GENERACION DE CODIGO PARA UN MINI-LENGUAJE

Por Pedro Roa Medina y Luis Fariñas del Cerro

1. INTRODUCCION

Se ha completado la realizacion y puesta a punto de un compilador, del cual ya estaba implementado su analizador morfologico, para un lenguaje sencillo de alto nivel, en la forma de generador de código.

Las caracteristicas globales mas importantes son:

1.1. Lenguaje aceptado por el compilador

El lenguaje que acepta el compilador (a nivel de analisis sintactico) viene dado por la gramatica siguiente, escrita en la forma normal de Backus:

```
<programa> ⟶ <instruccion> |
              <programa> ; <instruccion>
<instruccion> ⟶ <declarativo> |
                 <no declarativo> |
                 <control>
<declarativo> ⟶ 'REAL <letra> |
                  INTEGER <letra> |
                  LOGICAL <letra>
```

<no declarativo> → <asignacion> |
<condicional> |

<asignacion> → <letra> = <expresion>

<expresion> → <expresion> .OR. <exprel> |
<exprel>

<exprel> → <exprel> .AND. <expre2> |
<expre2>

<expre2> → <expre2> .RO. <expre3> |
<expre3>

<expre3> → <expre3> + <termino> |
<expre3> - <termino> |
<termino>

<termino> → <termino> *<factor> |
<termino> / <factor>
<factor>

<factor> → (<expresion>) |
<letra> |
.NOT. <factor> |
<numero>

<condicional> → <ifclay> THEN <asignacion> ELSE <no declarativo>

<ifclay> → IF <expresion>

<control> → END

<letra> → |A| ... |Z

<numero> → <inte> . <inte>

1.2. Analizador morfologico de partida.

Contabamos al comenzar el trabajo con un A.M. con las siguientes caracteristicas:

1.2.1. Gramática de las U.S. aceptadas, dada en B.N.F.

⟨us⟩ → ⟨cp⟩ |
⟨pi⟩ |
⟨orl⟩ |
⟨cc⟩ |
⟨us⟩ |
⟨int⟩

⟨cp⟩ → = | + |) | - | x | / | (

⟨pi⟩ → A | ... | Z |
R ⟨rcont⟩ |
I ⟨icont⟩ |
T ⟨tcont⟩ |
E ⟨econt⟩ |
L ⟨lcont⟩ |

⟨rcont⟩ → EAL | ϕ

⟨icont⟩ → NTEGER | F | ϕ

⟨tcont⟩ → HEN | ϕ

⟨econt⟩ → LSE | ND | ϕ

⟨lcont⟩ → OGICAL | ϕ

⟨orl⟩ → . ⟨rorl⟩

⟨rorl⟩ → G ⟨rg⟩ |
EQ. |
L ⟨rl⟩ |
OR. |
N ⟨rn⟩ |
AND. | ϕ

⟨rg⟩ → T. |
F.

⟨rl⟩ ⟶ E.

T.

⟨rn⟩ ⟶ OT.

E.

⟨cc⟩ ⟶ ,.

⟨int⟩ ⟶ DIGITO ⟨restin⟩

⟨restin⟩ ⟶ DIGITO⟨restin⟩

⟨digito⟩ ⟶ 0/ .... /9

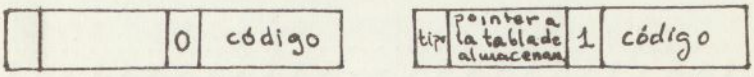
I.2.2. Forma de las U.S. que nos da el A.M.

Al recibir la cadena que forma el programa fuente el A.M. nos ira dando las distintas unidades sintacticas con arreglo a unos codigos como de indica en la tabla 1.

=	1	:AND.	15
+	2	/	16
;	4	.NOT.	17
)	5	(	18
.GE.	6	ELSE	19
.GT.	7	IF	20
.EQ.	8	REAL	21
-	9	THEN	22
.LE.	10	INTEGER	23
.LT.	11	END	24
*	12	LOGICAL	25
.OR.	13	.	26
.NE.	14	⟨entero⟩	27
		⟨ident.⟩	28

TABLA 1.

Cada unidad sintactica tendrá especificado ademas si es nodo o variable con un 0 o un 1 en el campo de tag, respectivamente, y en el caso de variable vendrá tambien indicado su tipo en bits 5,1,2 que contendrán un 1 si es integer, 2 si real y 3 para logical; y un pointer a la tabla de almacenamiento.



U.S. correspondiente a un nodo

U.S. correspondiente a una variable.

1.2.3. La implementacion concreta del A.M. es en forma de una rutina de analisis del tipo top-down que deja una U.S. en una determinada posición (CARCUR) cada vez que es llamada la rutina.

1.3. Descripción global del compilador.

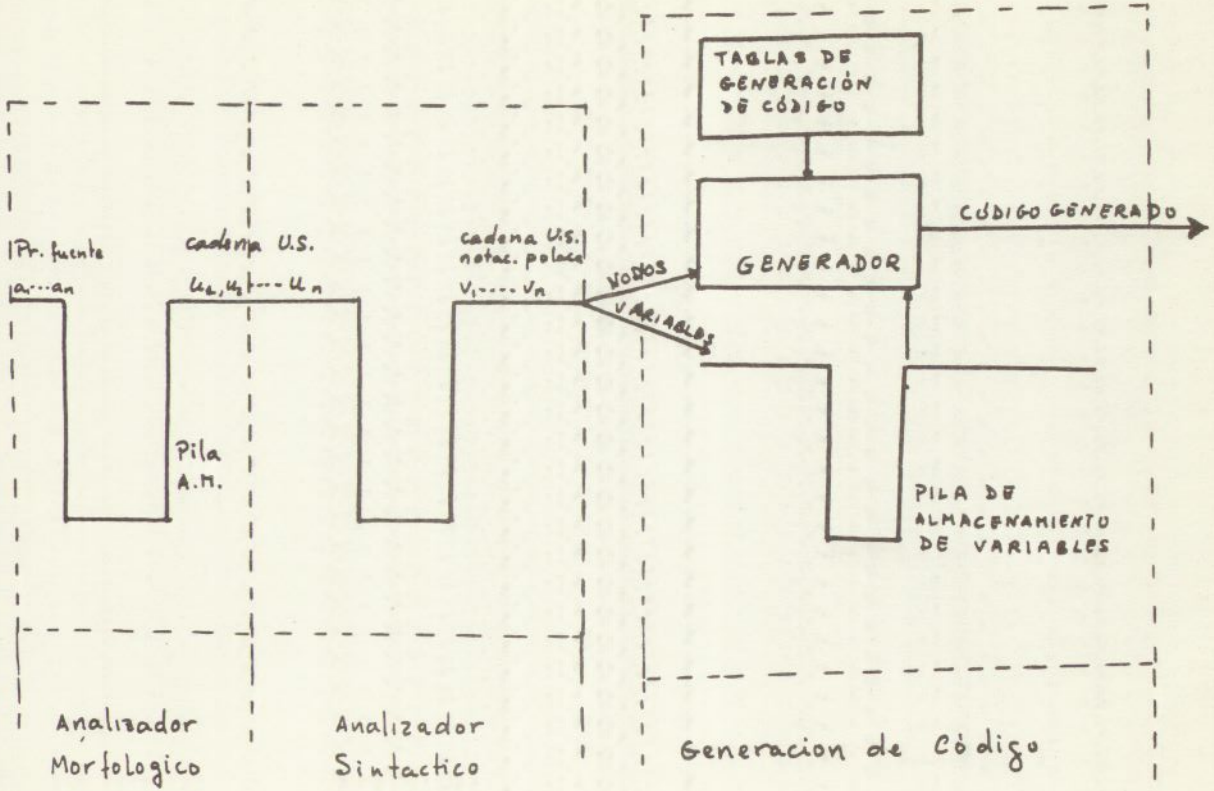
Consta de tres partes: a) Analizador morfologico
b) Rutina AS
c) Generador de código

El generador de codigo rige el funcionamiento general del proceso, este pide las U.S. a la rutina AS conforme las necesita, la cual a su vez dará control al A.M. siempre que sea necesario; la generacion se irá realizando, por tanto, en paralelo a la lectura del programa fuente.

El generador usa una pila de almacenamientode variables y genera código cada vez que recibe un operador, sacando de la pila los operandos a los que afecta este operador.

El esquema general de trabajo puede visualizarse en la siguiente figura:

COMPILADOR DIDACTICO



2.RUTINA AS.

Esta rutina nos dejara en ouput una U.S.en notación polaco inversa. Al llamar sucesivamente a esta rutina podemos distinguir cuatro es tructuras distintas en memoria .

2.1.Cola de entrada de U.S.

Los elementos de la cola de entrada consisten en una palabra de memoria (CARCUR),que nos viene dada al llamar,siempre que necesitemos un elemento,al analizador morfologico,y tendran la forma indicada en la descripción de este.

2.2.Pila de almacenamiento de U:S:

Sus elementos serán iguales a los de la cola anterior;se les dará una almohadilla en la inicialización del proceso.

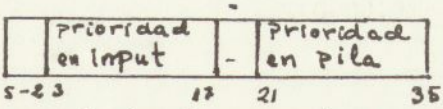
2.3.Cola de salida de U.S.

La cola de salida es una cola de U.S. conseguida al pasar la de entrada a notación polaco inversa.

2.4.Tabla de prioridades .

Tiene tantos elementos como U.S. posibles distintas tenga el lengua ge (en nuestro caso 28) más un elemento para las prioridades de la almohadilla de la pila.

Cada elemento de la tabla necesita una posición de memoria y contie ne las prioridades correspondientes a un solo tipo de U.S.,la prioridad de dicha U.S. será distinta si esta se encuentra en la cola de entrada o en la pila ,así en cada palabra almacenamos en la parte de decremento la prioridad en el imput y en la parte de dirección la prioridad en la pila.



Ejemplo de elemento de tabla de prioridades.

Descripción dinamica de la rutina AS.

Funciona segun el siguiente algoritmo.

1.- Calcular la prioridad del elemento primero de la cola de entrada.

2.- Calcular la prioridad del elemento tope de la pila.

3.- Comparar ambas prioridades.

4.- Si la prioridad del elemento del stack es igual o mayor que el de la cola, sacamos de la pila un elemento y lo daremos a la cola de salida, y devolvemos el control al programa principal; si es menor apilamos la cabeza de la cola de entrada y volvemos al punto 1.

U S	Pr. input	Pr. stack	U S	Pr. input	Pr. stack
Almoh.	0	0	.AND.	7	7
=	15	2	/	13	13
+	12	12	.NOT.	10	10
,.	1	20	(	20	1
)	2	0	ELSE	1	20
.GE.	11	11	IF	1	20
.GT.	11	11	REAL	1	20
.EQ.	11	11	THEN	1	20
-	12	12	INTEGER	1	20
.LE.	11	11	END	0	0
.LT.	11	11	LOGICAL	1	20
*	13	13	.	14	14
.OR.	6	6	<ente>	16	16
.NE.	11	11	<ident.>	16	16

TABLA DE PRIORIDADES. (valores estables)

La construcción de la tabla de prioridades se ha realizado de forma que:

a) Las U.S. del tipo IF, THEN, ELSE, ; , REAL, INTEGER y LOGICAL conservan en la cola de salida el mismo orden que tenían entre sí en la cola de entrada.

b) En el caso de parentesis se realiza su eliminación por parejas.

c) La U.S. END está tratada de forma que vacíe la pila (sacando hasta la almohadilla) la cual indica al generador de código que se ha terminado el proceso.

d) El resto de las U.S. se trata de forma standard.

3.GENERACION DE CODIGO

Las U.S. en notación polaca que el generador de código recibe - son clasificadas en tratadas de forma distinta segun sean nodos o vertices, estos ultimos son simplemente almacenados en una pila; cuando se reciba un nodo pueden suceder tres casos:

a) que sea un declarativo o una U.S. de codigo cero; es decir, que empieza una orden no ejecutable.

b) que sea un operador.

c) que sea una de las unidades IF, THEN, ELSE.

3.1. Expresiones no ejecutables.

Al recibir el G. de C. una U.S. INTEGER, REAL o LOGIC L ignora esta y todo lo que encuentre hasta que le llegue un ;. Si recibe una U.S. de codigo cero indica que el U.S. ha recibido un END que habra vaciado la pila; entonces se termina la compilacion y se dará control a la primera instruccion generada, para ejecutar el programa objeto.

3.2. Operadores

Cuando recibe el G.de C. de C. un operador, su operando u operandos o bien uno de los operadores estará en la pila de operandos o bien uno de los operadores estará en el Ac. y esto reindicará en la pila metiendo un elemento especial.

Generaremos por tanto el código correspondiente a estos elementos a continuación introduciendolos en una pila de código generado.

3.2.1 El código fuente correspondiente a las operaciones a reali-

zar que nos indica un operador están almacenadas en memoria en forma de tablas con la siguiente estructura: Una primera tabla de 3 elementos a los cuales entraremos al 1º si los dos operandos son identificadores o números; al 2º si el 1º es identificador ó número y el 2º está en el Ac.; al 3º en el caso opuesto.

Cada uno de los 3 elementos almacena:

1º número de órdenes a generar

2º dirección donde está almacenada la 1ª instrucción a generar, detras de la cual estarán el resto del código correspondiente.

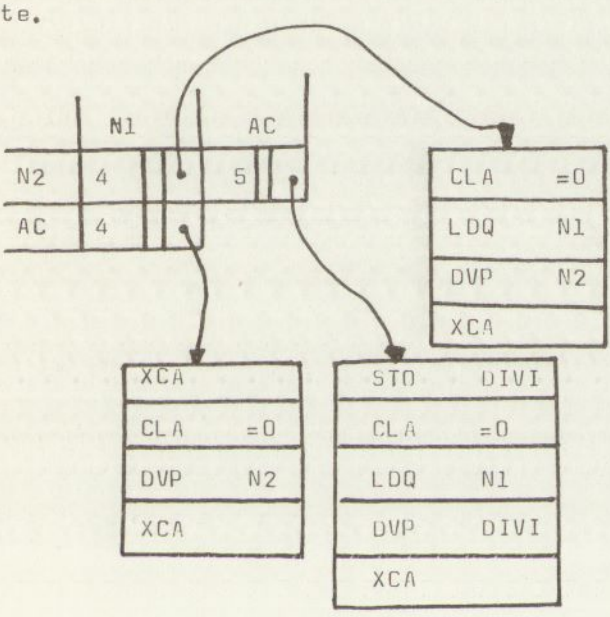


Tabla de generación de codigo de division entera.

3.2.2. Recibido el pperador de por el generador de código, se realiza en primer lugar una comprobacion de la igualdad de los tipos declarados a los operandos (si el operador es binario) a que afecta. Si alguno de ellos no lo tuviera declarado o si fue-

sen de tipos distintos nos daria un mensaje de error. Se efectua tambien comprobación de que el tipo de el operador y de los operandos sea compatible, en caso contrario se da mensaje de error; siempre que una orden tenga algun error se abandona la compilacion de esta buscando el ; que nos indican su final.

Para decidir que tabla de generacion hemos de usar se necesita conocer: a) de que operador se trata y b) tipo de los operandos afectados por este; una vez conocidos estos, se introduce en la pila de codigo generado la tabla correspondiente. En las tablas de codigo correspondientes a cada operador no se han colocado las direcciones de los operandos, solo se distingue si se trata del primero o del segundo de ellos; estas direcciones vendrán almacenadas en las U.S. que estan en la pila que fueron construidas por el A.M..

Cada vez que realizamos una generacion, lo haremos de tal forma que al ejecutar estas ordenes nos deje el resultado en el AC, y por tanto hemos de ver si el registro AC. estaba ya cargado, y en este caso generar una instrucción de almacenamiento en una pila de variables temporales.

3.3 Expresiones condicionales.-

3.3.1 Tratamiento en la rutina AS.

Cuando el A.M. nos da una expresión condicional, que puede consistir en IF encadenados, el paso por la rutina AS nos conserva

el orden de las U.S. IF, THEN, ELSE y ; entre si, si bien las - expresiones entre IF y THEN y las asignaciones entre THEN y ELSE, y las no declara entre el ELSE y el ; pasarían a notación polaca.

3.3.2 Generación de codigo.

Es necesaria para la implementación una pila (TETIQ) cuyo uso indicaremos seguidamente.

Al recibir una estructura if...then...else...; podemos distinguir dos partes: generación a realizar y acciones a tomar.

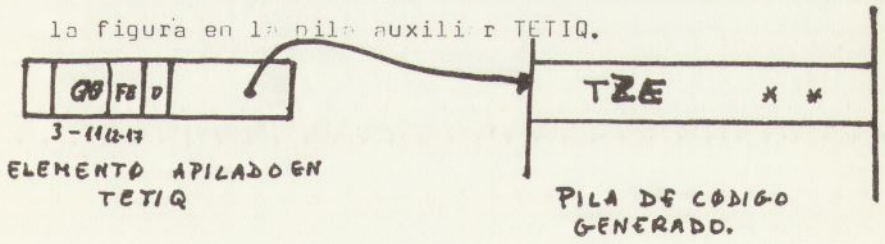
El algoritmo para el tratamiento de estas expresiones es el siguiente:

1.- Inicializamos el proceso poniendo a cero una variable FE que nos indica el nº de IF abiertos.

2.- Al recibir un IF aumentamos FE en una unidad y no generamos ningún codigo.

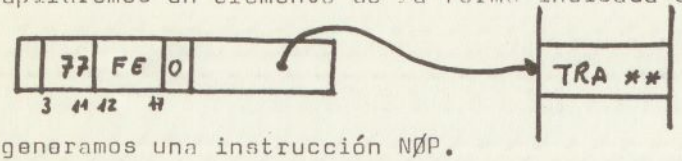
3.- Generación del codigo correspondiente a la expresión ante-- rior al THEN.

4.- Al recibir un THEN generaremos una instrucción TZE a la que no damos dirección y apilaremos un elemento de la forma de - la figura en la pila auxiliar TETIQ.



5.- Generaremos el código correspondiente a la expresión de asignación que venga entre THEN y ELSE.

6.- Al recibir un ELSE
   a) generaremos una instrucción TRA sin dirección
   b) apilaremos un elemento de la forma indicada en la figura.



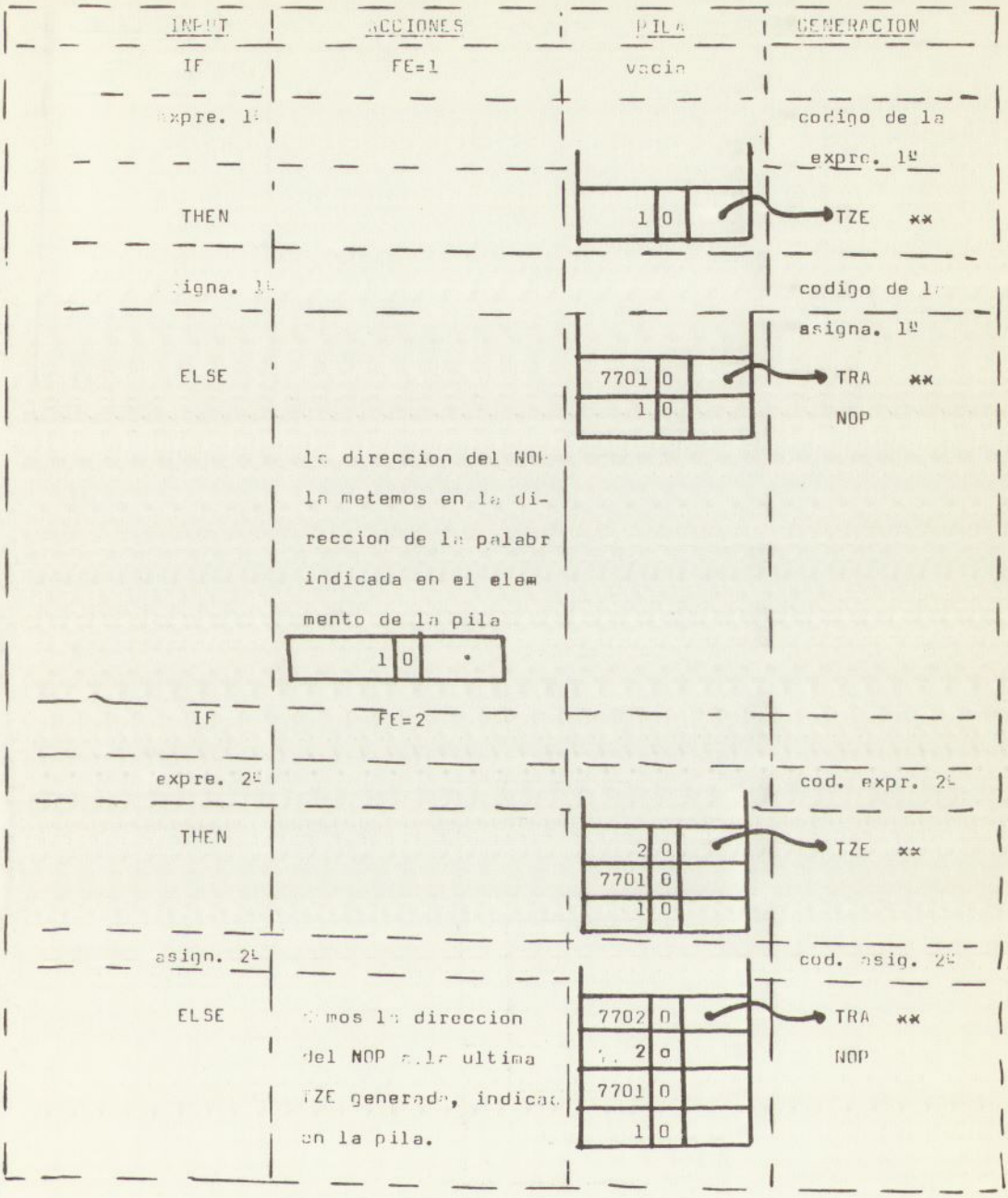
   c) generamos una instrucción NØP.
   d) buscamos en la pila un elemento tal que en su parte de decremento contenga el valor actual de FE y este nos dará la dirección de la instrucción TZE generada por el THEN anterior y a esta instrucción le daremos la dirección de la NØP ultimamente generada.

7.- Si encontramos un IF a continuación, pasar al paso 2; si no, generaremos el código correspondiente a la asignación siguiente y pasaremos al punto 8.

8.- Cuando recibimos el ; que nos termina la instrucción haremos:
   a) FE=0
   b) generar NØP
   c) mirar en todos los elementos de la pila que tengan 77 en bits 3-11 y poner la dirección de la última NØP generada como dirección de todas las instrucciones que nos indiquen estos elementos
   d) vaciar la pila.

3.3.3 Ejemplo aclaratorio.

Este ejemplo se realiza siguiendo el algoritmo descrito.


INPUT
ACCIONES
PILA
GENERACION
IF
FE=1
vacia
expre. 1ª
codigo de la
expre. 1ª
THEN
1 0
TZE **
asigna. 1ª
codigo de la
asigna. 1ª
ELSE
7701 0
1 0
TRA **
NOP
la direccion del NOP
la metemos en la di-
reccion de la palabr
indicada en el elem
mento de la pila
1 0 .
IF
FE=2
expre. 2ª
cod. expr. 2ª
THEN
2 0
7701 0
1 0
TZE **
asign. 2ª
cod. asig. 2ª
ELSE
mos la direccion
del NOP de la ultima
TZE generada, indica
en la pila.
7702 0
2 0
7701 0
1 0
TRA **
NOP

INPUT	ACCIONES	PILA	GENERACION
asig. 3º			codigo asig 3º
;	FE=0 dar la direccion del NOP generado a las TRA cuyas direcciones estan en la pila marcados por 77	vaciar pila	NOP

Al final obtenemos un código de la forma: