

Apuntes 8 de marzo de 2017.

Izcar Muñoz Torrez.



Instituto Tecnológico de Costa Rica

**Escuela de Ingeniería en Computación
Compiladores e Interpretes**

Grupo 40

Apuntes del 8 de marzo de 2017

Elaborado por:

Izcar Muñoz Torrez 2015069773

Profesor:

Dr. Francisco Torres Rojas

I Semestre, 2017

Contenidos

Quiz #4	3
Repaso de la clase del viernes 3 de marzo.	3
Fases de la compilación	4
Analizador Léxico (conocido como Scanner).....	5
Análisis Sintáctico (Conocido en el bajo mundo como Parser)	6
Analizador Semántico(no tiene un nickname, pero asigno “Semantec”).....	8
Optimizador de Fuente.	10
Generador de Código.	12
Optimizador de código.	13
Estructuras de datos.	15
Token.....	15
Árbol sintáctico.	15
Tabla de símbolos.	15
Tabla de literales.	16
Código intermedio.....	16
Archivos temporales.....	16
Preguntas sobre el proyecto y recordatorios.	17

Quiz #4

- 1) Defina detalladamente los siguientes conceptos:
 - a) Preproceso.
 - b) Linker.
 - c) Linker dinámico.
 - d) Loader.
 - e) Relocalización.
 - f) Profiler.
 - g) Debugger.
 - h) Análisis.
 - i) Backpatching.
 - j) Front End.

Repaso de la clase del viernes 3 de marzo.

Se habló del Loader y Linker.

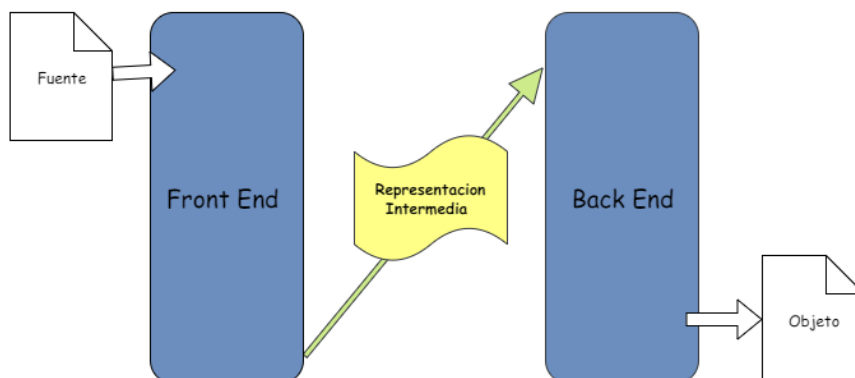
- Loader: su función es tomar un ejecutable y cargarlo a memoria. Se encarga de negociar el espacio.
- Linker: su función es agregar bibliotecas.

El capítulo (Ch7- The Assembly Language Level) para la tarea larga obviamente es materia de examen.



¡También huele ya a examen!

Estructura Básica del compilador.



Apuntes 8 de marzo de 2017.

Izcar Muñoz Torrez.

- Front End: Relacionado más que todo con Análisis, o sea con el “Lenguaje Fuente”.
- Back End: Relacionado con la representación intermedia y el “Lenguaje Objeto”.

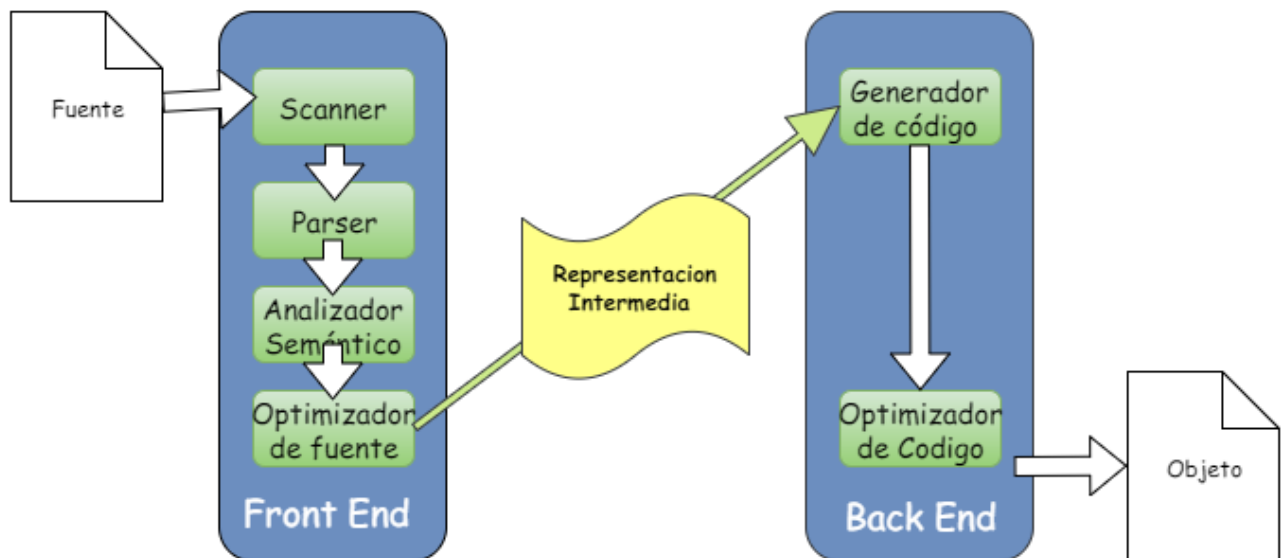
Ahora esta estructura la partiremos en módulos más detallados.

Nota: [Dudas al final de la clase sobre el proyecto 1](#). Para efectos de este documento será en las últimas páginas que se responderán las dudas.

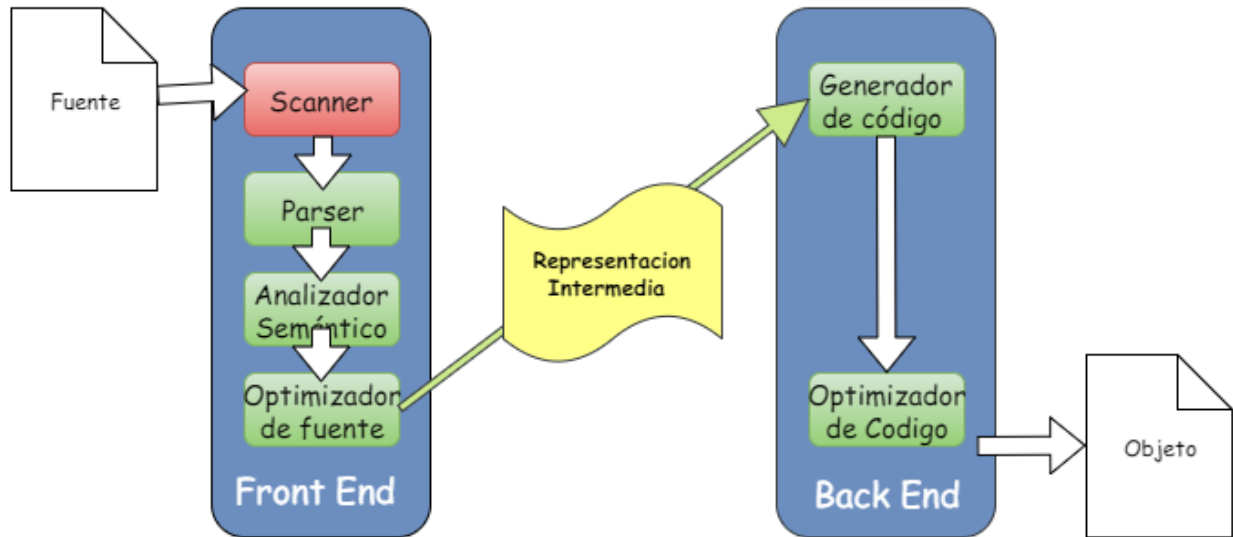
Fases de la compilación

- **Pasada:**
 - Lectura completa y conjunta del archivo fuente de inicio a fin.
 - Un archivo se puede leer **N** veces.
 - Algunos símbolos, etiquetas pueden estar definidos después de que fueron usados o reenviados por primera vez.
 - Entre más pasadas hay más flexibilidad, pero va a tardar más el proceso de compilación.
 - Formas de resolver problemas con unas pasadas demás.

Ahora refinaremos la estructura básica del compilador.



Se hablará de cada uno de los nuevos módulos brevemente, y durante el semestre se le dedicará semanas y semanas para cada uno.



El primero en aparecer es el **Scanner**.

Analizador Léxico¹ (conocido como Scanner)

- Primera fase de compilación.
- Analizan las palabras que son parte del lenguaje fuente.
- Descompone el fuente en “Unidades Léxicas Mínimas”.
 - Tokens.
- Se puede implementar con “Autómatas de Estados Finitos”.
- Hay herramientas que genera la mayor parte del SCANNER.
- Tiene mucha teoría y muy sólida.



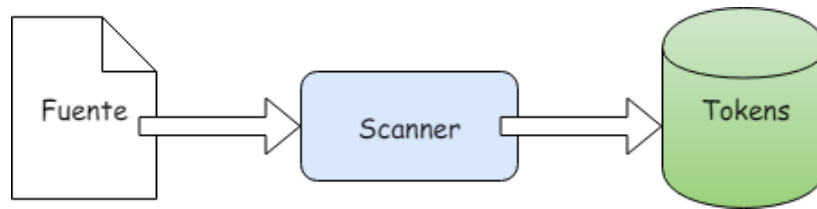
```
Burbuja N_entradas vector |
|
|
| indice, aux;
| indice = 0; indice < N_entradas; indice++
| | = indice + 1; | ++
| vector < vector[indice]
| aux = vector[]
| vector[] = vector[indice]
| vector[indice] = aux
|
|
|
|
|
```

- Verde -> identificadores, variables.
- Azul -> separadores.
- Rojo -> palabras reservadas.
- Violeta -> operadores.
- Amarillo -> literales.
- El proyecto es esto con colorcito :3.

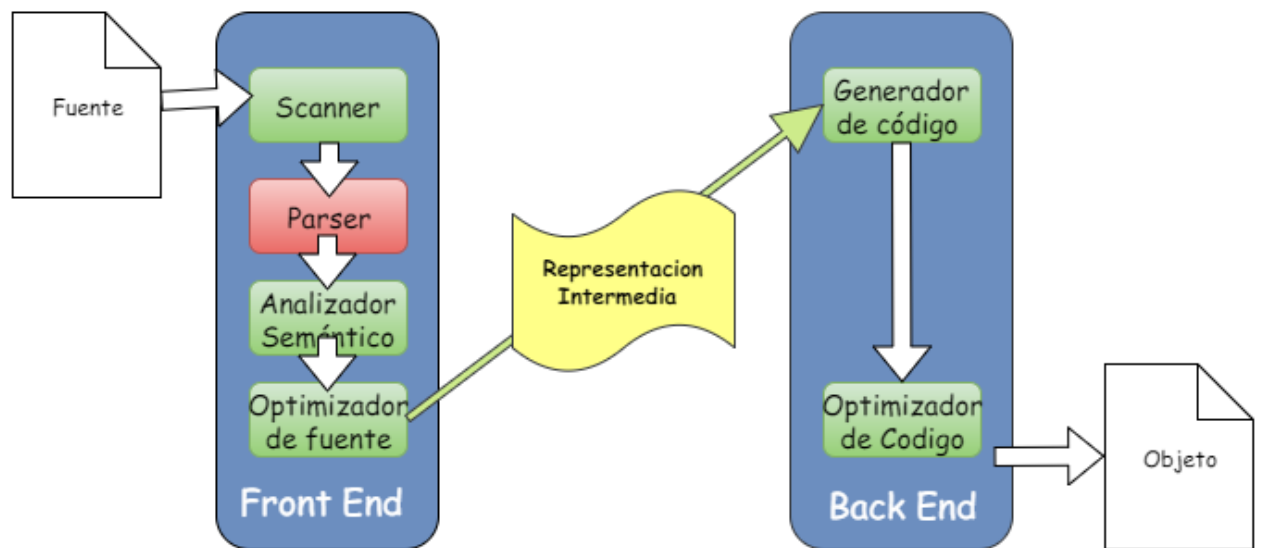
¹ Léxico: conjunto de palabras de un determinado lenguaje(Vocabulario).
No tenemos buen léxico.

- **Entrada y salida del Scanner.**

- Recibe el fuente, lo procesa y supongamos que nos devuelve una secuencia o lista de un montón de tokens.



El Segundo en aparecer ahora es el Parser.



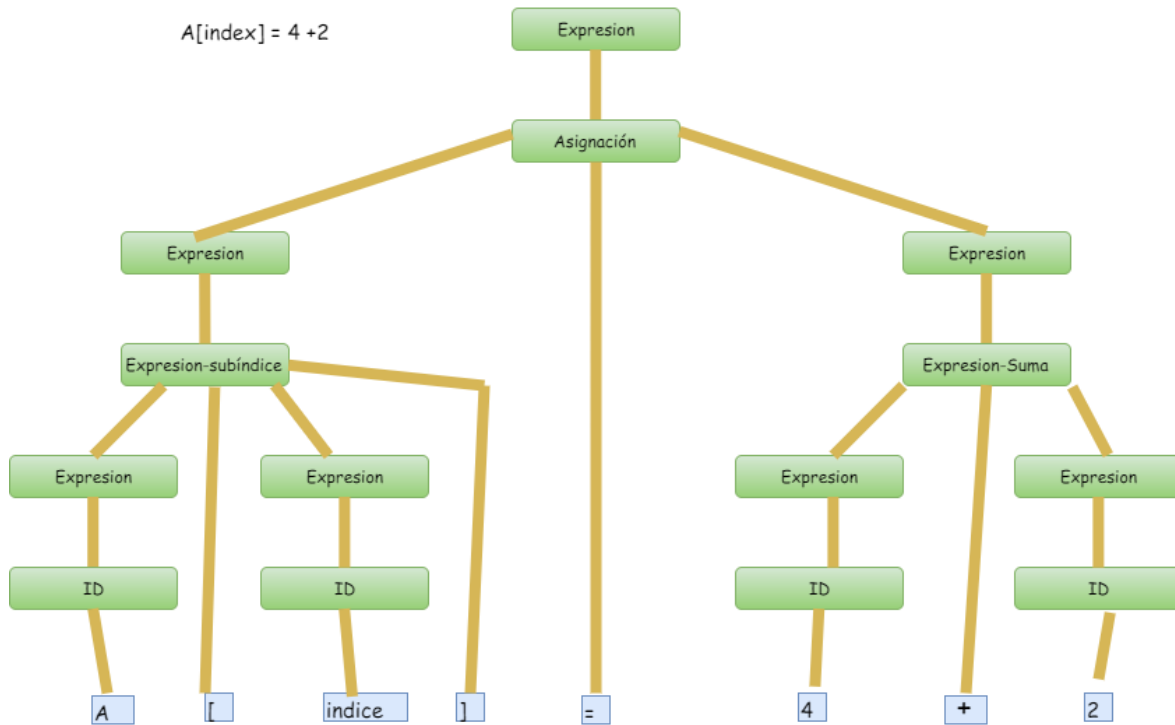
Análisis Sintáctico² (Conocido en el bajo mundo como Parser)

- Revisa que la sintaxis de un programa sea correcta.
- Toma todos los tokens que salieron del Scanner y verifica que corresponda a una estructura gramatical valida.
- Requiere de una gramática.
- Implementados con “**Push down Autómata**”.
- Para el proyecto siguiente hay herramientas que generan la mayor parte del **Parser**.(Se van a poder usar dependiendo del humor que ande el profe 😊).

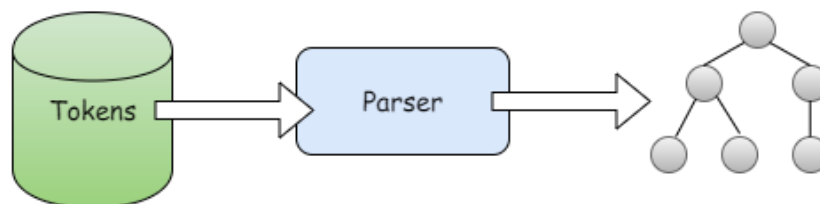
² Sintaxis: Orden correcto de las palabras dentro de una estructura.

Ejemplo.

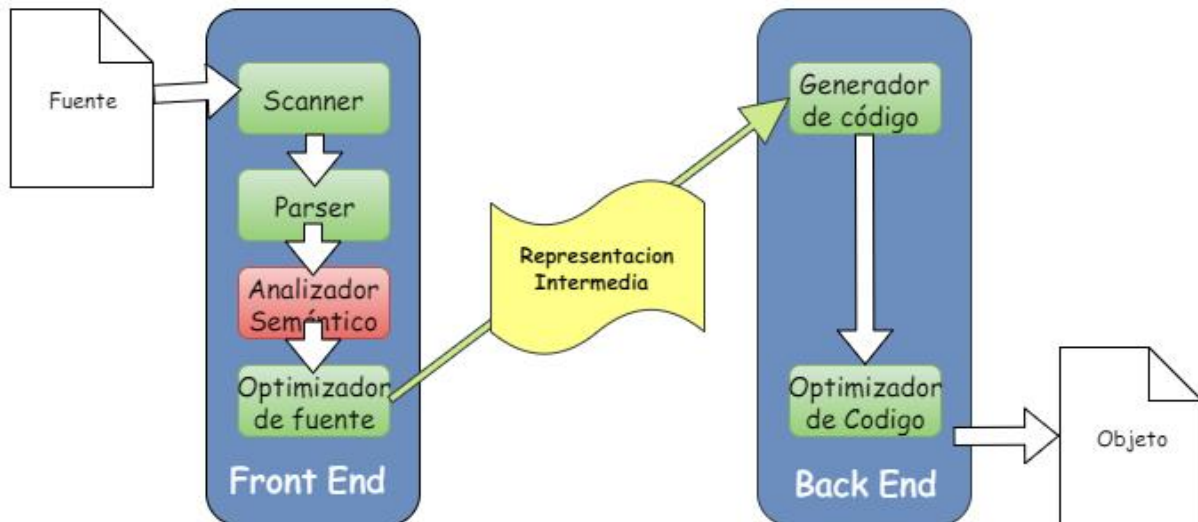
- $A[\text{index}] = 4 + 2$



- El Parser construye implícita o explícitamente un árbol, donde va identificando los componentes del código.
- En el **proyecto 0** el árbol se iba formando implícitamente.
- **Entrada y salida del Parser.**
 - Recibe un montón de tokens, el Parser los procesa y de salida da un **árbol sintáctico**.



Ahora aparece el Analizador Semántico.



Analizador Semántico³(no tiene un nickname, pero asigno "Semantec")

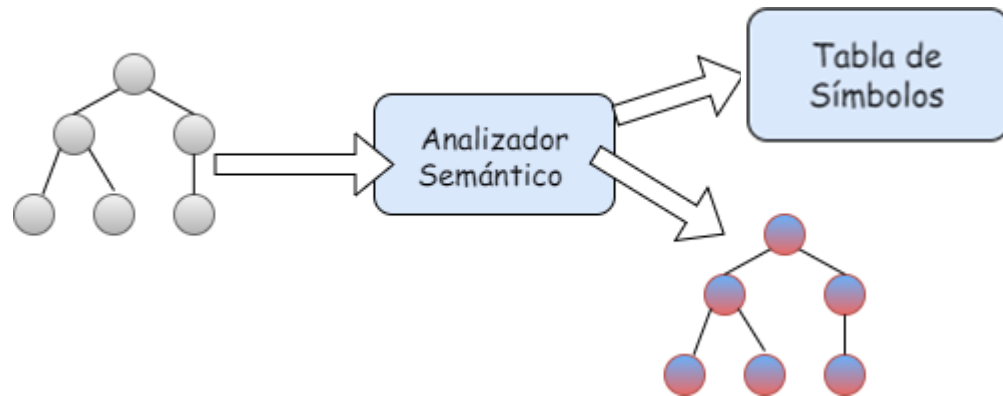
- Significado del programa.
- Revisa si tiene sentido lo que se le pide se haga.
- Determina su comportamiento en tiempo de corrida.
- Puede pasar el caso de que algo sea sintácticamente correcto, pero semánticamente equivocado.
 - 10.333 + "Hola";
Sintácticamente está bien, ya que el Parser no ve nada malo pero el Analizador Semántico se da cuenta que al tipo **hilara** no tiene sentido sumarle un **punto flotando**. Por lo tanto, es un error semántico (dependiendo del lenguaje).
- Un Parser no tiene forma de saber si una variable ha sido o no declarada. Esto se hace con trucos semánticos.
- **Semántica estática**: cuando se analiza el programa viendo fuente para deducir que es lo que va a pasar.
- Revisa declaraciones, tipos, etc.
- Construye la tabla de símbolos (la terminar de construir).
- **Árbol de atributos o árbol decorado**
 - Decora el árbol sintáctico con información de los componentes.



³ Semántica: Significado de algo.

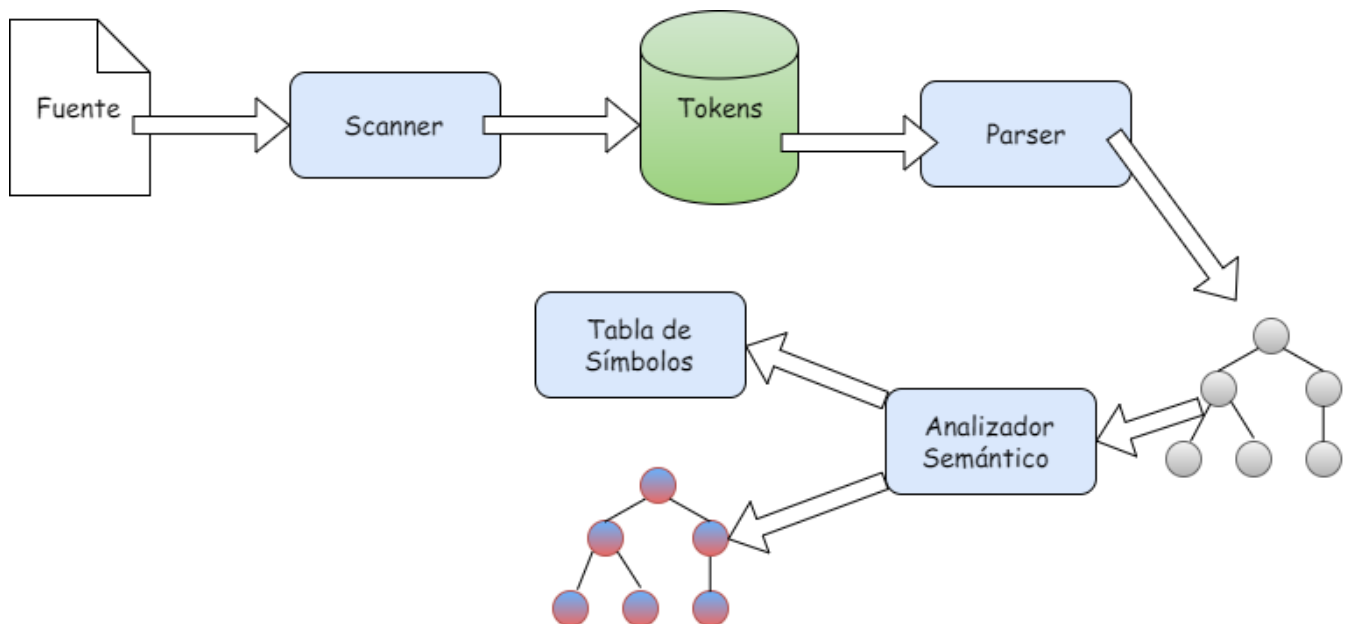
- **Entrada y salida del Analizador Semántico.**

- Le entra un Árbol sintáctico y de una forma tiene de salida una Árbol decorado y una tabla de símbolos.

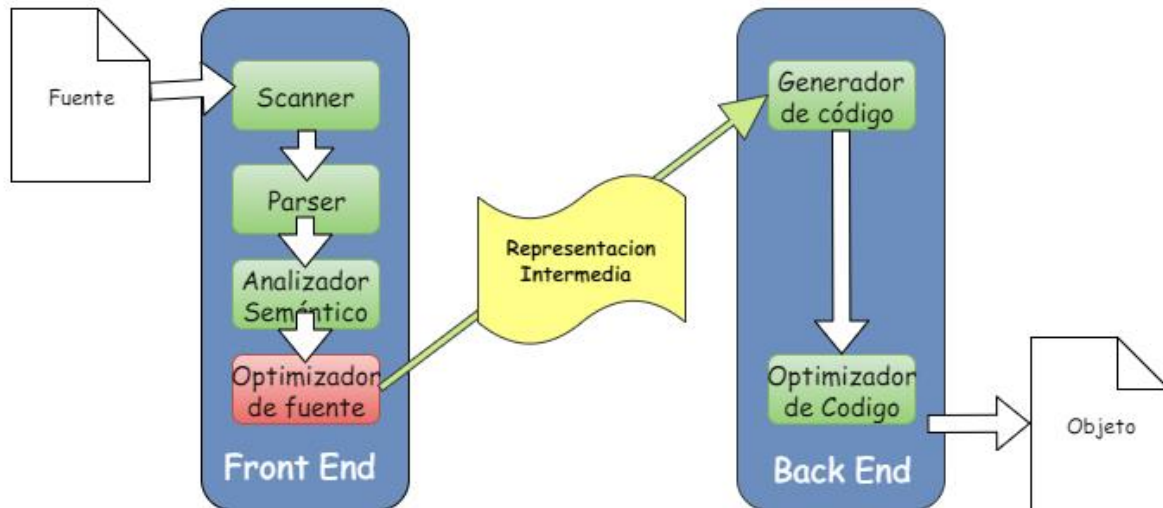


Repaso de lo que hemos formado:

- El fuente que pasa por el [Scanner](#), este da un montón de **tokens** los cuales van al Parser, donde son procesados, el [Parser](#) me retorna una **Árbol sintáctico** el cual va a ser recibido por el "[Semantec](#)" y este me va a dar de salida una **tabla de símbolos** y un **árbol decorado**

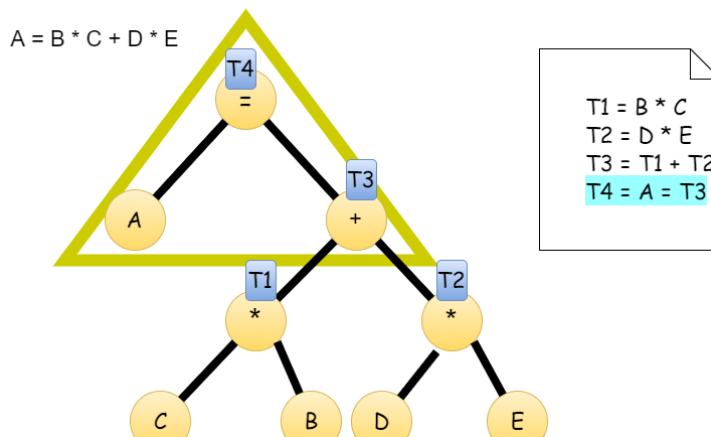
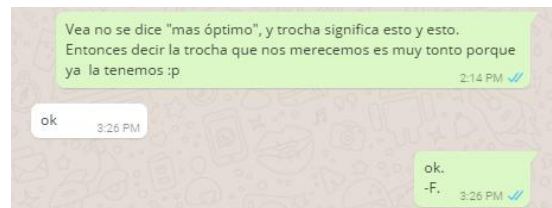


A continuación entra el Optimizador de Fuente.



Optimizador de Fuente.

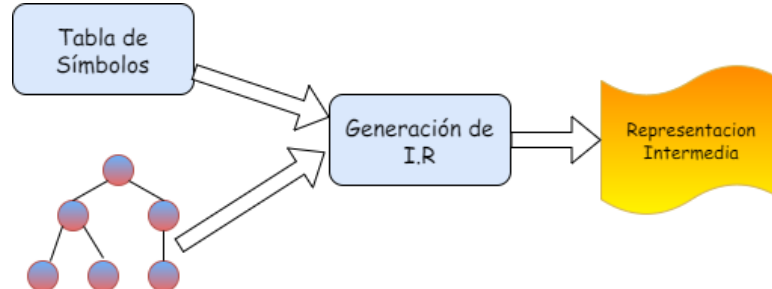
- Para empezar la palabra “Optimizar⁴” está mal usada.
- Cálculos de constante.
- Eliminación de código innecesario.
- Así se tuvo que haber visto el correo del profe con la reportera (Versión mensajería).
- Se utiliza análisis de grafos para esta tarea.
- El árbol decorado se convierte en una estructura lineal y se analiza.
- Normalmente el fuente se convierte en código de 3 direcciones⁵.
- Da de salida el código intermedio (Con ayuda de **variables temporales**).



⁴ Óp
⁵ 3 d

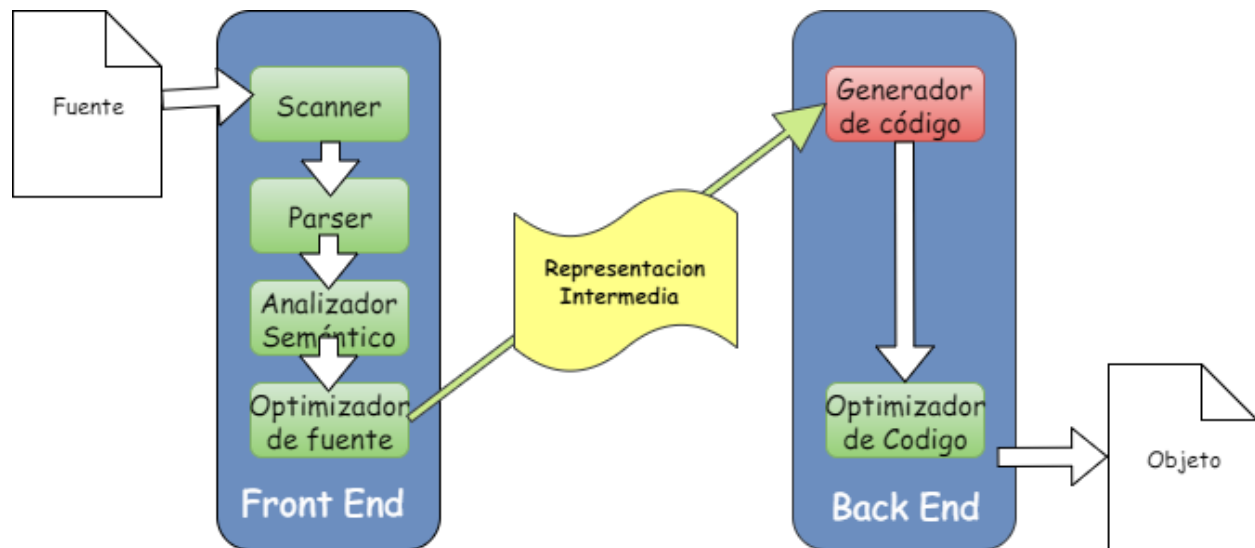
- **Entrada y salida de Generador de código intermedio(Optimizador de fuente).**

- Recibe un árbol decorado y la tabla de símbolos y nos da de salida la **representación intermedia**.



La parte del **Front-end** esta muy estudiada, existen muchas herramientas automáticas que facilitan la implementación de sus componentes. **Back – end**, es muy específico para cada arquitectura.

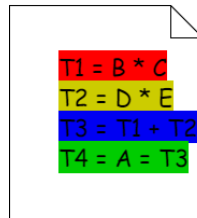
Seguimos con nuestro diagrama principal, ahora nos vamos a localizar dentro de **Back-end** con el **Generador de Código**.



Generador de Código.

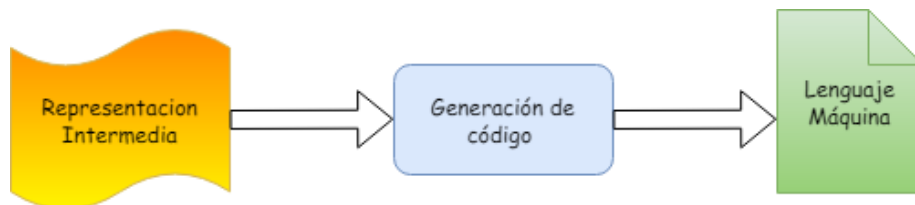
- Convertir el código intermedio en código real.
- ¿Ensamblador? ¿Lenguaje máquina?
- Ya en este punto nos interesa como se representa los datos.
 - Ejemplo :
 - Una variable **A** que tipo de dato es.
 - Cuantos bytes necesita.
 - Restricciones de colocación.
- ¿Genera ensamblador o binario ?, esta pregunta nos persiguió durante lo que quedaba de la clase ☹️.
 - Pues genera lo que queremos, si queremos solo escupimos ensamblador, o de un solo binario .

$A = B * C + D * E$

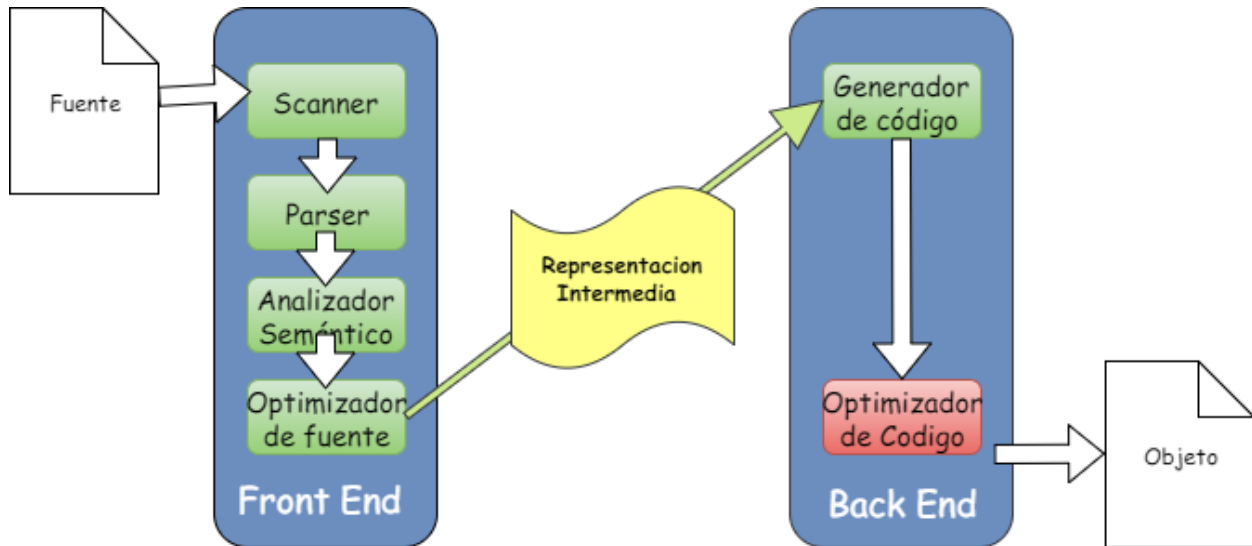


```
mov ax, C
mul B
mov T1, ax
-----
mov ax, E
mul D
mov T2, ax
-----
mov ax, T2
add ax, T1
mov T3, ax
-----
mov ax, T3
mov A, ax
mov T4, ax
```

- **Entrada y salida de Generador de código.**
 - Recibe la representación intermedia y da de salida lenguaje máquina.

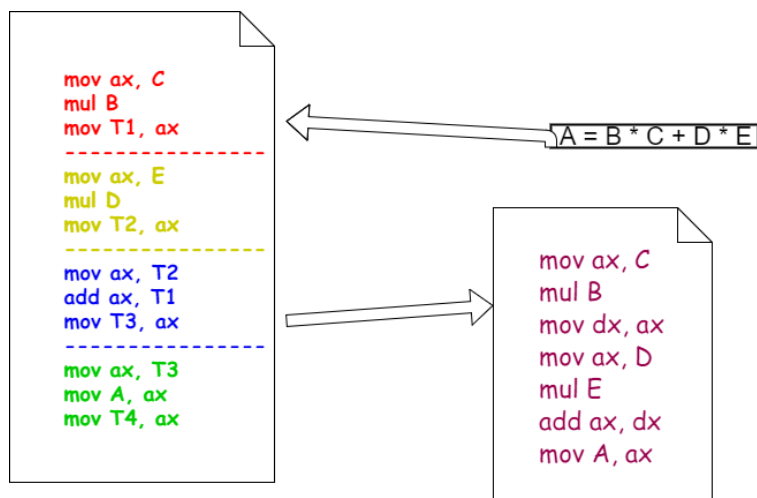


Ahora el siguiente componente.




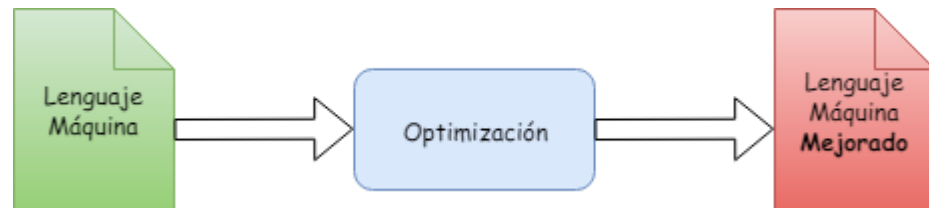
Optimizador de código.

- Mejorar código generado en el paso previo.
- Trata de dar instrucciones equivalentes pero más eficientes.
- Modo de direccionamientos más apropiados.
- Le saca el jugo a la arquitectura.
- Quita código redundante.
- Se puede optimizar por espacio o por tiempo.(más rápido o más corto)
- Ejemplo:



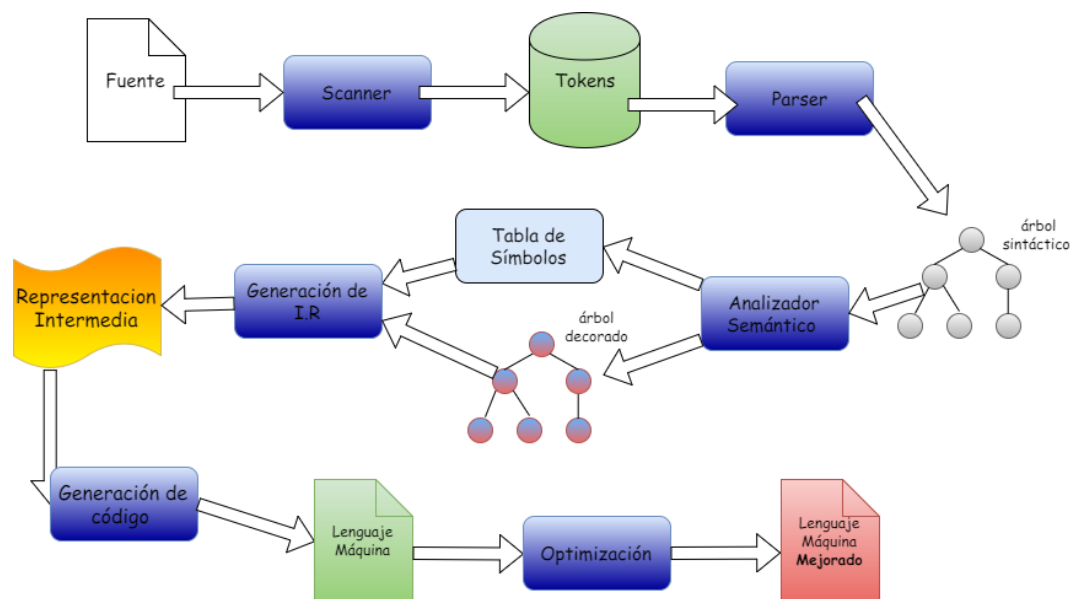
- **Entrada y salida del Optimizador de Código.**

- Recibe el código generado y lo transforma en un código equivalente más óptimo .



Y esto es lo que se tiene después de haber mencionado estos componentes.

- El fuente que pasa por el [Scanner](#), este da un montón de **tokens** los cuales van al Parser, donde son procesados, el [Parser](#) me retorna una **Árbol sintáctico** el cual va a ser recibido por el “[Semantec](#)” y este me va a dar de salida una **tabla de símbolos** y un **árbol decorado** luego estos dos elementos son recibidos por el [Optimizador de fuente](#), el cual nos va a dar la **representación intermedia**, después esta pasara por [el Generador de código](#) y este nos dará el **lenguaje máquina**, donde por último se pondrá en funcionamiento el [Optimizador de Código](#) y nos devolverá el **lenguaje maquina mejorado**.



luego estos dos elementos son recibidos por el [Optimizador de fuente](#), el cual nos va a dar la **representación intermedia**, después esta pasara por [el Generador de código](#) y este nos dará el **lenguaje máquina**, donde por último se pondrá en funcionamiento el [Optimizador de Código](#) y nos devolverá el **lenguaje maquina mejorado**.

Por lo tanto , después del fuente llegamos a un lenguaje maquina **superchivamegaoptimizado**. Pero...

Ya que es mentira que un compilador funcione como se muestra en la última imagen, lo único cierto es que entra un fuente y sale un archivo objeto.

TO BE CONTUINED.



Estructuras de datos.

se hablara de las principales estructuras de datos que se encuentra en el un compilador.

Token.

- Símbolo que representa algo.
- Retornado por el [Scanner](#).
- Estructura de 2 campos.
 - Código de Token:
 - VARIABLES.
 - INTEGER.
 - COMMA.
 - Hilera específica (EL Lexema⁶)
- Un token es una familia de lexema.
- Hay Familia de solo un miembro, como COMMA.

Árbol sintáctico.

- Creado por el [Parser](#).
- En memoria dinámica.
- Decorado por el [Analizador Semántico](#).
- A veces no existe explícitamente.
- Podrían ser punteros a otras estructuras.
- ¿Pesadilla u Orgasmo? para el Profe definitivamente es un orgasmo.



Tabla de símbolos.

- Cualquier símbolo que aparezca en nuestro programa va a acabar en ella.
 - Nombres de variables.
 - Nombres de funciones.
 - Nombre de tipos.
 - Cualquier identificador.
- Es toda promiscua, ya que tiene que ver con el Scanner, Parser, Semantex (Analizador semántico), etc. Todos se meten con la tabla de símbolos pero ella es feliz.
- Si se quiere mejorar el compilador , mejorar la tabla de símbolos va a ser muy buena idea.
- Estructura de datos más usada.
- Normalmente con acceso HASH.

⁶ Lexema: ejemplo particular de un token.

numeroLineas = 2; Token = INTEGER, LEXEMA = numeroLineas.

Apuntes 8 de marzo de 2017.

Izcar Muñoz Torrez.

Tabla de literales.

- Parecida a la tabla de símbolos.
- Guarda todos las constantes(números, hileras, etc.)
- Nos interesa distinguir estas constantes ya que no van a cambiar en el tiempo de ejecución, aunque tenga que residir en memoria.
- Ayuda a reducir el tamaño de código objeto.

Código intermedio.

- Hay que representarlo de algún modo. Hay muchas posibilidades.
 - Arreglos de punteros.
 - Archivos de texto.
 - Listas enlazadas.
 - En memoria.

Archivos temporales.

- Archivo de vida corta.
 - Guardan cosas.
 - Se leen.
 - ¡Luego se desechan!
- Resuelven problemas complejos fácilmente.
- Son nuestros amigos 😊.

Apuntes 8 de marzo de 2017.

Izcar Muñoz Torrez.

Preguntas sobre el proyecto y recordatorios.

- El proyecto debe de correr en LINUX. Nada de cosas de esas de vagabundos.
- El proyecto es para ahorita(¡ya hay miedo!).
- Para los incluye se recomienda utilizar la pila o archivos temporales.
 - ¿Qué sería lo más óptimo?
 - Cuidado con los LOOP.
- Preguntar en el Foro.



HUYAN



memegenerator.net