



Compiladores e Intérpretes

Apuntes de la clase del día miércoles 8 de Marzo del 2017

Apuntador:

Jonathan Rodríguez Martínez

2014015749

Profesor:

Dr. Francisco Torres Rojas

I Semestre 2017

Índice

<u>Quiz y repaso</u>	3
<u>Fases de la compilación</u>	4
<u>Estructura básica de compilación</u>	4
<u>Scanner</u>	5
<u>Parser</u>	6
<u>Análisis semántico</u>	8
<u>Optimizador de fuente</u>	9
<u>Generador de código</u>	12
<u>Optimizador de código</u>	14
<u>Estructuras de datos</u>	16
<u>Token</u>	16
<u>Árbol sintáctico</u>	16
<u>Tabla de símbolos</u>	16
<u>Tabla de literales</u>	17
<u>Código intermedio</u>	17
<u>Archivos temporales</u>	17
<u>Recordatorios</u>	17

Quiz

1) Defina detalladamente los siguientes conceptos:

- a) Preprocesador
- b) Linker
- c) Linking Dinámico
- d) Loader
- e) Relocalización
- f) Profiler
- g) Debugger
- h) Análisis
- i) Backpatching
- j) Front End

Comentarios y repaso de la clase anterior

Se inició la clase definiendo lo siguiente:

- a) Relocalización: Corregir las posiciones de memoria, ya que estas tienen direcciones arbitrarias (cero) y puedan ser cargadas a memoria y se pueda ejecutar correctamente, esta acción es realizada por el linker y el loader.

También se repasaron los conceptos:

- a) **Front End:** (Todos respondimos) Es el análisis, y se relaciona con el código fuente, da como resultado una representación intermedia.
- b) **Back End:** Es la síntesis, de la representación intermedia al lenguaje objetivo.

Esta era una pequeña división, ahora partámosla no por módulo, sino más detallado (A continuación).

PD: El capítulo de la tarea larga se evaluará en el examen.

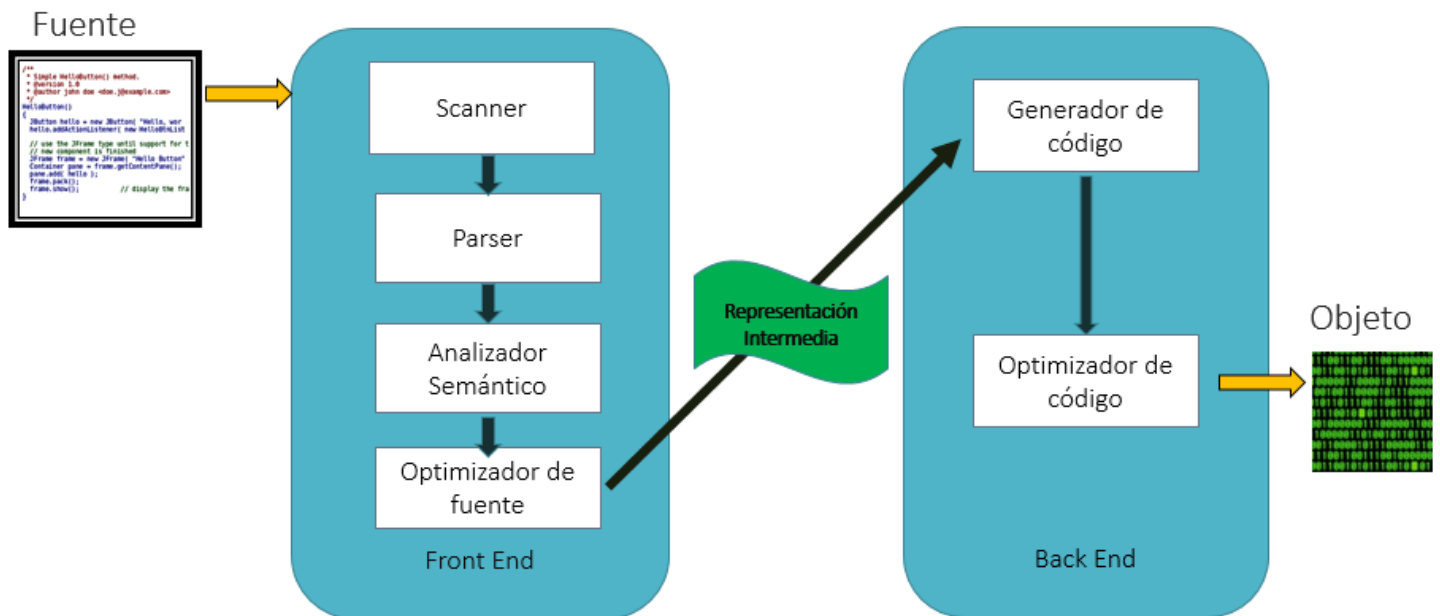
Fases de la compilación

Pasadas:

- Es una lectura completa del fuente, de inicio a fin.
- “N pasadas” significa que lee “N” veces del fuente.
- Algunas definiciones pueden estar después de su uso, por lo cual se puede hacer varias pasadas o una pasada haciendo uso de batpaching.
- Más pasadas significa más flexibilidad, pero también más tiempo de compilación.

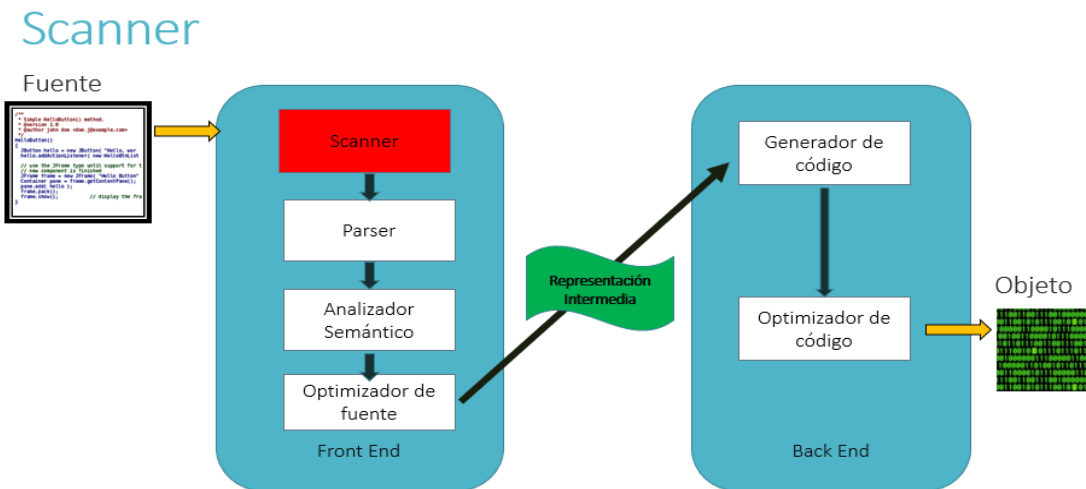
Estructura básica de un compilador

Estructura Básica de compilación



Vamos a ver rápidamente cada una de estas partes.

Scanner

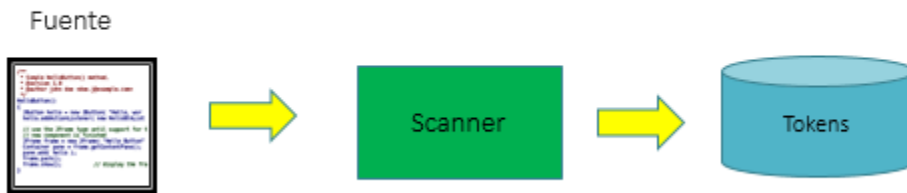


- Nombre Técnico: “Análisis Léxico”
 - Léxico: (La clase no sabía) Conjunto de palabras de un idioma.
 - Si léxico termina en “ón,ón,ón”, es un léxico muy pobre.
- Toma fuente de entrada y lo descompone en unidades léxicas mínimas.
- Conocido en el bajo mundo como scanner.
- Implementados con autómatas finitos.
- Tokens (más adelante formalizaremos este concepto)
- Hay herramientas que generan la mayor parte del scanner.

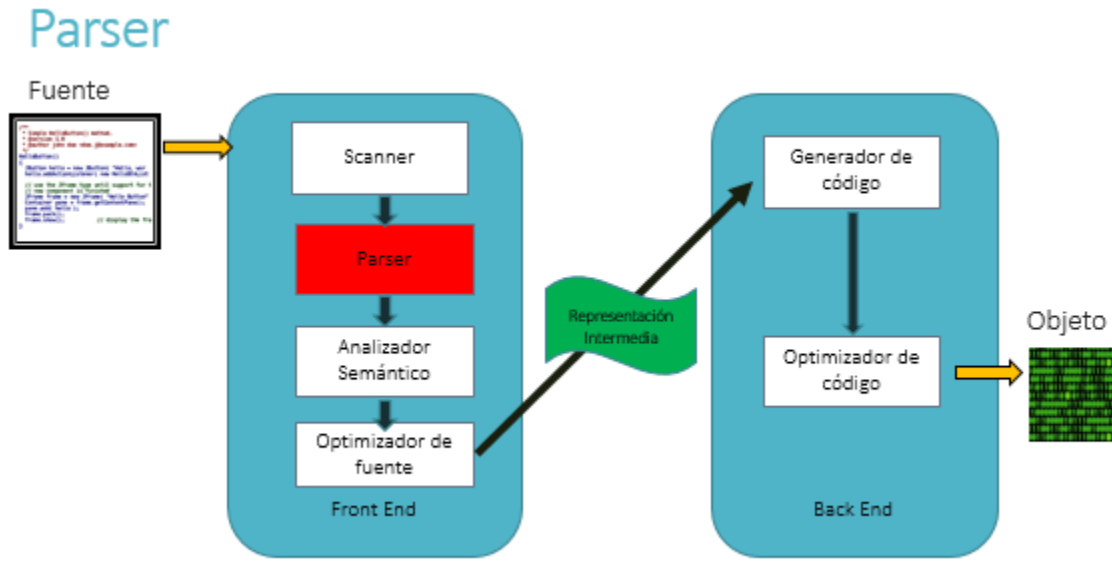
Ejemplo:

```
Burbuja ( N entradas, vector [ ] )
{
    int indice, j, aux;
    for (indice=0; indice < N entradas; indice++){
        for (j=indice + 1; j < n; j++){
            if (j = indice[j] < vector[indice])
            {
                aux = vector[j];
                vector[j] = vector[indice];
                vector[indice] = aux;
            }
        }
    }
}; palabras reservadas separadores operadores literales
```

Entradas - Salidas

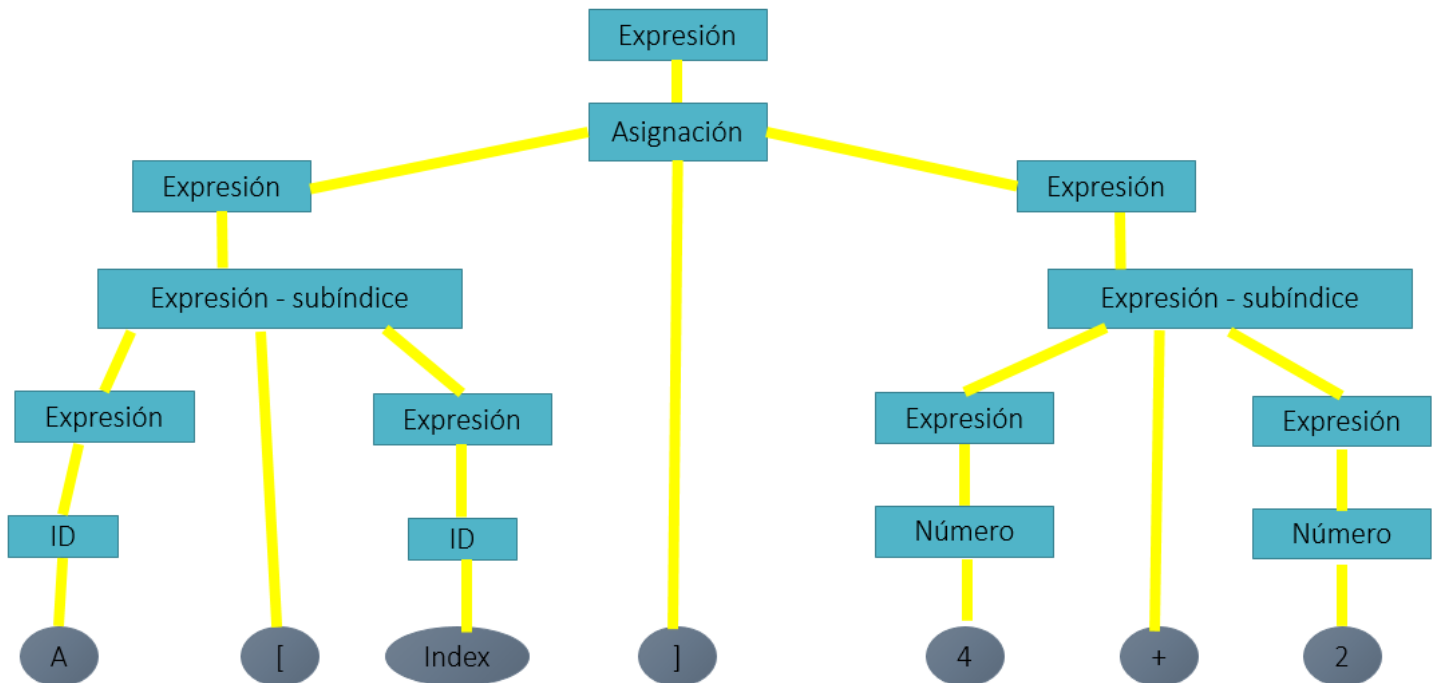


Parser:



- Nombre Técnico: “Análisis sintáctico”
 - ¿Qué es sintaxis? Es ver que las palabras puestas en una estructura (palabra muy buena) sea correcta en un lenguaje.
- Revisa que la sintaxis del programa este correcto
- Conocido en el bajo mundo como “Parser”
- Toma secuencia de tokens y verifica que corresponda a algo válido.
- Implementados con “push-down automata”
- Hay herramientas que generan mayor parte de un parser.

Análisis sintáctico ejemplo

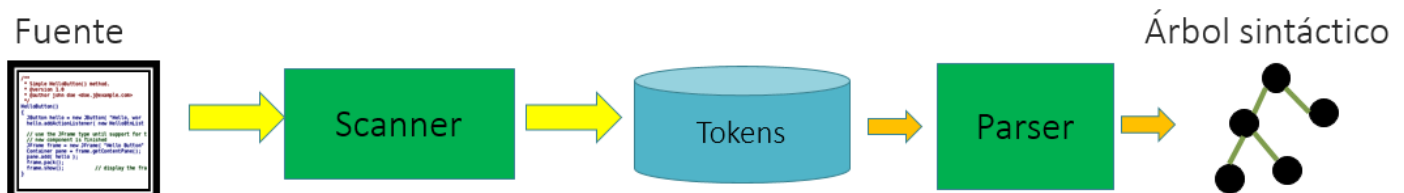


- Parser en forma implícita construye árbol.

Entrada - Salida

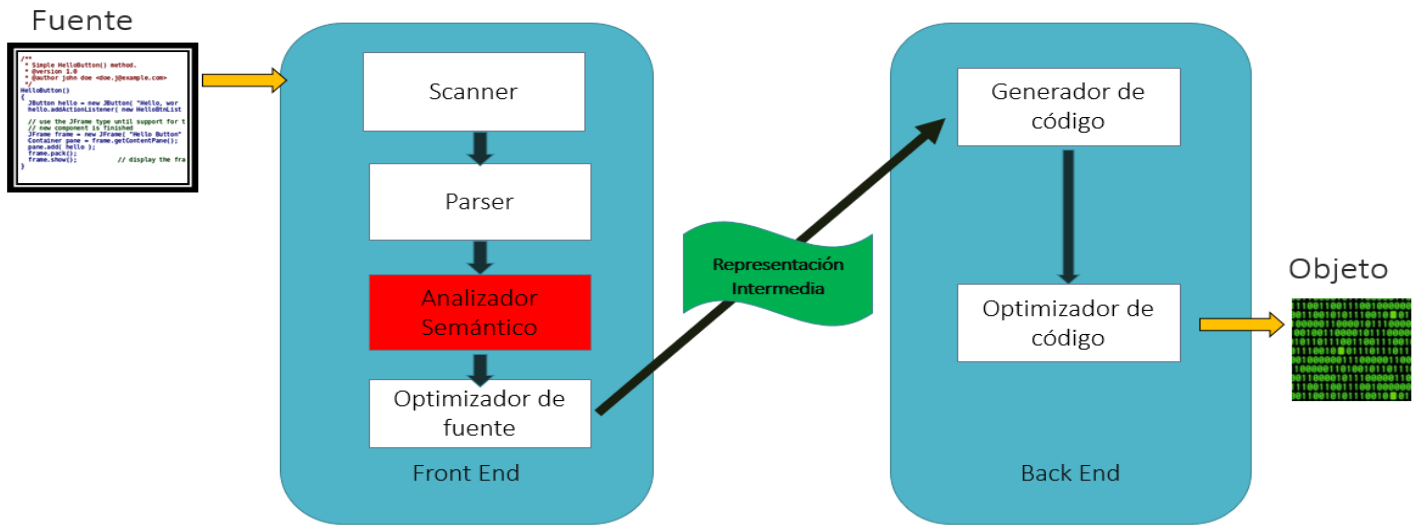


Todo Completo

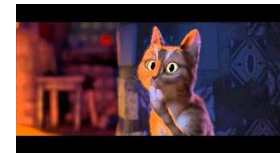


Análisis semántico

Analizador semántico

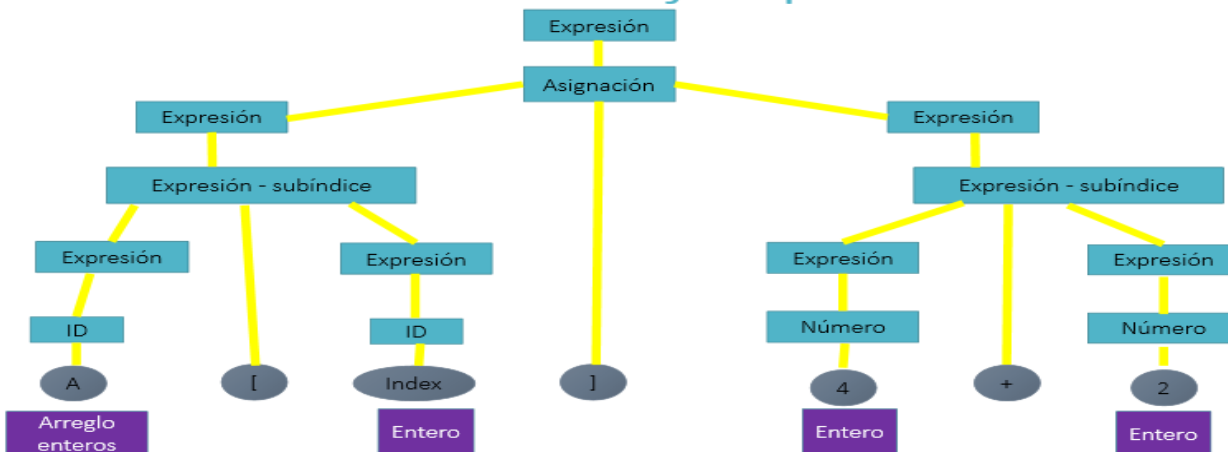


- ¿Qué es semántica? Es el significado de algo.
- Significado del programa
- Revisar si tiene sentido lo que se le pide que haga
- Determina su comportamiento en tiempo de corrida
- Algo sin declarar es un error semántico.



- Semántica estática: Se realiza cuando se analiza la semántica viendo el código fuente y deducir que es lo que va a pasar.
- Construye tabla de símbolos
- Le pone semántica al árbol sintáctico: Árbol de atributos o árbol decorado.

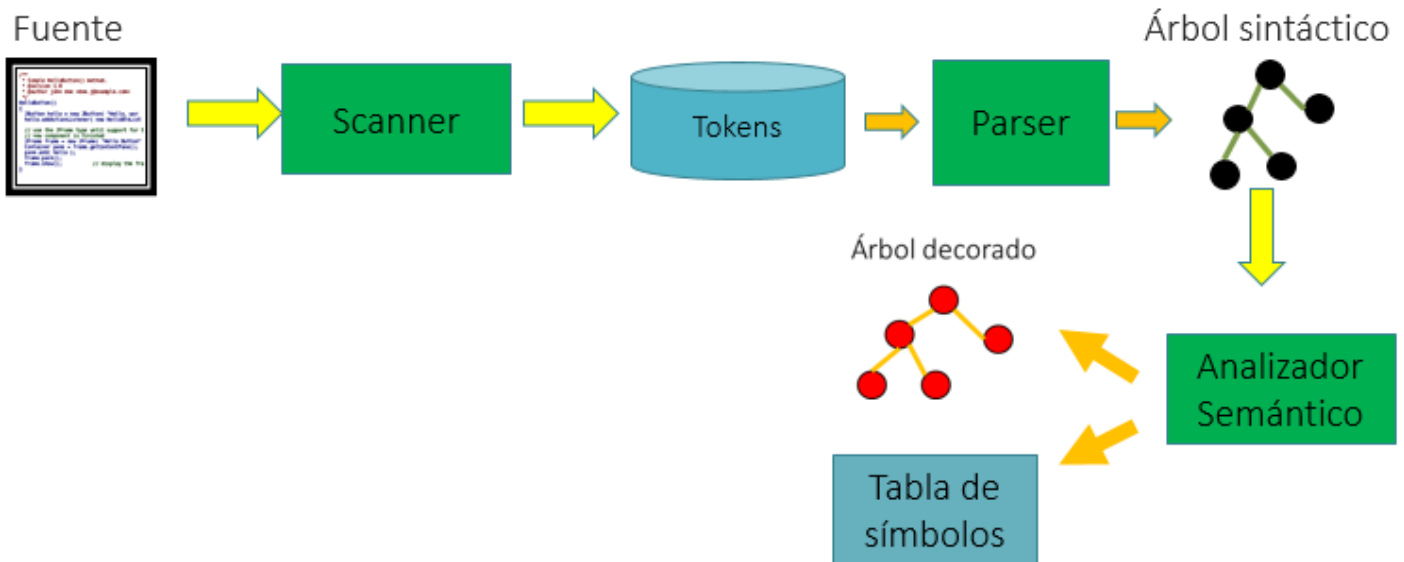
Análisis semántico ejemplo



Entradas - Salidas



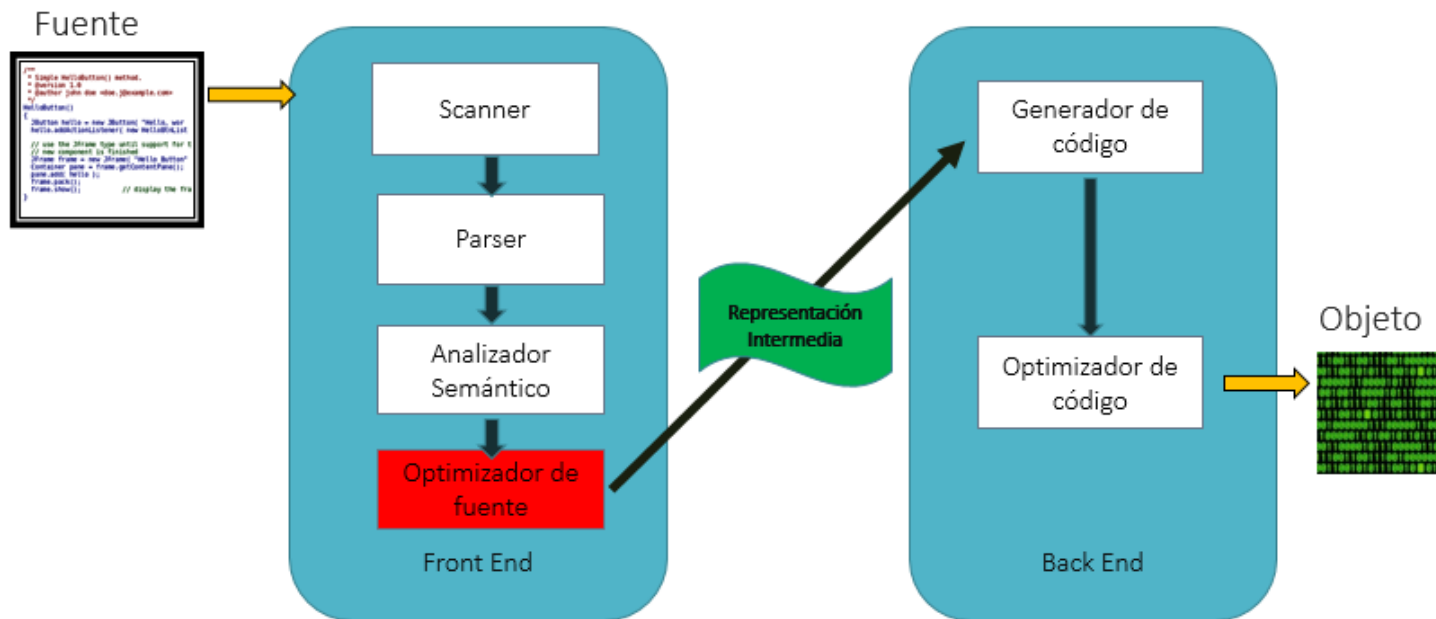
Todo Completo



Optimizador de Fuente

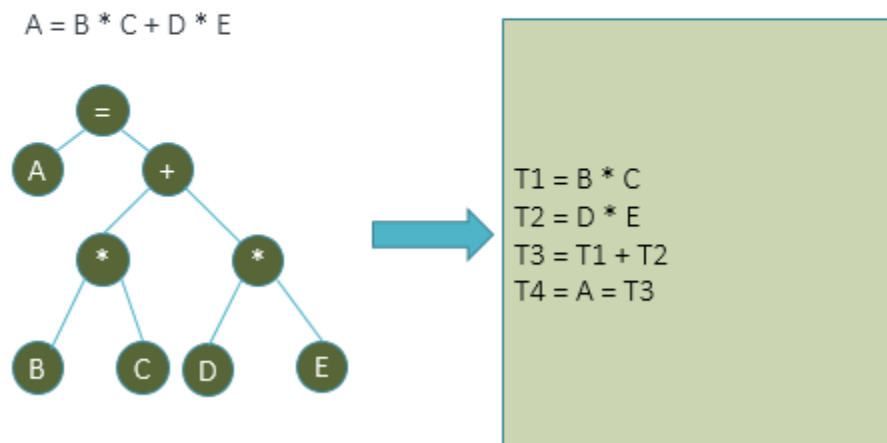
- ¿Qué es óptimo? Superlativo de mejor.
 - ¿Más óptimo?
 - Optimizar algo, ¿hay algo mejor?
 - Se emplea de manera incorrecta, por mencionar un ejemplo del profe: “Ojalá algún día los costarricenses tengamos la trocha que merecemos”.
 - PD: no utilizar esas frases “más óptimo”, es tan feo como “más mejor”

Optimizador de fuente

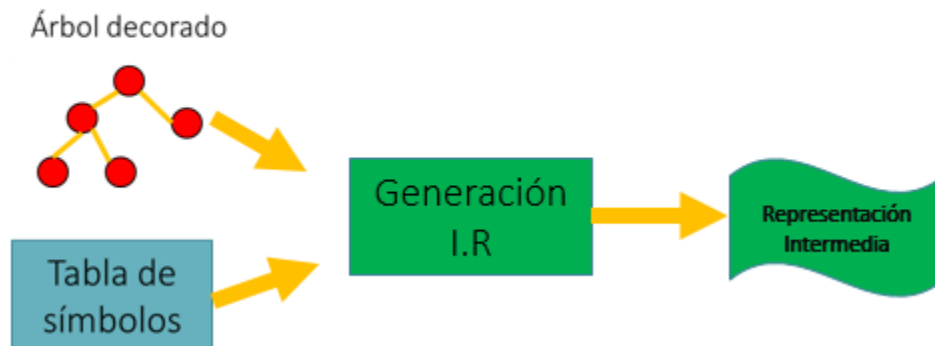


- Introduce optimizaciones
- Cálculo de constantes – propagación de constantes
- Eliminar código innecesario
- Se puede hacer sobre árbol anotado
- Es fácil convertir árbol en estructura lineal
- Similar a ensamblador de tres direcciones
- Responsable de escupir código intermedio

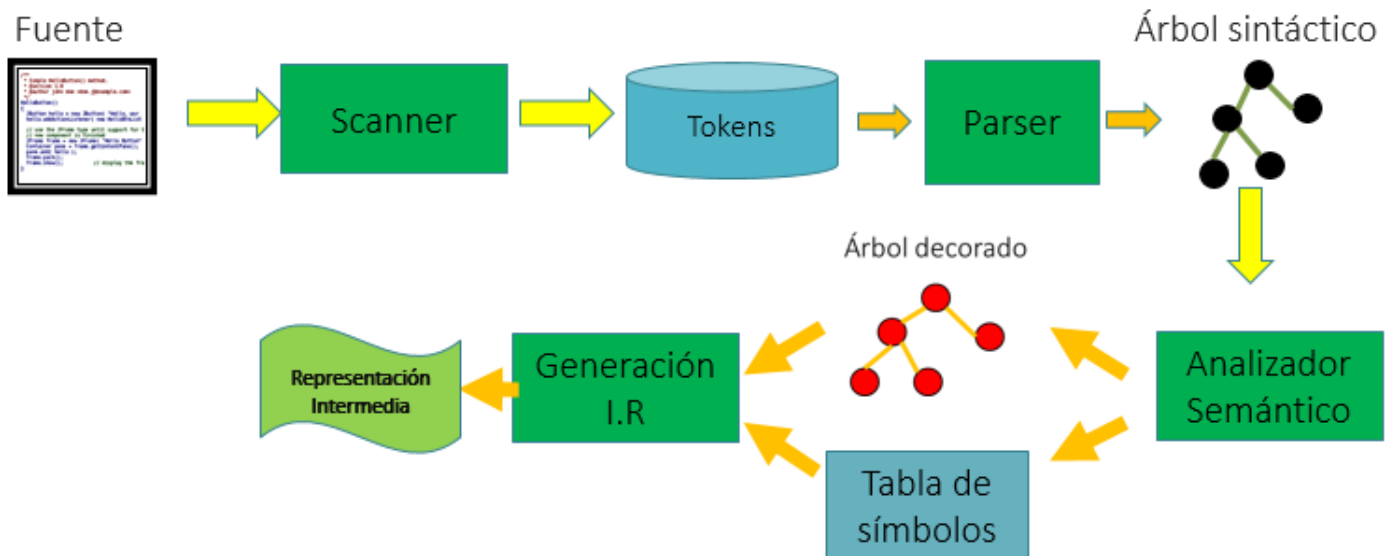
Código Intermedio - ejemplo



Entradas – Salidas

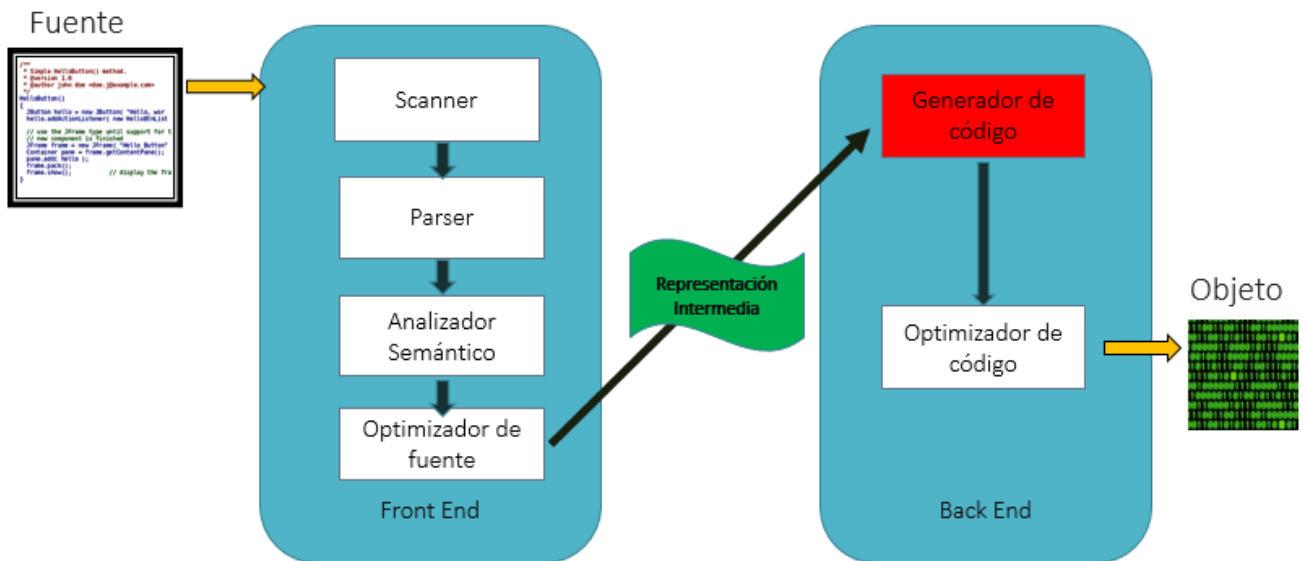


Con su amigos



Generador de código

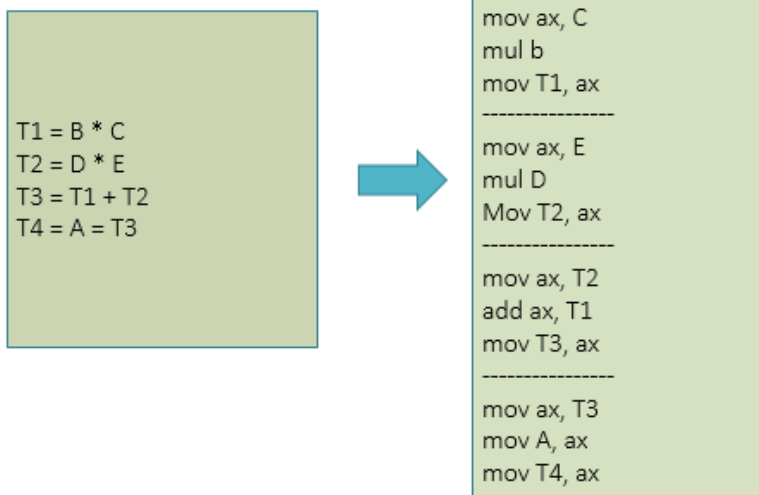
Generación de código



- Vive en el back end del compilador
- Convierte el código intermedio a código real.
- ¿Ensamblador? ¿Máquina? No debería ser problema.
- Representación es importante
 - Cantidad de bytes
 - Restricciones de colocación
- Hay que conocer las posibilidades de la arquitectura

Ejemplo Generación de código

$A = B * C + D * E$

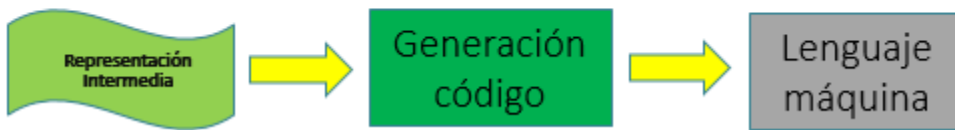




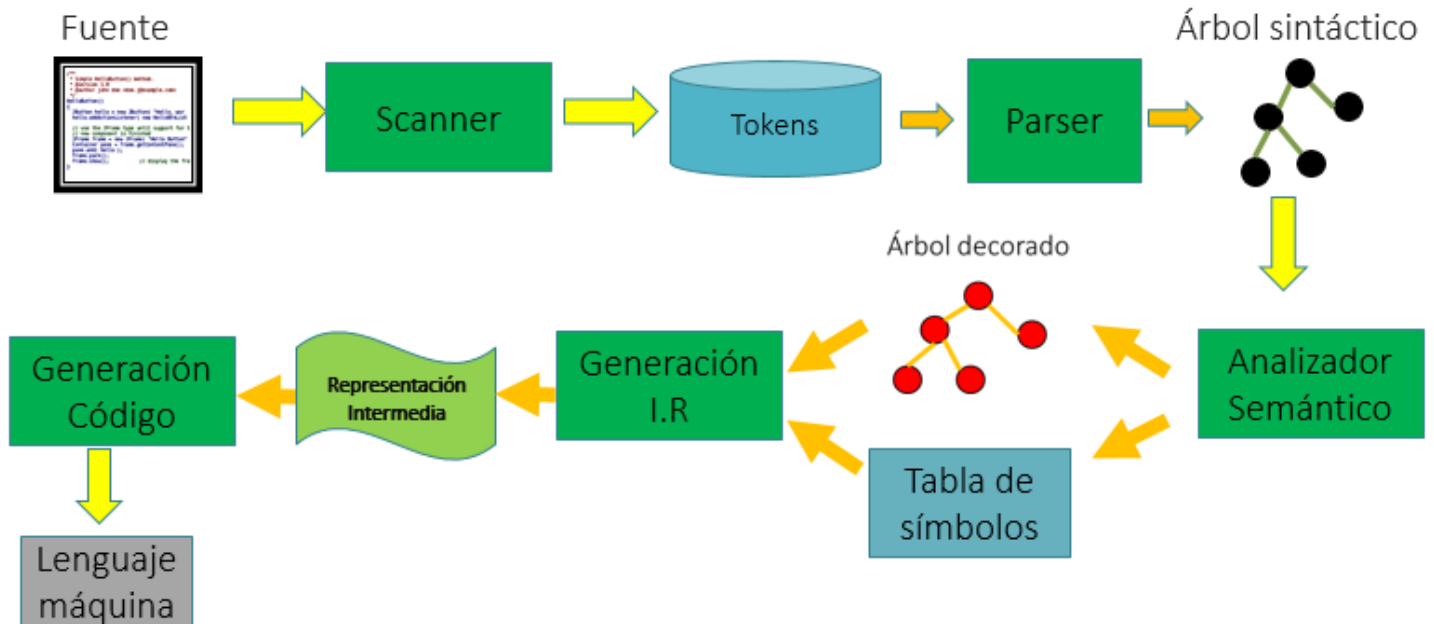
El código anterior, ¿es demasiado óptimo?

No necesariamente buen código, y de eso estaba consiente Backus, muy lento, muy purete.

Entradas - Salidas

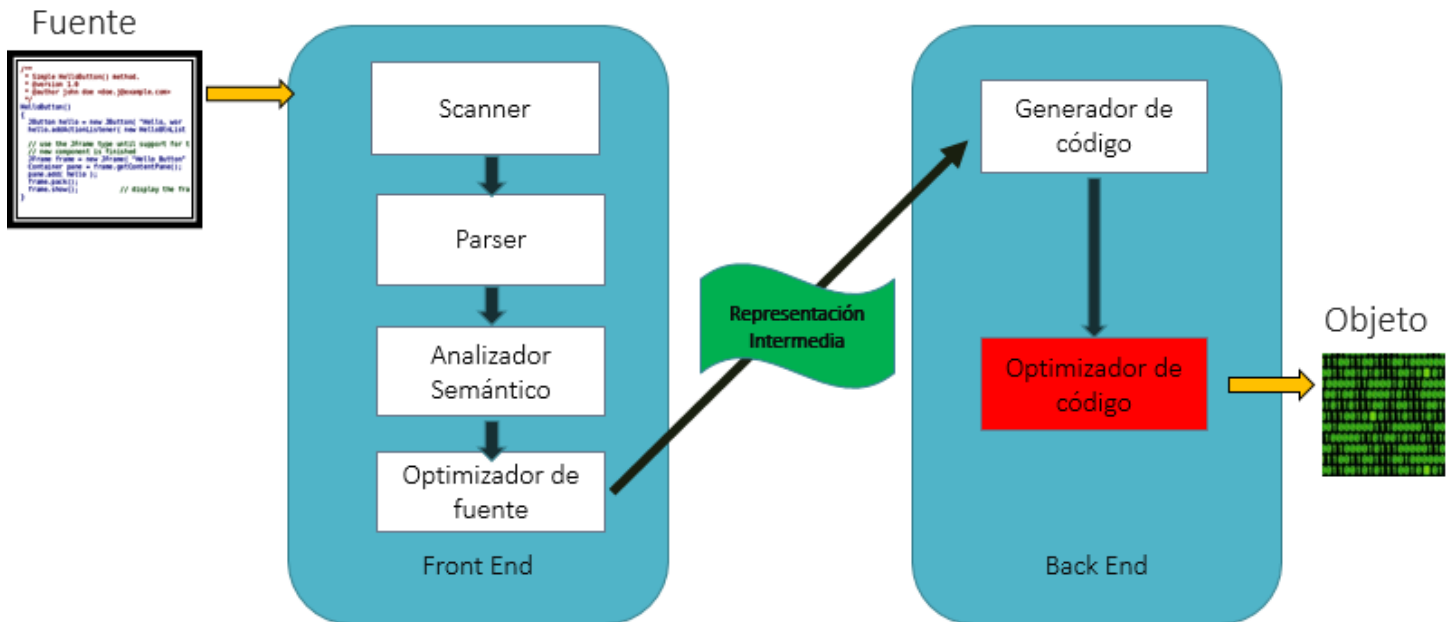


Con su amigos



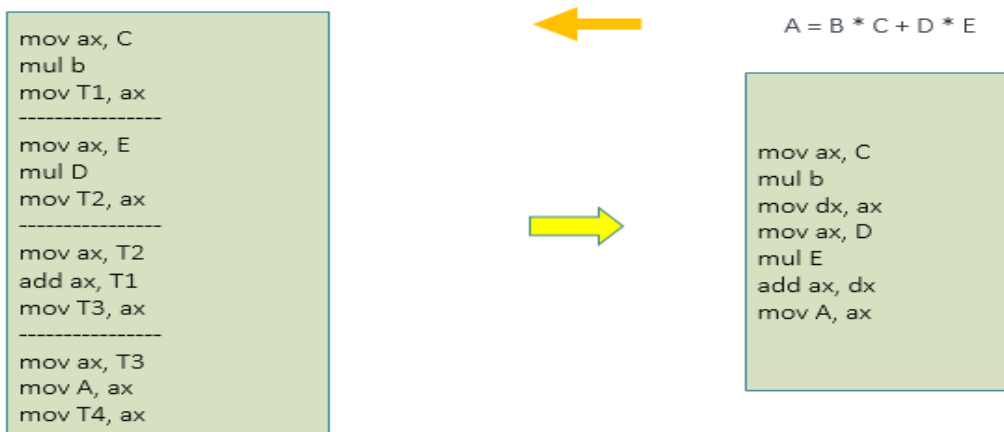
Optimizador de Código

Optimizador de código

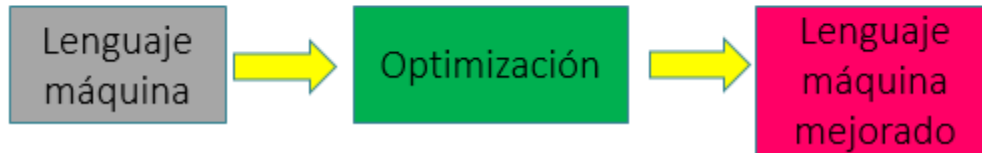


- Mejora el código generado en el paso previo.
- Cambia las instrucciones equivalentes pero más rápidas.
 - Por ejemplo: Si hay una multiplicación por dos, hacer mejor un shift izquierda
- Modos de direccionamiento apropiado
- Eliminar código redundante
- Eliminar código innecesario
- Optimizador para memoria y espacio

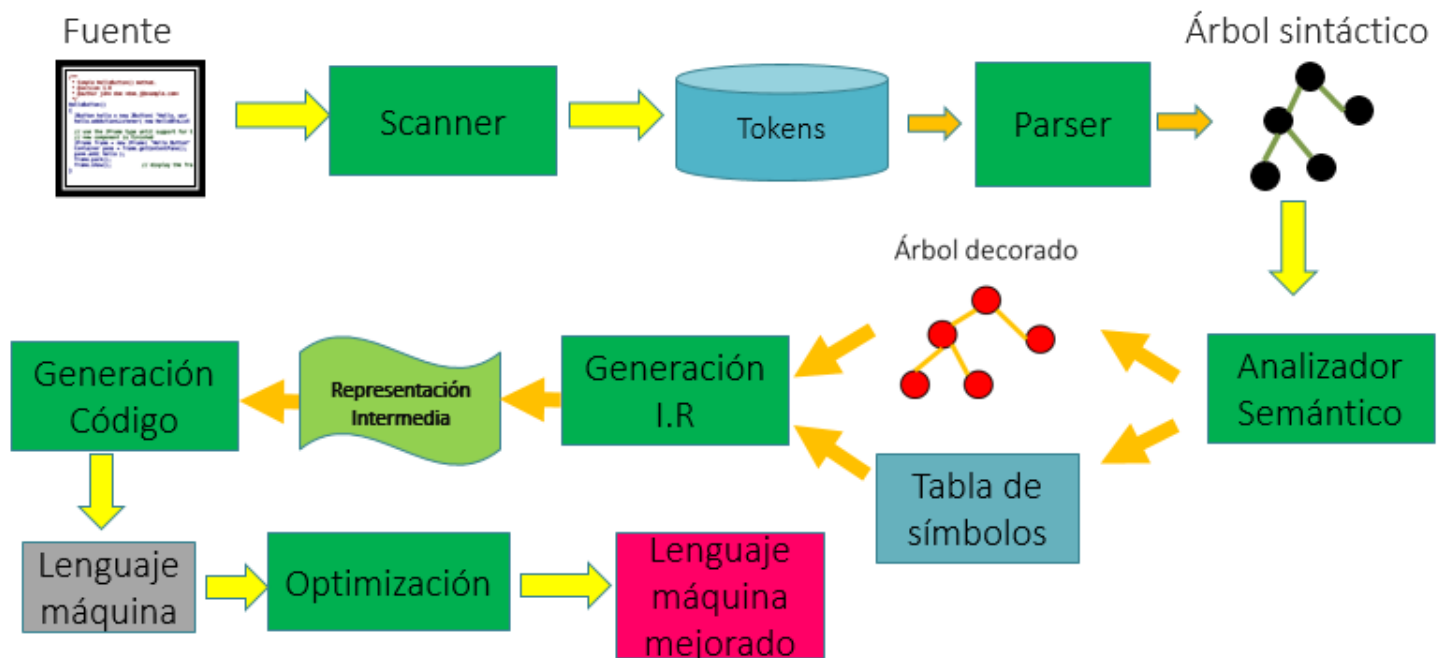
Ejemplo optimización de código



Entradas - Salidas



Con su amigos



Ok todo lo que describimos anteriormente es mentira.

Todos esos personajes existen en un compilador, pero no se ejecutan secuencialmente, todos corren al mismo tiempo. Lo único **cierto** es que entra un código fuente y sale un código máquina mejorado.

Estructuras de datos

A continuación vamos a estudiar las estructuras de datos presentes en un compilador.

Token:

- Significado de token: símbolo.
- Regresados por el scanner
- Usualmente dos campos
 - Código de token (es un numero)
 - ❖ Variable
 - ❖ Enteros – constantes
 - ❖ Parentesis (Derecho e izquierdo)
 - ❖ Coma
 - Hilera específica (llamada también lexema) PD: suena a enfermedad
- La diferencia entre lexema y token es que por ejemplo puedo definir:
Contador = 1; donde la familia de token es “variable” y el lexema (específica que escribió programador) es “Contador”.

Árbol sintáctico:


- Creado por el parser.
 - Decorado por el analizador semántico
 - Árbol en memoria dinámica
 - Hay variable que señala a raíz.
 - Estructuras con campos llenados por el parser y analizador semántico.
 - Podrían ser punteros a otras estructuras.
 - Pueden ser una pesadilla o un orgasmo, según de cuanto le gusten las estructuras de datos (para el profe es un orgasmo).
- 
- A veces no existe explícitamente.

Tabla de símbolos:

- Cualquier símbolo dentro de nuestro programa va a aparecer aquí
- Conserva información, asocia identificadores:
 - Variables
 - Procedimientos y funciones
 - Tipos de datos
- Interactúa con scanner, parser, analizador semántico...
- Estructura más usada en la compilación
- Acceso por medio de hash

Tabla de literales

- Constantes (números, hileras)
- Su valor no cambia en ejecución
- Deben residir en memoria

Código Intermedio



- Hay que representarlo de algún modo
- Muchas posibilidades
 - Punteros a hileras
 - Archivos Temporales
 - Listas enlazadas
- La optimización puede organizar

Archivos temporales

- Nuestros amigos



- Archivo de vida corta
- Crear archivo
 - Grabar en el archivo
 - Leer del archivo
 - Borrar el archivo
- Típico que los compiladores los utilicen
- Ayudan a resolver problemas complejos.

Recordatorios

- **Ir trabajando en el proyecto 1**
- **En el proyecto loops opcionales porque es difícil**
- **No C C ☹**
- **Preguntas en el foro**



DEFINE