Tecnológico de Monterrey

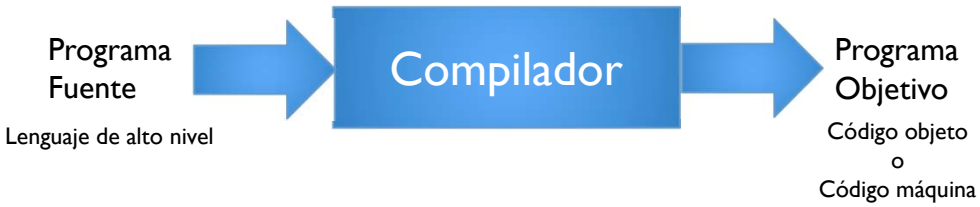
Introducción

Diseño de Compiladores

Dr. Víctor de la Cueva
vcueva@itesm.mx

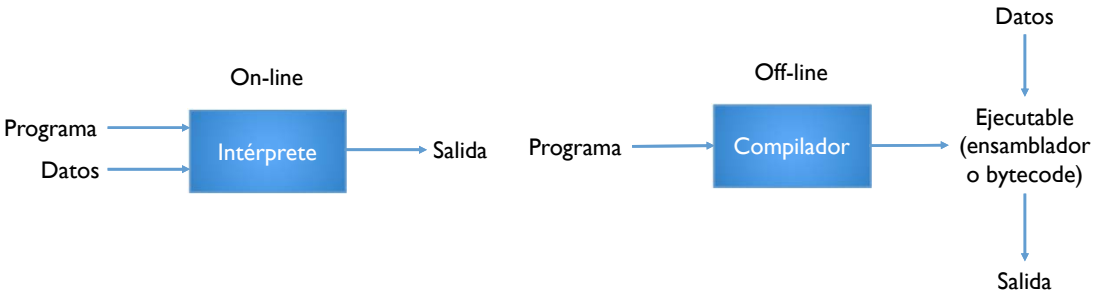
Definición

- Son programas de computadora que traducen un lenguaje a otro.



Dos aproximaciones

- Hay dos aproximaciones para la implementación de un lenguaje de programación:



Comentario inicial

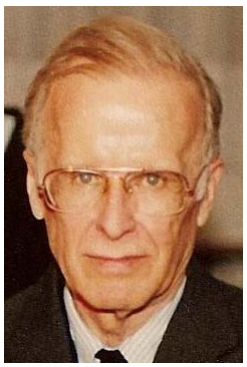
- Es una de las áreas más formal de las ciencias computacionales.
- Se deriva directamente de la Teoría de la Computación.
  - Tiene su parte artesanal
- Nace junto con la computadora

Los iniciadores

https://en.wikipedia.org/wiki/John_von_Neumann#/media/File:JohnvonNeumann-LosAlamos.gif
By Plerre.Lescanne - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=66119797
http://americanempireproject.com/authors/noam-chomsky/

- Muchas gente ha participado en la historia de los compiladores pero hay tres que iniciaron todo:

John von Neumann
(1903-1957)



John Backus
(1924-2007)

Noam Chomsky
(1928-)

¿Por qué un compilador?

- John von Neumann (finales 40s): computadora con programa almacenado:
  - Lenguaje de máquina (e.g. C7 06 000 002)
  - Lenguaje ensamblador (e.g. MOV X, 2):
    - Difícil de escribir y comprender
    - Depende en extremo de una máquina particular
  - Lenguajes de alto nivel
- John Backus en IBM
  - 1953, Speedcoding para IBM 704 (10-20x lento, 300 b ≈ 30% mem)
  - 1954-1957: primer compilador para FORTRAN I (1958, 50% prog).

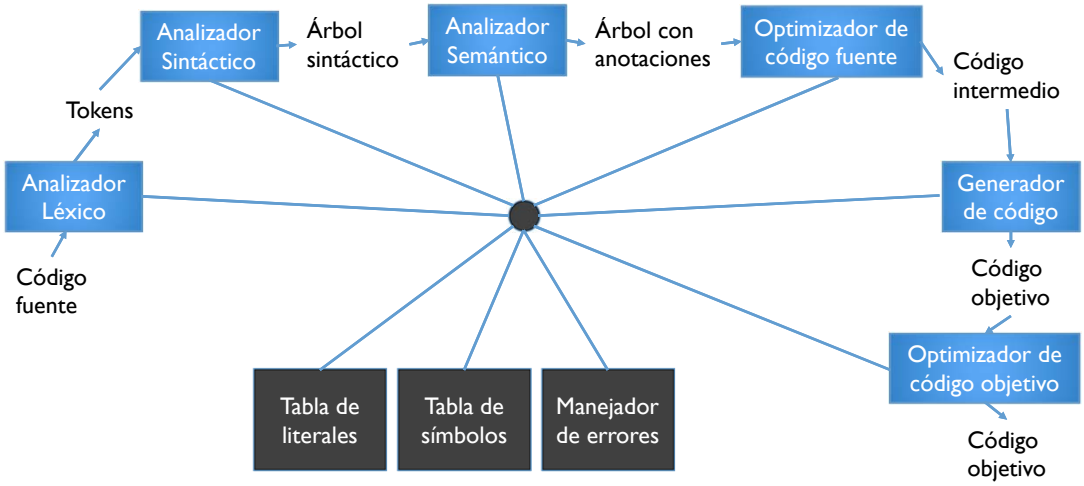
Herramientas

- Noam Chomsky: inicia el estudio de la estructura del lenguaje natural.
  - Facilita la construcción de compiladores (se puede automatizar)
  - Clasifica los lenguajes de acuerdo a la complejidad de sus gramáticas y la potencia de los algoritmos necesarios para reconocerlas (cada una es una especialización de su predecesora):
    - Tipo 0: No restringida o recursivamente enumerable (Máquina de Turing)
    - Tipo 1: Sensible al contexto (MT no determinísticas con restricciones lineales)
    - Tipo 2: Libres de contexto (Autómatas *pushdown* no-determinísticos)
    - Tipo 3: Regulares (Autómata finitos o expresiones regulares)

Otras herramientas

- Técnicas de optimización o de mejoramiento de código
- Programas que automatizan la creación de compiladores
  - Generadores de analizadores léxicos (*scanners* o *lexers*)
    - Mike Lesk y Eric Schmidt (1975): Lex (Flex)
  - Generadores de analizadores sintácticos
    - Steve Johnson (1975): Yacc (Bison)

Proceso de traducción



Analizador léxico

- Scanner o lexer.
- Lectura real del programa fuente.
  - Flujo de caracteres
- Análisis léxico
  - Secuencias de caracteres en unidades significativas llamadas tokens
  - Similar al deletreo
- Otras funciones:
  - Identificadores a tabla de símbolos
  - Literales (constantes numéricas o de texto) a tabla de literales
- Ejemplo:
  - a[i] = 5 * 3

Analizador sintáctico

- Parser
- Recibe los tokens del analizador léxico y realiza análisis sintáctico:
  - Verifica la estructura del programa
  - Similar a análisis gramatical
  - Determina los elementos estructurales del programa y sus relaciones
  - Regresa un árbol de análisis gramatical o árbol sintáctico
- Ejemplo: elemento *expresión*

Analizador semántico

- La semántica de un programa es su significado.
  - Determina su comportamiento en el tiempo de ejecución
  - Algunas veces antes: semántica estática
    - Declaración y verificación de tipos
- El analizador semántico se encarga de la semántica estática.
  - Las partes extras que se calculan con este analizador se llaman atributos
  - Se agregan al árbol como anotaciones

Optimizador del código fuente

- Etapas para el mejoramiento del código fuente.
- Se pueden colocar en varios lugares.
  - Algunas se pueden realizar sobre el árbol:
    - Incorporación de constantes (e.g. 5 * 3)
  - Es más fácil optimizar sobre una forma linearizada del árbol:
    - Código de tres direcciones
    - Código P
- Su salida es un código intermedio o RI (e.g. de tres direcciones)

Generador de código

- Toma el código RI y genera el código para la máquina objetivo:
  - Código máquina
  - Ensamblador
- Ejemplo:

```
MOV     R0, i       ; valor de i -> R0
MUL     R0, 2       ; duplica el valor en R0
MOV     R1, &a      ; dirección de a -> R1
ADD     R1, R0      ; sumar R0 a R1
MOV     *R1, 6      ; constante 6 -> dirección en R1
```

Optimizador de código objetivo

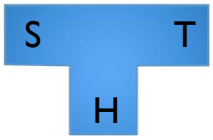
- Intenta mejorar el código objetivo generado:
  - Modos de direccionamiento
  - Reemplazo de instrucciones lentas por rápidas
  - Eliminar operaciones redundantes
  - Etc.
- Ejemplo: usar instrucción de corrimiento a la izquierda (SHL) para reemplazar la multiplicación por dos

Estructuras de datos

- Tokens (enumerado)
- Árbol sintáctico
- Tabla de símbolos
  - Mantiene la información asociada con los identificadores
    - Funciones, variables, constantes y tipos
  - Interactúa con casi todas las fases de compilador
- Tabla de literales
  - Almacena constantes: numéricas y de cadena
- Código intermedio (arreglo de cadenas de texto)
- Archivos temporales (no se tiene memoria suficiente)

Lenguaje anfitrión

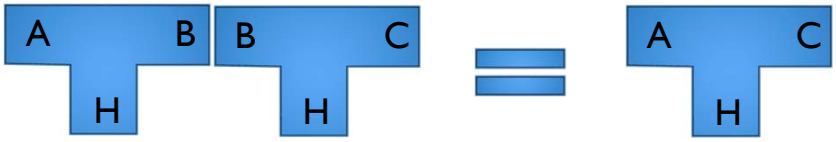
- Tercer lenguaje involucrado.
- Es en el que se escribe el compilador.
  - Ya existe un compilador para este lenguaje
- Se esquematiza esto en un diagrama T:
  - Compilador se escribe en un lenguaje H (host)
  - Traduce lenguaje S (source)
  - A lenguaje T (target)



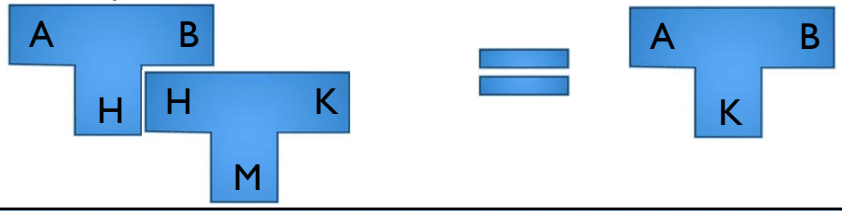
- Equivalente a decir que el compilador se ejecuta en la máquina H.
  - Típicamente esperamos que H = T

Combinación de diagramas T

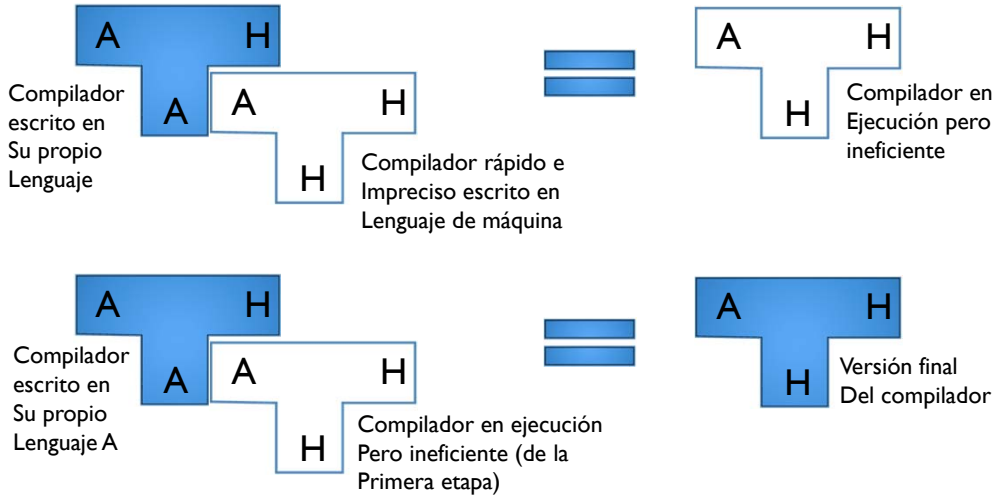
- Dos maneras:
  - Tenemos dos compiladores que se ejecutan en la máquina H, uno traduce de A a B y el otro de B a C



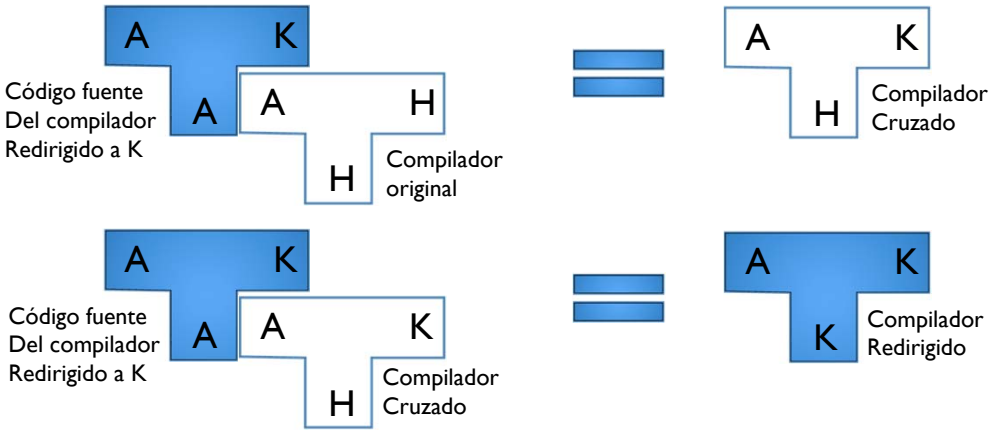
  - Usar el compilador de H a K para traducir el lenguaje de implementación de otro compilador de H a K



Ventaja: transferencia



Ventaja: Portabilidad

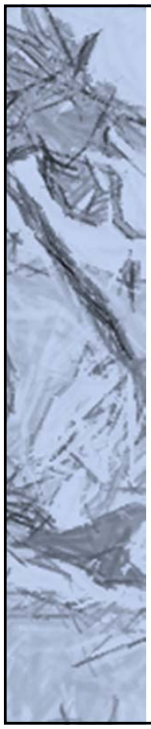


Lenguaje TINY

- Es el lenguaje que se usará como muestra durante todo el curso:
  - Muy simple
  - Sintaxis con características de: Pascal, C, C++ y Ada
  - Con todas las características necesarias para hacer un compilador completo
- Se corre en lenguaje ensamblador con algunas propiedades de máquinas con arquitectura RISC.

Lenguaje C-minus o C-

- Es el lenguaje para el que se desarrollará el compilador del curso:
  - Conjunto considerablemente restringido de C



Referencias

- A.V. Aho, M. S. Lam, R. Sethi, and J. D. Ullman. *Compilers: Principles, Techniques, and Tools*. 2nd Pearson (2012).
- K.C. Louden. *Contrucción de Compiladores: principios y práctica*. Thomson (2004).
- Alex Aiken. Compilers. Stanford Online (2018).
  - https://lagunita.stanford.edu/courses/Engineering/Compilers/Fall2014/about