INTRODUCCIÓN A COMPILADORES

El reto es vencer al dragon

11 de agosto de 2022

Ing. Msc. Víctor Orozco

Universidad Rafael Landivar

TOC

INTRODUCCIÓN

OBJETIVOS

- ☐ Entender que hace un compilador
- ☐ Entender como funciona un compilador
- ☐ Entender como se construye un compilador

LENGUAJES DE PROGRAMACIÓN

- Interpretes que ejecutan los programas
- Compiladores que traducen los programas

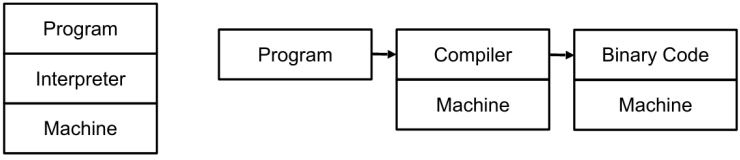


Figura: Compilador vs interprete

IMPLEMENTACIONES

- Los compiladores dominan los lenguajes de bajo nivel (C, C++, Go, Rust)
- Los intérpretes dominan los lenguajes de alto nivel (Python, Ruby)
- Algunas implementaciones de lenguaje proporcionan ambos (Java, JS, WebAssembly)
- Tendencia: Intérprete + compilador JIT

HISTORIA

- 1954: IBM desarrolla el 704
- El costo del software excede al hardware
- Toda la programación hecha en ensamblador

SOLUCIÓN

- Speedcoding
- Interprete
- 10-20 veces más lento que el ensamblador escrito a mano

FORTRAN I

- John Backus
- Traducir código de alto nivel a ensamblador
- No fue el primer intento pero si el primero exitoso

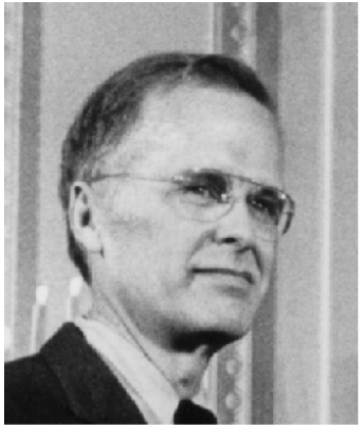
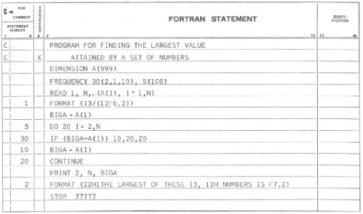


Figura: John Backus

FORTRAN I

- 1954-57 - El proyecto FORTRAN I
- 1958 - 50 % de software en el mundo escrito en Fortran
- El tiempo de desarrollo se redujo y el desempeño era equiparable a ensamblador



C			FORTRAN STATEMENT
C			PROGRAM FOR FINDING THE LARGEST VALUE
C		X	ATTAINED BY A SET OF NUMBERS
			DIMENSION A(999)
			FREQUENCY 30(2,1,10), 5(100)
			READ 1, N, (A(I), I=1,N)
	1		FORMAT (I3/(12F6.2))
			BIGA = A(1)
	5		DO 20 I = 2,N
	30		IF (BIGA-A(I)) 10,20,20
	10		BIGA = A(I)
	20		CONTINUE
			PRINT 2, N, BIGA
	2		FORMAT (22H1THE LARGEST OF THESE I3, 12H NUMBERS IS F7.2)
			STOP 77777

Figura: Fortran

FORTRAN I

- El más importante compilador en CS
- La mayoría de compiladores preserva muchas teorías creadas por FORTRAN
- ¿Compiladores modernos?

COMPILADORES

LA ESTRUCTURA DE UN COMPILADOR

1. Análisis léxico - identificar palabras
2. Análisis sintáctico - Identificar oraciones
3. Análisis semántico - Analizar oraciones
4. Optimización - Del código
5. Generación - Traducción

ANÁLISIS LÉXICO

Primer paso: Reconocer palabras
- La unidad más pequeña luego de las letras

Esta es una oración.

ANÁLISIS LÉXICO

Ejemplo: taes se anun nociora

ANÁLISIS LÉXICO

Los analizadores léxicos dividen los programas en “palabras“ o “tokens“

if x == y then z = 1; else z =2;

ANÁLISIS SINTÁCTICO

1. Entender la estructura de la oración
2. Parsing = Brindar estructura a una oración (Generalmente un arbol)

ANÁLISIS SINTÁCTICO

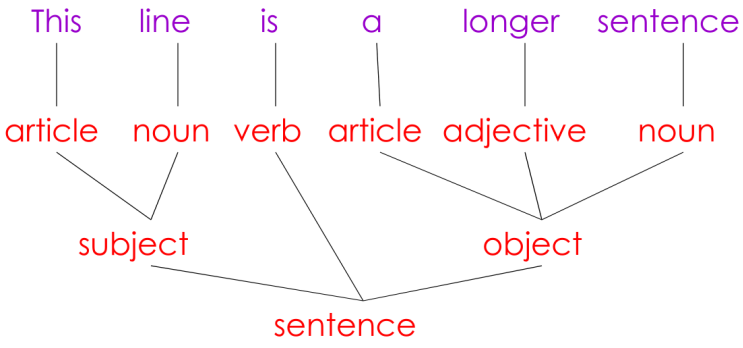


Figura: Parsing

ANÁLISIS SINTÁCTICO

Consideremos:

if x == y then z = 1; else z =2;

Diagrama:

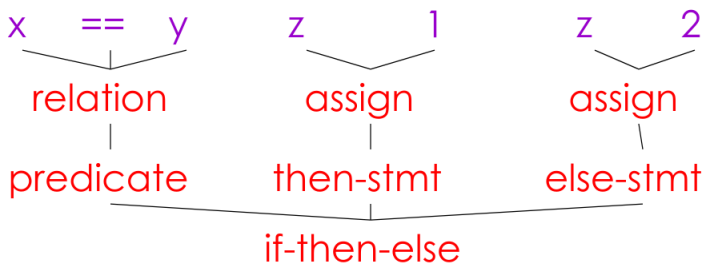


Figura: Parsing

ANÁLISIS SEMÁNTICO

1. Una vez entendida la estructura de la oración necesitamos entender su significado
2. Detección de inconsistencias

ANÁLISIS SEMÁNTICO

Ejemplo:

Juan dijo que José dejo su tarea en casa

¿Quien la dejó en casa?

Aun peor:

Juan dijo que Juan dejo su tarea en casa

¿El multiverso de los Juan? ¿Cuantos Juan son?

ANÁLISIS SEMÁNTICO

```
{
    int juan = 3;
    {
        int juan = 4;
        cout << juan;
    }
}
```

ANÁLISIS SEMÁNTICO

Los compiladores ejecutan diversas verificaciones semánticas, por ejemplo verificación de tipos:

Ella es Abelardo Perez

"Type mismatch" entre ella y Abelardo

OPTIMIZACIÓN

- Similar a la edición
  - Minimizar el tiempo de lectura
  - Minimizar los elementos que el lector debe conservar a corto plazo memoria
- Modificar automáticamente los programas para que
  - Corran más rápido
  - Usar menos memoria
  - En general, para usar o conservar algún recurso

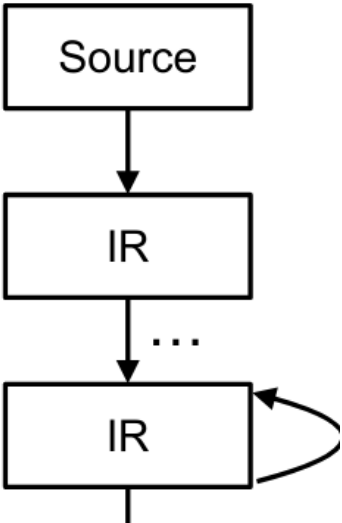
OPTIMIZACIÓN

Ejemplo

```
X = Y *0
..
X = 0
```

GENERACIÓN DE CÓDIGO

- Al día de hoy muchos compiladores generan representaciones intermedias
- En cada nivel se reduce la abstracción



GRACIAS