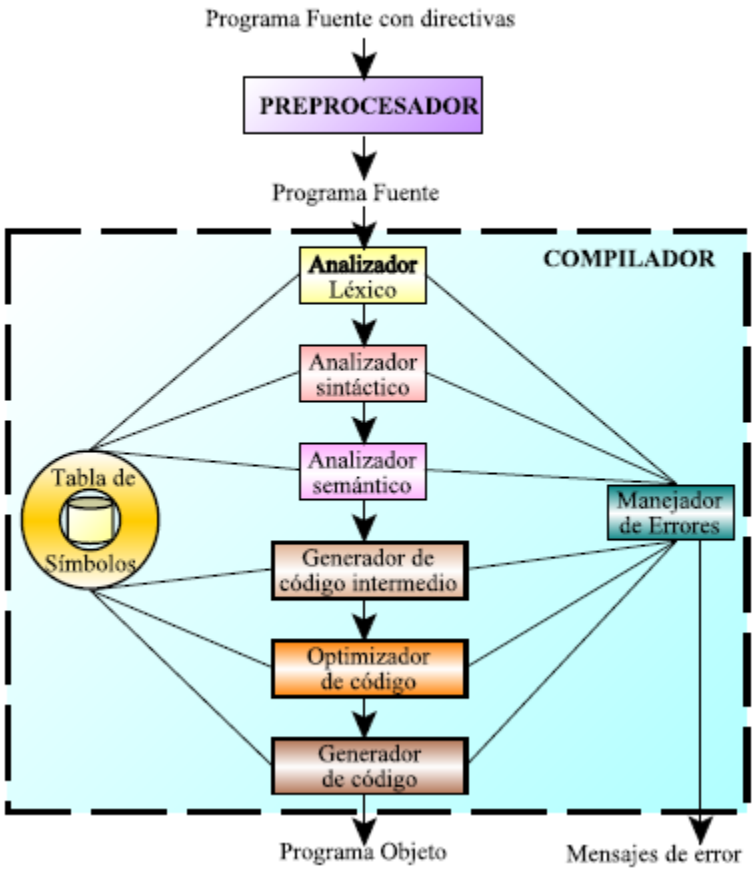
Fases del compilador

Fases de compilación:



Fases de análisis: 1-3, Fases de síntesis: 4-6

La fase 0. Preprocesador

Permiten modificar el programa fuente antes de la verdadera compilación. Hacen uso de macroinstrucciones y directivas de compilación. Por ejemplo, en lenguaje C, el preprocesador sustituye la directiva #include Uno.cpp por el código completo que contiene el fichero “Uno.cpp”.

Investiga que significa las palabras: análisis, síntesis, léxico, sintáctico y semántico.

Etapa de análisis

En esta etapa se controla que el texto fuente sea correcto en todos los sentidos, y se generan las estructuras necesarias para la generación de código.

1. Fase de análisis lexicográfico

En esta fase, la cadena de caracteres que constituye el programa fuente se lee de izquierda a derecha y se agrupa en componentes léxicos, que son secuencias de caracteres que tienen un significado atómico; además el analizador léxico *trabaja con la tabla de símbolos* introduciendo en ésta los nombres de las variables.

Ejemplo:

comision= fijo + valor * 8 ;

se agruparían en los componentes léxicos siguientes:

1.- El identificador comision.

2.- El símbolo de asignación '='.

3.- El identificador fijo.

4.- El signo de suma '+'.

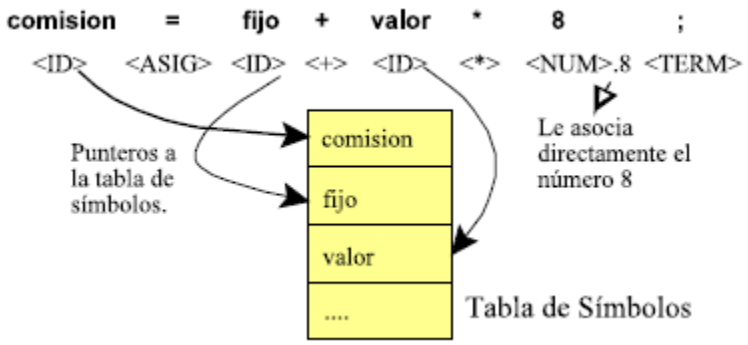
5.- El identificador valor.

6.- El signo de multiplicación '*'.

7.- El número 8.

8.- El símbolo de fin de sentencia ';'.

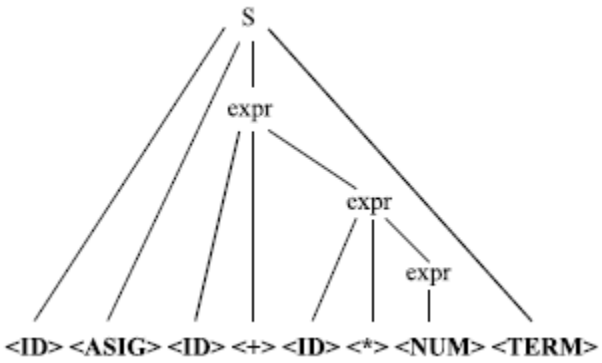
A continuación, se muestra que ocurriría con la tabla de símbolos.



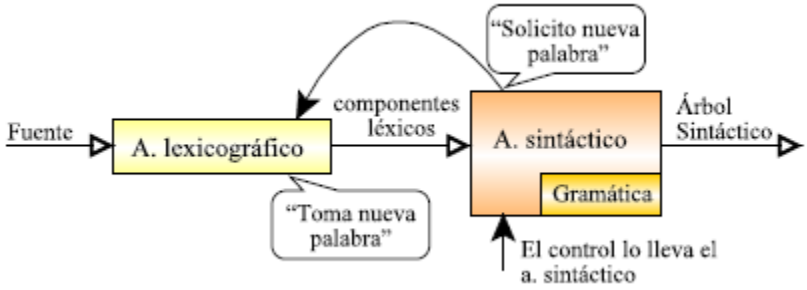
2. Fase de análisis sintáctico

Trabaja con una gramática de contexto libre y *genera el árbol sintáctico* que reconoce su sentencia de entrada. En nuestro caso las categorías gramaticales del análisis léxico son los terminales de la gramática.

Para el ejemplo:



Compilación dirigida por sintaxis:



3. Fase de análisis semántico

Esta fase revisa el árbol sintáctico junto con los atributos y la tabla de símbolos para tratar de encontrar errores semánticos. Para todo esto se analizan los operadores y operandos de expresiones y proposiciones. Finalmente reúne la información necesaria sobre los tipos de datos para la fase posterior de generación de código.

El componente más importante del análisis semántico es la *verificación de tipos*. Aquí, el compilador verifica si los operandos de cada operador son compatibles según la especificación del lenguaje fuente.

Si suponemos que nuestro lenguaje solo trabaja con números reales, la salida de esta fase sería su mismo árbol, excepto porque el atributo de <NUM>, que era el entero 8 a la entrada, ahora pasaría a ser el real 8.0.

Además, se ha debido controlar que las variables implicadas en la sentencia, a saber, comision, fijo y valor son compatibles con el tipo numérico de la constante 8.0.

Etapa de síntesis

En la etapa anterior se ha controlado que el programa de entrada es correcto. Por tanto, el compilador ya se encuentra en disposición de generar el código máquina equivalente semánticamente al programa fuente. Para ello se parte de las estructuras generadas en dicha etapa anterior: árbol sintáctico y tabla de símbolos.

4. Fase de generación de código intermedio

Después de la etapa de análisis, se suele generar una representación intermedia explícita del programa fuente. Dicha representación intermedia se puede considerar como un programa para una máquina abstracta.

Cualquier representación intermedia debe tener dos propiedades importantes; debe ser fácil de generar y fácil de traducir al código máquina destino. Así, una representación intermedia puede tener diversas formas.

Ejemplo:

Se trabajará con una forma intermedia llamada "*código de tres direcciones*", que es muy parecida a un lenguaje ensamblador para un microprocesador que carece de registros y sólo es capaz de trabajar con direcciones de memoria y literales. En el código de tres direcciones cada instrucción tiene como máximo tres operandos. Siguiendo el ejemplo propuesto, se generaría el siguiente código de tres direcciones:

t1 = 8.0

t2 = valor * t1

t3 = fijo + t2

comision = t3

De este ejemplo se pueden destacar varias propiedades del código intermedio escogido:

• Cada instrucción de tres direcciones tiene a lo sumo un operador, además de la asignación.

• El compilador debe generar un nombre temporal para guardar los valores intermedios calculados por cada instrucción: t1, t2 y t3.

• Algunas instrucciones tienen menos de tres operandos, como la primera y la última, instrucciones del ejemplo.

5. Fase de optimización de código

Esta fase trata de mejorar el código intermedio, de modo que en la siguiente fase resulte un código de máquina más rápido de ejecutar. Algunas optimizaciones son triviales. En nuestro ejemplo hay una forma mejor de realizar el cálculo de la comisión, y pasa por realizar sustituciones triviales en la segunda y cuarta instrucciones, obteniéndose:

t2 = valor * 8.0

comision= fijo + t2

El compilador puede deducir que todas las apariciones de la variable t1 pueden sustituirse por la constante 8.0, ya que a t1 se le asigna un valor que ya no cambia, de modo que la primera instrucción se puede eliminar. Algo parecido sucede con la variable t3, que se utiliza sólo una vez, para transmitir su valor a comision en una asignación directa, luego resulta seguro sustituir comision por t3, a raíz de lo cual se elimina otra de las líneas del código intermedio.

6. Fase de generación de código máquina

La fase final de un compilador es la generación de código objeto, que por lo general consiste en código máquina reubicable o código ensamblador. Cada una de las variables usadas por el programa se traduce a una dirección de memoria (esto también se ha podido hacer en la fase de generación de código intermedio). Después, cada una de las instrucciones intermedias se traduce a una secuencia de instrucciones de máquina que ejecuta la misma tarea. Un aspecto decisivo es la asignación de variables a registros.

Siguiendo el mismo ejemplo, y utilizando los registros R1 y R2 de un microprocesador hipotético, la traducción del código optimizado podría ser:

MOVE [1Ah], R1

MULT #8.0, R1

MOVE [15h], R2

ADD R1, R2

MOVE R2, [10h]

El primer y segundo operandos de cada instrucción especifican una fuente y un destino, respectivamente. Este código traslada el contendido de la dirección [1Ah] al registro R1, después lo multiplica por la constante real 8.0. La tercera instrucción pasa el contenido de la dirección [15h] al registro R2. La cuarta instrucción le suma el valor previamente calculado en el registro R1. Por último, el valor del registro R2 se pasa a la dirección [10h]. Como el lector puede suponer, la variable comision se almacena en la dirección [10h], fijo en [15h] y valor en [1Ah].