switch expresion
  begin
        case valor1 : sentencias1
        case valor2 : sentencias2
        ...
        case valorN : sentenciasN
        default sentenciasD
end
```


Imagen 44: Diseño Salto Condicional Case

ORGANIZACIÓN DE LENGUAJES Y COMPILADORES 2

Ejemplo

```
SELECT CASE dia
CASE 1:
  msgbox("El dia es LUNES")
CASE 2:
  msgbox("El dia es MARTES")
CASE 3:
  msgbox("El dia es MIERCOLES")
...
CASE ELSE:
  msgbox("Debe ser entre 1 al 7")
END SELECT
```

8.5 Generación de código intermedio 3

Figura 42. Páginas 81-84 Libro "Compiladores Avanzado"

9 Generación de Código Intermedio 3

NOMBRE DE CURSO: Lenguajes y Compiladores Avanzados
LABORATORIO

MODULO: Lenguajes y Compiladores Avanzados – 9

OBJETIVOS:

- Establecer el proceso para la construcción de código tres direcciones de las proposiciones más comunes.

9.1 Instrucciones de control de flujo

9.1.1 Ciclo Condicional

Se refiere al uso de while, for, do bajo alguna condición. Para el traslado hacia código de tres direcciones se hace por medio de la asignación de Labels y sentencias goto.

```
L1: ...
goto L2
...
L2: ...
```

9.1.2 Ciclo Condicional While

Partiendo de la instrucción **While E do S**

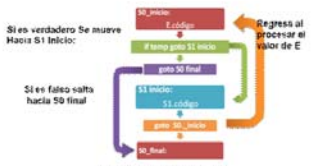


Imagen 45: Diseño Ciclo While

Ejemplo

Calculo 5 múltiplos de 2

```
i = 1
while (i <= 5) {
  print( 2 * i)
  i= +1; }
```

Imagen 46: Ejemplo Código Tres Direcciones, Ciclo While

9.1.3 Esquema de Traducción

Producción	Regla Semántica
S → while **E** do **S**	S0_inicio = nuevaEtiqueta() S0_final = nuevaEtiqueta() S1_inicio = nuevaEtiqueta() S0.codigo = generarCI('S0_inicio:')+ E.codigo+ generarCI('if E.dir goto S1_inicio')+ generarCI('goto S0_final)+ generarCI('S1_inicio:')+ S1.codigo+ generarCI('goto S0_inicio)+ generarCI('S0_final:')

S.cod = todo el código asociado al if.
nuevaEtiqueta() = obtención de un etiqueta nueva
generarCI(parms) = genera el código intermedio

Imagen 47: Esquema de Traducción, Ciclo While

9.1.4 Ciclo Condicional For

Imagen 48: Ejemplo Código Tres Direcciones, Ciclo For

Figura 43. Páginas 85-88 Libro "Compiladores Avanzado"

Ejemplo

Calculo de una potencia x^x

```
for(int i=1;i<n;i++)
x=x*x;
```

Imagen 49: Ejemplo Código Tres Direcciones, Ciclo For

9.1.5 Esquema de Traducción

Producción	Regla Semántica
S → for id = E to E do S	S0_inicio = nuevaEtiqueta() S0_final = nuevaEtiqueta() S0_test = nuevaEtiqueta() S1_inicio = nuevaEtiqueta() S0.codigo = generarCI('S0_inicio:')‖ E1.codigo ‖ E2.codigo ‖ generarCI('id.dir = E1.dir') generarCI('S0_test:')‖ generarCI('if (id.dir <= E2.dir) goto S1_inicio')‖ generarCI('goto S0_final)‖ generarCI('S1_inicio:')‖ S1.codigo‖ generarCI('id.dir = id.dir + 1')‖ generarCI('goto S0_test)‖ generarCI('S0_final:')

9.1.6 Llamadas a procedimientos y funciones

Al momento de utilizarlos es importante establecer el funcionamiento de estos, tanto en la llamada, como en el retorno o salida del mismo. Es aconsejable la utilización de registros de activación para el buen funcionamiento del traductor. A continuación se describen los aspectos importantes al hacer la llamada

- Evaluar valores de los parámetros
- Crear el registro de activación asociado al nuevo procedimiento
- Paso de parámetros
- Guardar estado del procedimiento actual
- Guardar dirección de retorno
- Salto al inicio del código del procedimiento llamado

Los aspectos importantes al hacer el retorno son:

- Recuperar estado del procedimiento donde se efectuó la llamada
- Salto a siguiente instrucción posterior a la llamada
- Recuperación del valor devuelto, en caso de ser función.

Para la llamada se debe de calcular cada uno de los parámetros, luego hacer el paso de parámetros y llamar al procedimiento o función.

Imagen 50: Diseño de llamadas a procedimientos

Ejemplo

calc(3+2,x,3*2)

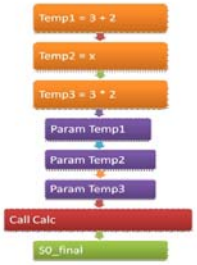


Imagen 51: Ejemplo Código Tres Direcciones, Procedimientos y Funciones

10 Optimización de CI

NOMBRE DE CURSO:	Lenguajes y Compiladores Avanzados LABORATORIO
MODULO:	Lenguajes y Compiladores Avanzados – 10
OBJETIVOS:	• Definir el proceso de optimización dentro de la síntesis. • Establecer las diversas reglas de traducción para la optimización. • Introducir el uso de bloques básicos y diagramas de flujo.

Figura 44. Páginas 89-92 Libro "Compiladores Avanzado"

10.1 Optimización de Código intermedio

Busca reducir el código, mejorar la utilización de memoria y optimizar el código. Pueden utilizarse optimizadores de una o dos pasadas. En las que se procede a optimizar el código y luego este código generado volverlo a optimizar.

Mejorar el código objeto final, preservando significado del programa

- Mejorar la velocidad de ejecución del programa
- Disminuir el tamaño del programa
- Optimizar la utilización de memoria, de acuerdo a las necesidades del programa.

Se deben de ejecutar las optimizaciones creando un equilibrio en el tamaño, velocidad y memoria.

10.1.1 Clasificación de Optimizadores

- En función de la arquitectura
  - Dependientes de la máquina
  - Independientes de la máquina
- En función del ámbito de aplicación
  - Optimizaciones locales
  - Optimizaciones Globales

Dependientes de la máquina

Busca maximizar el código haciendo uso de las características de la máquina. Utilización de modos de direccionamiento específicos para la máquina, en la asignación de registros. Utilización de instrucciones especiales (IDIOMS). Predicción de saltos y la utilización de estrategias de aprovechamiento de memoria cache.

Independientes de la máquina

Son aplicables en cualquier tipo de máquina objetivo, mejora en la ejecución en tiempo de compilación. Eliminación de código redundante y ccambios en el orden de ejecución, para optimizar la utilización de memoria.

Optimizaciones locales

Aplicadas dentro de un Bloque Básico. Utilización de todas de las transformaciones que optimizan el código intermedio generado.

Optimizaciones Globales

Aplicadas a más de un Bloque Básico. Aplica las transformaciones dentro de más de un B.B. considerando contenido y flujo de datos. Debe llevar un control del flujo de los B.B. y sus interrelaciones. Utilizadón de un algoritmos de análisis global de flujo de datos

10.1.2 Estructura Optimizador

Estructura de un optimizador de Código intermedio

Imagen 52: Estructura de un Optimizador de Código

10.1.3 Optimizaciones Locales

Algunas transformaciones que se realizan de forma local son:

- Pre cálculo de expresiones
- Reutilización de Expresiones Comunes
- Propagación de Copias

Algunas transformaciones algebraicas que se realizan de forma local son:

- Eliminación secuencias nulas
- Reacondicionamiento de operadores

Pre cálculo de expresiones constantes

Pre calcula expresiones constantes con constantes o variables cuyo valor no cambia.

Imagen 53: Ejemplo Pre cálculo de Expresiones

Reutilización de expresiones comunes

Al momento de encontrar una instrucción que utiliza las mismas variables, y estas no han cambiado su valor, se puede substituir por la variable que la contiene.

Imagen 54: Ejemplo Reutilización de Expresiones Comunes

Propagación de copias

Con asignaciones directas de tipo x = y, donde las variables no cambian su valor se puede substituir x por y.

Imagen 55: Ejemplo Propagación de Copias

Figura 45. Páginas 93-96 Libro "Compiladores Avanzado"

Eliminación secuencias nulas

- Operaciones en la que su resultado no cambia.

Imagen 55: Ejemplo Eliminación de Secuencias Nulas

Eliminación de Código Inactivo

El código inactivo o inútil es un conjunto de proposiciones que nunca llegan a utilizarse en el código.

Imagen 57: Ejemplo Eliminación de Código Inactivo

Reacondicionamiento de operadores

Cambiar el orden de evaluación ampliando propiedades asociativa y distributiva

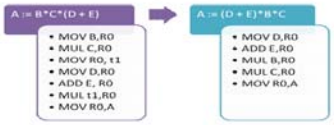


Imagen 58: Ejemplo Reacondicionamiento de Operadores

10.2 Bloques básicos y Diagrama de flujos

10.2.1 Bloques Básicos y Diagrama de Flujos

Bloques Básicos

Es una secuencia de código intermedio, de forma consecutiva de instrucciones donde el flujo de control es lineal.

Diagrama de Flujo

Representación gráfica que permite estructurar y recoger la información del programa. Se representa por nodos (bloques básicos) y arcos (representan el flujo entre B.B)

10.2.2 Identificación de Bloques Básicos

Identificación del principio de un BB

- Primera instrucción de un B.B.
- Cualquier instrucción destino de un salto,
- Cualquier instrucción inmediatamente después de un salto

Identificación del final de un BB

- Comprende las instrucciones que van desde el principio del BB hasta el siguiente BB (sin incluirlo) o hasta el final de programa

10.2.3 Construcción Diagramas de Flujo

Creación de Arcos

- Si el B.B. termina en salto incondicional: incluir un arco hasta el B.B. destino del salto
- Si el B.B. termina en salto condicional
- Incluir un arco hasta el B.B. destino del salto e incluir otro arco al siguiente B.B.
- En otro caso: incluir arco al siguiente B.B. (si lo hay)

10.2.4 Diagramas de Flujo

```
100: i := m - 1        110: t5 := a [t4]      120: a [t10] := x
101: j := ni           111: if (t5 > v) goto 108  121: goto 104
102: t1 := 4 * n       112: if (i > j) goto 122   122: t11 := 4 * i
103: v := a [t1]       113: t6 := 4 * i       123: x := a [t11]
104: i := i + 1        114: x := a [t6]       124: t12 := 4 * i
105: t2 := 4 * i       115: t7 := 4 * i       125: t13 := 4 * n
106: t3 := a [t2]      116: t8 := 4 * j       126: t14 := a [t13]
107: if (t3 < v) goto 104  117: t9 := a [t8]   127: a [t12] := t14
108: j := j -1         118: a [t7] := t9      128: t15 := 4 * n
109: t4 := 4 * j       119: t10 := 4 * j      129: a[t15] := x
```

Imagen 59: Código Tres Direcciones Ejemplo

Identificación del principio de cada bloque básico

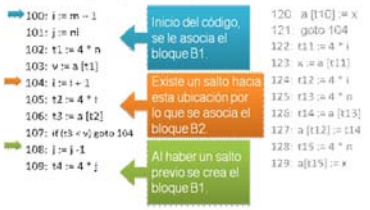


Imagen 60: Identificación de Bloques Básicos (parte 1)

Identificación del principio de cada bloque básico

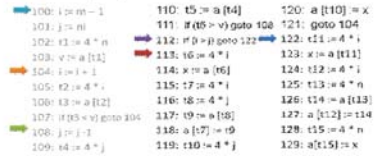


Imagen 61: Identificación de Bloques Básicos (parte 2)

CONCLUSIONES

1. Se desarrolló un instrumento para definición, desarrollo y evaluación de laboratorios de cursos denominado “Guía del Instructor” basado en la metodología del *IT Education Centre of Excellence,* utilizado en el proyecto *India-Guatemala IT Education Centre of Excellence*.

2. Se elaboró las guías de laboratorio de los cursos Lenguajes Formales y de Programación, Organización de Lenguajes y Compiladores 1, Organización de Lenguajes y Compiladores 2. El contenido que se incluyó en cada laboratorio fue basado en los actuales programas de laboratorio y actualizados por sugerencias de los catedráticos titulares de los cursos en cuestión.

3. En cada Guía de Instructor se desarrollaron los elementos: Información general del curso, distribución de horas y actividades, forma de evaluación, detalle de sesiones, distribución de sesiones, detalle de tareas asignadas, detalle de tutoriales, detalle de exámenes, ejemplos de exámenes, bibliografía y anexos.

4. Se desarrolló el contenido de los laboratorios en presentaciones, donde cada presentación constituye una sesión teórica del laboratorio, proveyendo a estudiantes y auxiliares una base estandarizada de información.

5. Para las prácticas, exámenes y tutoriales de laboratorios se definieron los objetivos que se buscan de cada actividad, basado en la aplicación gradual de los contenidos y la evaluación constante como un medio de aprendizaje.

6. Se realizó la guía de clase para los Lenguajes Formales y de Programación, Organización de Lenguajes y Compiladores 1, Organización de Lenguajes y Compiladores 2. El contenido fue estructurado con base a los programas de curso y actualizado en un esfuerzo en conjunto con los catedráticos titulares de los cursos mencionados anteriormente.

7. Se recopiló un conjunto de artículos relacionados con los contenidos de los cursos, proporcionando información adicional que puede ser de para enriquecer la formación.

8. Mediante los entregables del proyecto se conformó una guía detallada para impartir los laboratorios y clases de los cursos anteriormente mencionados. Se provee la plataforma para estandarizar los contenidos y para facilitar la actualización de los mismos.

RECOMENDACIONES

1. Es crucial para el éxito de este proyecto tomar en cuenta que el contenido debe ser actualizado constantemente, es importante mantener la estructura básica de la guía y todos los módulos definidos, ya que cada uno tiene un objetivo específico.

2. Es importante para conservar la integridad de estas guías, que las personas encargadas de realizar las actualizaciones tengan experiencia en cada curso y que sean guiadas por el catedrático titular del mismo.

3. Finalmente se sugiere llevar control de las diferentes versiones de las guías al realizar actualización de contenido, para evitar pérdida de información.

BIBLIOGRAFÍA

1. Panda, Mrutunjaya. **Curso de Planificación Educativa a través de Instructor Guidelines.** India-Guatemala IT Education Centre of Excellence. Guatemala. Enero a Julio de 2008.

2. **Aho, Alfred V., Sethi, Ravi y Ullman, Jeffrey D.** Compiladores, pricipios, técnicas y herramientas. 1990.

3. **Gálvez Rojas, Sergio y Mora Mata, Miguel Ángel.** Traductores y Compiladores con LEX/YACC, JFLEX/CUP Y JAVACC. 2005.

4. **Louden, Kenneth C.** Construcción de compiladores/ Construction of compilers: Principios y práctica. 2004.

5. **Rodríguez Avila, Eduardo René.** Diseño de Compiladores. 2000.

6. **Taylor, Gregory.** Models of Computation and Formal Languages . 2003.

ANEXOS

Para la estructuración de los laboratorios y el contenido de la documentación de apoyo se contó con el soporte y revisión de cada catedrático responsable de cada uno cursos en la Escuela de Ciencias y Sistemas.

Por lo cual, en el anexo se adjuntan las cartas autorizando el contenido de los laboratorios por parte del catedrático titular de cada curso:

- Ing. Cristian Lavarreda – Lenguajes Formales y de Programación.
- Ing. Manuel Francisco Noriega – Organización de Lenguajes y Compiladores 1.
- Ing. Bayron López López – Organización de Lenguajes y Compiladores 2.

Donde se revisó los documentos necesarios y a criterio de cada ingeniero estos cumplen con el objetivo y contenido planteado para el laboratorio y documentación de cada curso.

Guatemala, 29 de septiembre de 2008

Ingeniero
Jorge Armin Mazariegos
Asesor de EPS
Facultad de Ingeniería
USAC

Por este medio me dirijo a usted para informarle que he revisado la **Guía del Instructor del Curso *"Lenguajes Formales y de Programación"***. Documento que forma parte del Proyecto "*Estructuración de los laboratorios y documentación de apoyo de los cursos Lenguajes Formales y de Programación*". Y a mi parecer cumple con los objetivos de la guía de laboratorio que se encuentra vigente.

Agradeciendo su atención a la presente,

Atentamente,

Ing. Cristian Lavarreda
Catedrático del Curso Lenguajes Formales y de Programación
Escuela de Ciencias y Sistemas, Facultad de Ingeniería, USAC

Guatemala, 16 de octubre de 2008

Ingeniero
Jorge Armin Mazariegos
Asesor de EPS
Facultad de Ingeniería
USAC

Por este medio me dirijo a usted para informarle que he revisado el **Guía del Instructor del Curso "*Organización de Lenguajes y Compiladores 1*"**, del Proyecto "Estructuración de los laboratorios y documentación de apoyo de los cursos Lenguajes Formales y de Programación, Organización de Lenguajes y Compiladores I y Organización de Lenguajes y Compiladores II", y a mi parecer cumple con los objetivos de guía de laboratorio que actualmente se encuentra vigente.

Agradeciendo su atención a la presente,

Atentamente,

Ing. Manuel Francisco Noriega
Catedrático del Curso Organización de Lenguajes y Compiladores 1
Escuela de Ciencias y Sistemas, Facultad de Ingeniería, USAC

Guatemala, 8 de julio de 2009

Ingeniero
Jorge Armin Mazariegos
Asesor de EPS
Facultad de Ingeniería
USAC

Por este medio me dirijo a usted para informarle que he revisado el **Guía del Instructor del Curso "*Organización de Lenguajes y Compiladores 2*"**, del Proyecto "Estructuración de los laboratorios y documentación de apoyo de los cursos Lenguajes Formales y de Programación, Organización de Lenguajes y Compiladores I y Organización de Lenguajes y Compiladores II", desarrollado por Lisbeth Ricart y a mi parecer cumple con los objetivos de guía de laboratorio vigente durante el ciclo 2008, esta guía será la base para el desarrollo del nuevo contenido del laboratorio durante el ciclo 2009 y posteriores.

Agradeciendo su atención a la presente,

Atentamente,

Ing. Bayron López López
Catedrático del Curso Organización de Lenguajes y Compiladores 2
Escuela de Ciencias y Sistemas, Facultad de Ingeniería, USAC