Contenido

Métodos sintácticos descendentes.

Métodos sintácticos ascendentes.

Introducción

Desarrollar un compilador es una de las aplicaciones más complejas en el área de sistemas computacionales, debido a que se necesita integrar un conocimiento profundo en el área de matemáticas discretas, la programación de propósito general y las estructuras de datos.

Sin embargo, el desarrollo de un compilador puede facilitarse si se analizan detalladamente cada una de las etapas que lo integran.

El concepto de compilador está intimamente ligado con el de lenguajes de programación, que en la actualidad son poderosas herramientas de desarrollo, los cuales integran un ambiente de programación que generalmente incluye un editor, ambientes visuales, depuradores, preprocesadores, un compilador, cargadores, funciones de biblioteca de propósito general, manejadores de bases de datos, controladores para sistemas distribuidos, instaladores, etc.

La responsabilidad del compilador dentro de los lenguajes de programación es la de convertir las instrucciones de alto nivel a código ensamblador, el resto de las responsabilidades en la generación de aplicaciones en un lenguaje de programación corren a cargo de las herramientas antes mencionadas.

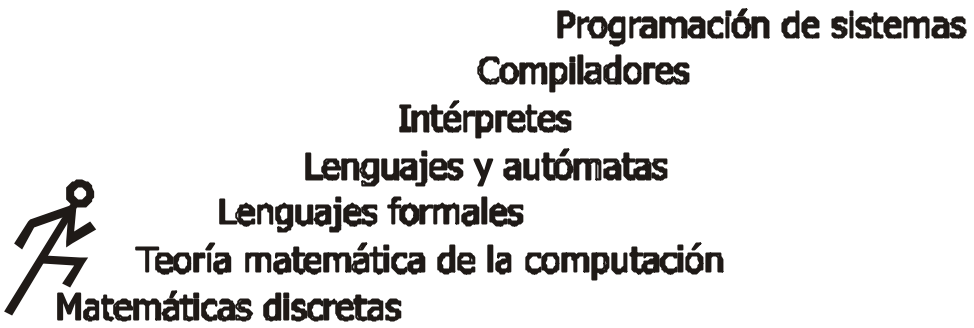
Se reconocen casi universalmente seis etapas en el desarrollo de un compilador, las cuales son.

- Léxico.
- Sintaxis.
- Semántica.
- Generación de código intermedio.
- Optimización de código.
- Generación de código ensamblador.

Las tres primeras etapas tienen una alta carga matemática y las tres restantes una alta carga de algoritmos y estructura de datos. Por lo anterior es necesario dominar, realmente dominar, la parte matemática y de programación combinada con estructuras de datos para desarrollar un compilador.

El objetivo de esta obra es aplicar con profundidad, claridad, detalle y “desde cero” los modelos matemáticos que sustentan la teoría de compiladores, para esto es necesario que el programa de aprendizaje incluya temas periféricos o que se supone se deben tratar de manera especializada y por separado.

El camino del conocimiento necesario para desarrollar un compilador se encuentra plasmado en la gráfica que ilustra la portada.



Si se tienen algunos antecedentes en los temas se puede observar que se dedican obras de manera individual y especializada a cada uno de los temas.

Para lograr el objetivo de esta obra, se sintetizan los temas con mayor relevancia en el área de compiladores, distribuidos en los capítulos de la siguiente forma.

Matemáticas discretas.	Capítulo 1. Teoría de conjuntos El material está orientado a manejar la terminología necesaria en el desarrollo de compiladores y se considera como antecedente a la teoría de compiladores.
Teoria matemática de la computación.	Capítulo 3. Lenguajes formales. Capítulo 4. Autómatas finitos. Capítulo 5. Gramáticas. Capítulo 6. Autómatas de pila. Capítulo 7. Máquina de Turing. Teoría matemática de la computación es el término moderno de lo con anterioridad se conoce como lenguajes formales y teoría de autómatas.
Tema periférico, lenguaje natural.	Capítulo 2. Lenguajes. El material incluye un tema de lenguajes desde el punto de vista no formal, muestra soluciones a problemas que involucran lenguajes, sin fundamentarlos matemáticamente. Este tipo de soluciones se implementarían si el diseñador no tiene conocimientos en la teoría matemática de computación.
Lenguajes formales.	Capítulo 3. Lenguajes formales. El material establece la primer relación, la cual está dada en la definición de que un lenguaje formal es un tipo especial de conjunto.
Lenguajes y autómatas.	Capítulo 4. Autómatas finitos. El material ilustra la forma en que los lenguajes formales con una alta carga conceptual se pueden implementar en máquinas que pueden programarse.

Tema periférico, manejo de expresiones.	Capítulo 8. Expresiones, primer acercamiento. Las expresiones requieren de un manejo especial y cuidadoso. El material incluye un capítulo que muestra algoritmos no formales en la solución de expresiones. Por ser métodos no formales tienen poca aplicación práctica en un lenguaje de programación robusto, pero ilustran de manera muy clara conceptos útiles en los métodos formales.
Intérpretes.	Capítulo 4. Autómatas finitos. Capítulo 5. Gramáticas. Capítulo 9. Análisis sintáctico, autómatas de pila determinísticos. Capítulo 10. Análisis semántico, traducción dirigida por la sintaxis. Capítulo 11. Intérprete MIBASIC. Para poder desarrollar un lenguaje de programación completo se necesitan las etapas de léxico, sintaxis y semántica. Con éstas es posible desarrollar una aplicación que ejecute instrucciones. Existen dos formas de implementar un lenguaje de programación, una en formato compilador y la otra en formato intérprete. Formato compilador. Se traducen las instrucciones de alto nivel a código ensamblador, posteriormente se procesan en un compilador para ensamblador y se obtiene el código objeto, posteriormente se procesa en un cargador (linker) y se obtiene el código ejecutable que es independiente de las plataformas en que se desarrollaron los procesos de conversión del código. Formato intérprete. Se ejecutan las instrucciones conforme se están generando, la construcción de un lenguaje a nivel intérprete se puede llevar a cabo con las tres etapas iniciales de un compilador y es el objetivo de este libro, que el lector pueda desarrollar su propio lenguaje de programación con fundamento matemático, orden, control, eficacia, dominio y elegancia.
Compiladores.	Capítulo 4. Autómatas finitos. Capítulo 5. Gramáticas. Capítulo 9. Análisis sintáctico, autómatas de pila determinísticos. Capítulo 10. Análisis semántico, traducción dirigida por la sintaxis. En esta obra no considera el desarrollo final de la construcción de un compilador, considera las tres primeras etapas. Debido a que resultados palpables en una aplicación tomarán un poco más de tiempo, lo cual didáctimente no es aconsejable. Por lo anterior, esta obra aplica todo el fundamento matemático en el desarrollo de un lenguaje implementado en formato intérprete, lo que permite ver resultados en el mediano plazo de forma clara, práctica, contundente e inclusive divertida, aunque a mis estudiantes nunca los he visto muy divertidos en mis cursos.
Programación de sistemas.	No tratado en este libro. La programación de sistemas involucra herramientas paralelas al compilador o

	intérprete en los lenguajes de programación, algunas ya mencionadas anteriormente como son el preprocesador, el compilador de código ensamblador a código objeto, el cargador o linkeador, la interacción del lenguaje con el sistema operativo y la arquitectura de la computadora. La programación de sistemas es el último escalón en la construcción robusta de lenguajes de programación.

Las etapas restantes de un compilador; generación de código intermedio, optimización y generación de código junto con la programación de sistemas serán tratados en la obra que le dará seguimiento a “Compiladores, el comienzo ...”.