


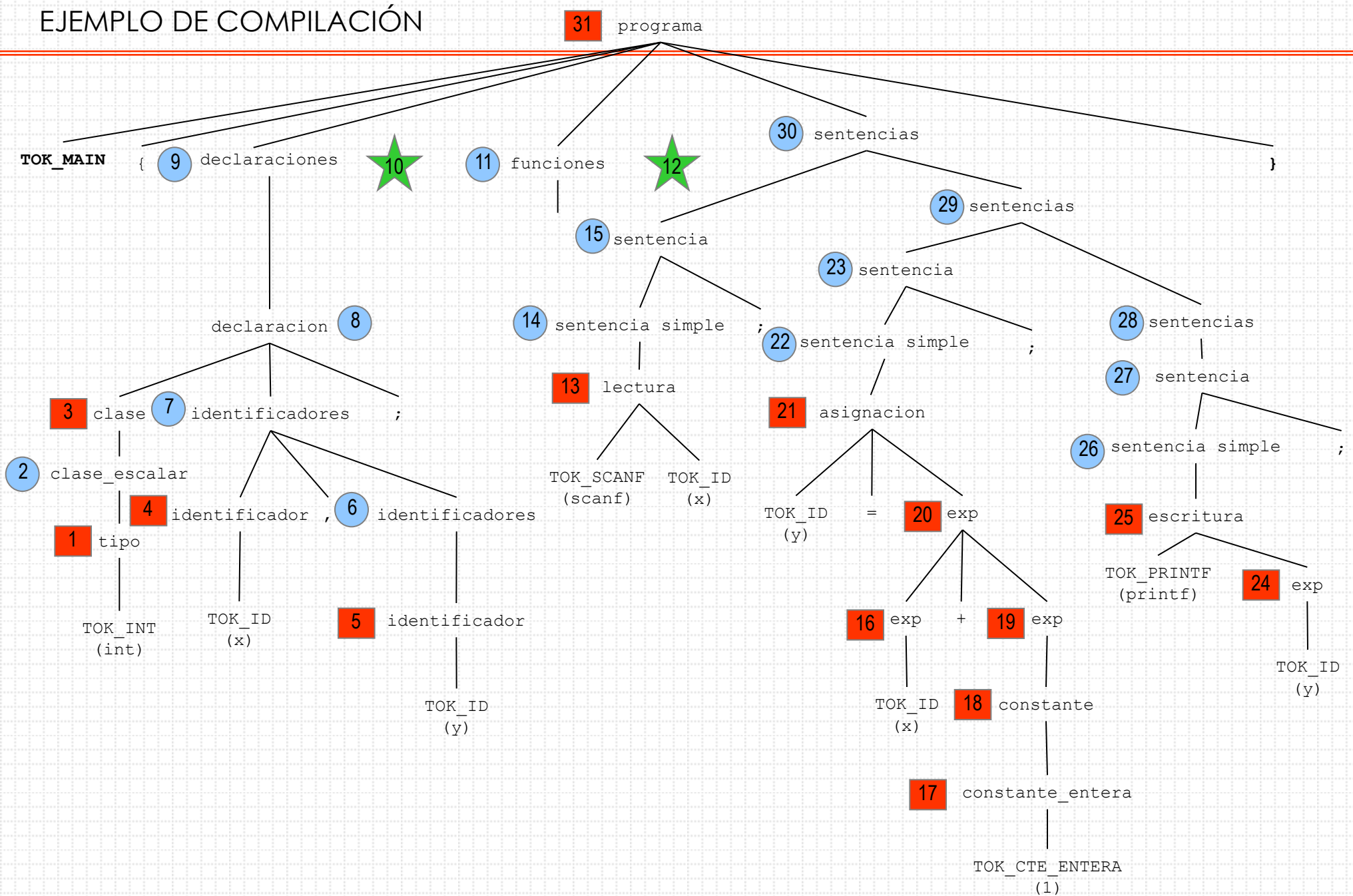


- ✖ En este documento se presenta un ejemplo de compilación de un programa escrito en ALFA. Durante el proceso de compilación se traduce el código ALFA a código NASM. El objetivo es identificar cómo incorporar al proceso de análisis sintáctico la generación de código ensamblador, sabiendo que cada vez que se reduce una regla de la gramática se ejecuta su acción asociada (si tiene).
- ✖ El programa ALFA que se utilizará en el ejemplo se muestra en la figura.

```
// Incrementa en 1 el  
// dato de entrada  
// y muestra el resultado  
  
main  
{  
    int x, y;  
    scanf x;  
    y = x + 1;  
    printf y;  
}
```

- ✖ En la siguiente figura se muestra el árbol de análisis del programa de ejemplo en el que se indica el orden en el que se reducen las reglas con la siguiente notación:
 -  Si una regla está marcada con un cuadrado rojo significa que en su acción semántica se ejecutan tareas.
 -  Si una regla está marcada con un círculo azul significa que su acción semántica está vacía.
 -  La estrellas verdes representan puntos en los que se necesita hacer una acción semántica. Al tratarse de puntos no situados al final de una regla hace necesario manipular la gramática .

EJEMPLO DE COMPILACIÓN



1

`tipo : TOK_INT`

- Establecer el valor de las variables globales `tipo_actual = INT`

3

`clase : clase_escalar`

- Establecer el valor de la variable global `clase_actual = ESCALAR`

4

`identificador : TOK_IDENTIFICADOR`

- Insertar el identificador (x) en la tabla de símbolos

5

`identificador : TOK_IDENTIFICADOR`

- Insertar el identificador (y) en la tabla de símbolos

10

Nueva producción lambda

- Escribir la tabla de símbolos en el fichero ensamblador, en el segmento de datos “bss”.
- Escribir la primera parte (constante) del segmento de código

```
segment .bss
    _x resd 1
    _y resd 1
segment .text
    global main
    extern scan_int , scan_float, etc
```

12

Nueva producción lambda

- Escribir la etiqueta de inicio del main, “main:” (no se tiene en cuenta el código correspondiente a la preservación del puntero de pila para su restauración al final del programa).

```
; -----
; PROCEDIMIENTO PRINCIPAL
; -----
main:
```

13 `lectura : TOK_SCANF TOK_IDENTIFICADOR`

- Buscar en la tabla de símbolos el identificador (x).
- Insertar en la pila la dirección donde almacenar el dato leído (lexema del identificador precedido de “_”).
- Invocar a la rutina `scan_int`.
- Dejar la pila como estaba antes de la invocación

```
push dword _x
call scan_int
add esp, 4
```

16 `exp : TOK_IDENTIFICADOR`

- Buscar en la tabla de símbolos el identificador (x)
- Propagar atributos (tipo, `es_direccion`)
- Apilar la dirección de inicio del identificador (que es el lexema del identificador precedido de “_”)

```
push dword _x
```

17 `constante_entera : TOK_CONSTANTE_ENTERA`

- Apilar la constante entera
- Propagar atributos (tipo, `es_direccion` y valor)

```
push dword 1
```

18 constante : constante_entera

- Propagar atributos (tipo y es_direccion)

19 exp : constante

- Propagar atributos (tipo y es_direccion)

20 exp : exp '+' exp

- Depositar en edx el valor del segundo operando (es la constante 1)
- Depositar en eax el valor del primer operando (es la variable x)
- Sumar eax y edx dejando el resultado en eax
- Apilar el resultado de la suma

```
pop dword edx
pop dword eax
mov dword eax, [eax]
add eax,edx
push dword eax
```

21

`asignacion : TOK_IDENTIFICADOR '=' exp`

- Realizar la asignación (la parte derecha de la asignación está en la pila y la parte izquierda es la variable `y`).

```
pop dword eax  
mov dword [_y], eax
```

24

`exp : TOK_IDENTIFICADOR`

- Buscar en la tabla de símbolos el identificador (`y`)
- Propagar atributos (tipo y `es_direccion`)
- Apilar la dirección de inicio del identificador (que es el lexema del identificador) precedido de “_”.

```
push dword _y
```


25 `escritura : TOK_PRINTF exp`

- Depositar en `eax` el valor para la salida
- Apilar `eax`
- Invocar a la función `print_int`
- Dejar la pila como estaba antes de la invocación
- Invocar a la función `print_endofline`

```
pop dword eax
mov dword eax, [eax]
push dword eax
call print_int
add esp, 4
call print_endofline
```

31 `programa : TOK_MAIN '{' declaraciones funciones sentencias '}'`

Escribir el final del fichero ensamblador (no se tiene en cuenta el código correspondiente a la restauración del puntero de pila ni a la gestión de errores en tiempo de ejecución).

```
ret
```

EJEMPLO DE COMPILACIÓN

10
segment .bss
 _x resd 1
 _y resd 1

12
segment .text
 global main
 extern scan_int ;etc
 ; -----
 ; PROCEDIMIENTO PRINCIPAL
 ; -----
main:

13
 push dword _x
 call scan_int
 add esp, 4

16
 push dword _x

17
 push dword 1

20
 pop dword edx
 pop dword eax
 mov dword eax, [eax]
 add eax,edx
 push dword eax

21
 pop dword eax
 mov dword [_y], eax

24
 push dword _y

25
 pop dword eax
 mov dword eax, [eax]
 push dword eax
 call print_int
 add esp, 4
 call print_endofline

31
 ret