

## PREPROCESADOR DE ALGOL 60 (GRAMATICA LL(1))

Por Pedro Díaz Muñoz, Teodomiro García Vadillo y Andrés Morales Martos

### I. INTRODUCCION

Dado un lenguaje de programación más o menos distante de cada "código de máquina" en particular, surge evidentemente el problema de la conversión y análisis de las cadenas alfanuméricas, que van siendo introducidas por uno u otro medio físico en el ordenador. A todo este proceso se refieren las técnicas de compilación, de las que veremos a continuación algún caso particular.

Dentro del todo que constituye un compilador para un lenguaje dado, veremos aquí con detalle un ejemplo de la primera de sus fases, la denominada preproceso, que posee una entidad propia dentro del conjunto, debido a una serie de razones que así lo aconsejan.

La construcción del preprocesador incluye como primer paso la elección de lo que denominaremos clases sintácticas. La elección de estas es más o menos arbitraria, aunque en cualquier caso hay ciertas clases típicas en todos los casos, tales como: identificador, palabra reservada, entero, operador, etc. El preprocesador se encarga de la lectura del programa en su forma física correspondiente y del encuentro y clasificación de todo ese texto dentro del conjunto total de las clases sintácticas en principio elegidas; proporcionará una salida numérica identificadora de cada clase encontrada y esto habrá de ser lo leído y tratado en fases posteriores.

Se dan una serie de buenas razones para la separación del preproceso de los siguientes tratamientos tales como una reducción en el tiempo de compilación, la automatización a que puede reducirse la construcción de preprocesadores y algunas otras.

El resto de los instrumentos de compilación o en propiedad lo que denominaremos analizador sintáctico solicitará salidas paulatinas de la fase de preproceso, a medida que éstas sean necesarias y este flujo continuo de información permitirá un mejor aprovechamiento de memoria, al no ser necesario tener guardado en la misma todo el conjunto de salidas del preproceso.

Daremos a continuación un ejemplo particular de construcción de un preprocesador. Ilustraremos de forma esencial los problemas de elección de clases y forma automática de construcción del mismo, auxiliados por una gramática eficiente en este caso. Elección de clases y relación gramática-programa preprocesador son los puntos clave de lo que sigue. Se incluye como apéndice un listado del programa realizado en M.A.P. con alguna ayuda de macro-instrucciones, así como también de varias subrutinas auxiliares en lenguaje FORTRAN. Una mejor explicación de cada punto será dada en su momento oportuno.

## 2. SISTEMA EMPLEADO

### 2.1. - Fases del sistema

El sistema consta esencialmente de dos fases. La primera lleva a cabo un análisis morfológico de las cadenas de símbolos, asociando a cada una su clase sintáctica correspondiente, de acuerdo con la gramática dada previamente. La segunda fase consistirá en la construcción de tablas necesarias para el análisis sintáctico.

La salida podría haberse realizado también, enviando como información para el análisis sintáctico el código de la clase sintáctica correspondiente a una cadena de símbolos y dicha cadena de símbolos, en lugar de dar el código de clase sintáctica y la posición que ocupa dicha cadena de símbolos en la tabla que se va construyendo. A nivel sintáctico el primer método resulta más complicado, ya que implica la construcción de tablas con el siguiente aumento de tiempo y de posiciones de memoria, sin embargo en la posterior fase de generación de código es conveniente tener estas tablas.

EJEMPLO:

Supongamos el identificador CASA y sea 13 el código de identificador.

El primer método dará como salida dos números N1 y N2. N1 será 13 (código del identificador) y N2 dará la posición en la tabla de identificadores, construída paralelamente al análisis, del identificador CASA.

El segundo método dará como salida 13 CASA; es decir el código seguido por el identificador en particular.

### 2.2. - Análisis Morfológico

El análisis morfológico puede realizarse por distintos métodos. Usaremos el método denominado Top down o también llamado análisis descendente, que partiendo del axioma llega a los terminales. Veamos esto con más detalle. Supongamos la cadena de símbolos terminales  $x_1 \dots x_n$  y por supuesto una gramática que dé las producciones de las clases sintácticas. Partimos del axioma  $\langle US \rangle$  y trataremos de encontrar una producción de la forma  $\langle US \rangle \longrightarrow x_1 \langle A_1 \rangle$

Una vez hallada buscaremos una de la forma  $\langle A_1 \rangle \longrightarrow x_2 \langle A_2 \rangle$  y así sucesivamente hasta llegar a una última producción  $\langle A_{n-1} \rangle \longrightarrow x_n$ , llegando a este punto queda reconocida la cadena de símbolos y puede ser emitido el código de la unidad sintáctica correspondiente.

Evidentemente en el caso de que la gramática cumpla los cuatro axiomas de Knuth - este es posible hacerlo sin ninguna dificultad, ya que siempre encontraremos en cada paso una y sólo una producción a la que ir. (Existe una ya que siempre suponemos que no hay errores en la cadena de entrada).

En caso de que no se cumpliera alguna de estas reglas, podrá realizarse este proceso



solo en algunos casos y con bastante dificultad; si por ejemplo no se cumple el segundo de los axiomas podríamos tener dos producciones:  $\langle A_1 \rangle \rightarrow b \langle C \rangle$   $\langle A_1 \rangle \rightarrow b \langle D \rangle$  en este caso en una cadena de la forma .....b d ....., al llegar a b tomaríamos la primera de las producciones y si nos encontráramos con que  $\langle C \rangle$  nunca generaba d ....., habríamos de volver hacia atrás y probar con la otra regla. Incluso podrían presentarse casos más complicados que harían preciso una técnica de vuelta atrás con la correspondiente pérdida de tiempo y dificultad en la implementación.

En nuestro caso, para evitar estas dificultades, exigiremos que la gramática sea tal que con el análisis de un solo caracter de la cadena de entrada conozcamos la producción por la que decidimos, es decir que la gramática sea decidible y determinista, para lo que son necesarios y suficientes las cuatro condiciones de Knuth.

### 2.3. - Tablas

Utilizaremos tablas Hash para todas las clases en que sea necesario el empleo de estas. Explicaremos más detalladamente el método de construcción de las mismas en la parte correspondiente a programa.

## 3. GRAMATICA

### 3.1. - Elección de las clases

El sistema empleado en la construcción del preprocesador en particular, así como la optimización del tiempo de compilación, son los factores determinantes de la elección de las clases sintácticas.

En cualquier caso, con ciertas clases típicas en todos los casos se consiguen buenos resultados. Estas con más o menos variantes han sido las elegidas en lo que sigue a continuación. Hay que hacer notar también la influencia del lenguaje de programación en la elección de clases sintácticas. Así en el ejemplo que nos ocupa esta causa nos ha hecho añadir la clase STRING que solo en algunos lenguajes existe.

Básicamente nuestras clases sintácticas son las siguientes:

CARACTER DE PUNTUACION  
 NUMERO  
 PALABRA RESERVADA  
 STRING  
 IDENTIFICADOR

La clase Caracter de puntuación abarca tanto caracteres simples como compuestos, considerándose también los operadores aritméticos (\*, +, -, /) como caracteres de puntuación.

La clase número comprende a los enteros y reales en coma fija. Los reales en coma flotante (p.e. 21.3104) se tratan como :  $\langle \text{número} \rangle \langle \text{caracter de puntuación} \rangle \langle \text{número} \rangle$ .

La clase palabra reservada incluye todas las del ALGOL; en nuestro programa, con fines puramente didácticos, hemos reducido este conjunto de palabras a las que empiezan por las letras: A, B, C, D, E, y U.

En la implementación en 7090 de estas palabras reservadas las mismas vienen encuadradas entre comillas (p.e. 'EQUAL'), la clase sintáctica palabra reservada abarca tanto la palabra en sí como ambas comillas.

La clase String en su implementación en 7090 consiste en una cadena de símbolos limitados en su principio por '(' (1). De nuevo estos caracteres de puntuación no son tratados como tales y se incluyen en la propia clase string.

La clase Identificador abarca toda clase de variables del lenguaje; es decir cualquier cadena de símbolos, empezando por una letra y no conteniendo ningún carácter de puntuación.

Las restricciones usadas en este trabajo con respecto de las clases sintácticas usualmente empleadas en ALGOL vendrán dadas paralelamente a la gramática, donde todas ellas vienen reflejadas.

### 3.2. - Gramática empleada

Como ya hemos explicado en 2.2. utilizaremos una gramática del tipo LL(1). A partir de la carta sintáctica ALGOL obtenemos una gramática B.N.F. y modificando las reglas llegamos a la gramática LL(1) referida, que damos a continuación.

#### GRAMATICA

```

<US> ::= letra <ID> / cifra <RN> / '<RPPS>' / <CP> / <IGNORE>
<ID> ::= letra <ID> / cifra <ID> / Ø
<RN> ::= cifra <RN> / <PPD> / Ø
<PPD> ::= cifra <RPD> / <RP>
<RPD> ::= cifra <RPD> / Ø
<RPPS> ::= ('<RS>' / <RPR>
<RPR> ::= A <A> / B <B> / COMENT' / DO' / E <E> / UNTIL'
<A> ::= ND' / RRAY'
<B> ::= EGIN' / OOLEAN'
<E> ::= ND' / Q EQ / LSE'
<EQ> ::= U' / '
<RS> ::= <NOTA1> <RS> / '<PFS>'
<PFS> ::= ) <RPFS> / <NOTA2> <RS>
<RPFS> ::= ' / <NOTA1> <RS>
<CP> ::= , / + / - / ) / * <RA> / / <RB> / . <RP> / ( <RPA>
<RA> ::= * / Ø
<RB> ::= / / ) / Ø
<RP> ::= . / , / = / Ø
<RPA> ::= / / Ø

```

(1) - Y en su final por ')'



## NOTAS.-

<IGNORE > puede ser o un espacio en blanco o \$

<NOTA 1 > es cualquier símbolo excepto '

<NOTA2 > es cualquier símbolo excepto )

Las letras son todas las posibles letras del alfabeto aceptadas por el ordenador

Las cifras son 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

### 3.3. - Comentarios

3.3.1. - Aun cuando según la gramática, y por tanto en todo el proceso de análisis morfológico, se puede reconocer un identificador con un número indefinido de caracteres, al haberse construido la tabla Hash de identificadores con una sola posición de memoria para almacenar cada identificador, estos solo pueden constar de seis caracteres a lo sumo.

3.3.2. - No se admiten espacios en blanco dentro de identificadores o números. Si, por ejemplo, aparece la cadena de símbolos P EPE, entonces se emitirá código de identificador para P, un código especial de carácter ignorado para el espacio blanco, y un nuevo código de identificador para EPE. Este código especial de carácter ignorado es utilizado también ante los símbolos \$ y = cuando este último no lleva un punto delante.

3.3.3. - No se admite como número : . "entero", ni tampoco : "entero".; en cualquiera de los dos casos emitiremos un código de carácter de puntuación y otro de número . Esta restricción es necesaria para que se cumpla la segunda condición de Knuht.

3.3.4. - No se admite introducir un STRING dentro de otro STRING, pues ello implicaría una vuelta atrás.

## 4. PROGRAMA

### 4.1. - Funcionamiento del programa

Ante una sentencia en lenguaje ALGOL, que será la entrada, vamos procesando carácter a carácter. El carácter a procesar le almacena en la posición CARCO, y una vez procesado pasa a CARCO-1, entrando en CARCO el carácter siguiente. Cada vez que mediante la gramática quede reconocida una cadena de estos caracteres como perteneciente a una clase sintáctica, se almacenará en la posición CARCUR el código de esta cadena, enviando a CARCUR-1 el código almacenado en CARCUR, y se imprimirá el contenido de estas cuatro posiciones. Es decir, a cada cadena que pertenezca a una clase sintáctica se le asignará un código propio; si la cadena pertenece a la clase de las palabras reservadas, se le asigna directamente el código de esa palabra reservada, prefijado de antemano; análogamente si es un carácter de puntuación. Si es un identificador, número o string el código no irá fijado antes de iniciar el programa, sino que vendrá determinado por la posición en sus tablas respectivas de dicha cadena.

### 4.2. - Explicación de Macros

LEER.- Lee una tarjeta completa de datos y la almacena de las posiciones IBUF a

IBUF + 80 (un caracter en cada posición). Almacena en CARCO el primero de los caracteres.

LLAMAR - NOMBRE - . - Da control a la rutina cuyo nombre aparece a la derecha; aumenta en 1 la dirección del lugar más alto de la pila ocupado. Controla si se ha saturado la pila, emitiendo en este caso un mensaje de OVERFLOW y detiene el programa. Al acabar la rutina NOMBRE se devuelve el control a la rutina que llamó a LLAMAR.

ENTRAR - NOMBRE - . - Da control a la rutina NOMBRE.

SALIR.- Devuelve control a la última dirección guardada en la pila y disminuye en una posición el contenido de la pila y compara con el principio de la pila.

VOLVER.- Sirve como delimitador inferior de rutina.

RUTINA.- Sirve como delimitador superior de la rutina.

DECIDE L, (X1, X2, ....., XN).- Compara el caracter de CARCO con todos los X1, ...XN y en caso de alguna coincidencia da control a L. En caso contrario pasa a la instrucción siguiente.

TEST - NUMERO - . - Examina si el caracter de CARCO coincide con el código numérico NUMERO. En caso contrario emite el caracter de error. Pasa el contenido de CARCO a CARCO-1 y en CARCO introduce el caracter siguiente.

PUT - NUMERO - . - Pasa el contenido de CARCUR a CARCUR-1 y almacena en CARCUR NUMERO.

#### 4.3. - Paso de la gramática al programa:

Ante unas producciones del tipo

$$A - B_1 B_2 \dots B_m / C_1 C_2 \dots C_n / \dots / G_1 G_2 \dots G_r$$

escribimos:

RUTINA A  
 DECIDE L<sub>1</sub>, (TER (B<sub>1</sub>)) (Terminal inicial de B<sub>1</sub>)  
 DECIDE L<sub>2</sub>, (TER (C<sub>1</sub>))  
 .....

L<sub>1</sub> TEST B<sub>1</sub>  
 LLAMAR B<sub>2</sub>  
 .....  
 .....  
 ENTRAR B<sub>m</sub>

L<sub>2</sub> TEST C<sub>1</sub>  
 .....  
 .....

L <sub>r</sub>	TEST	G <sub>1</sub>
	LLAMAR	G <sub>2</sub>
	.....	
	ENTRAR	G <sub>r</sub>
	VOLVER	

En caso de que B<sub>1</sub>, por ejemplo, tuviera varios terminales iniciales el DECIDE se realizará sobre todos ellos y se sustituye TEST B<sub>2</sub> por AVANZA 1.

#### 4.4. - Tablas

Este programa construirá tres tablas, una para identificadores otra para constantes y otra para strings. Explicaremos cada una separadamente.

##### 4.4.1. - Tabla de variables o identificadores.

Para esta tabla se precisa, como ya se hizo notar en 3.3., que los identificadores consten a lo sumo de 6 caracteres.

##### 4.4.1.1. - Descripción.

La tabla es de tipo Hash: las variables son almacenadas en un vector que en nuestro programa ocupará 180 posiciones. Las variables que consten de un solo caracter se irán almacenando dentro de las 30 primeras posiciones por orden de aparición sin repetirse ninguna (si una variable ha sido ya almacenada no se vuelve a almacenar).

Las variables de dos caracteres se almacenarán en las 30 siguientes posiciones y así sucesivamente.

##### 4.4.1.2. - Programa.

La tabla es construída por medio de una rutina TAB en Fortran; esta rutina necesita como argumentos el identificador y el número de caracteres de dicho identificador. Para esto en la rutina ID se van almacenando los caracteres que se van reconociendo como pertenecientes al identificador en una posición INIC mientras que mediante un contador N se van contando estos caracteres.

La rutina busca entre las posiciones correspondientes a variables de N caracteres el identificador que se acaba de reconocer, caso de encontrarlo devuelve el control a ID caso de no encontrarlo lo almacena en la primera de las posiciones reservada a identificadores de N caracteres que no esté ocupada.

##### 4.4.1.3. - Salida.

La rutina TAB devuelve a ID la posición relativa en la tabla del identificador en cuestión, este será un número de 3 cifras. Este número es almacenado en CARCUR delante del código correspondiente a identificador.



#### 4.4.2. - Tabla de constantes

##### 4.4.2.1. - Descripción.

Las constantes son primero evaluadas y seguidamente son almacenadas en un vector secuencialmente por orden de aparición sin mirar si están repetidas como en el caso de los identificadores. Esto está justificado ya que no es corriente que una constante se repita mucho en un programa y por tanto la pérdida de posiciones de memoria no será muy grande, mientras que como se verá a continuación se ahorrará algún tiempo al no ser la salida tan complicada.

##### 4.4.2.2. - Programa.

La rutina en Fortran TAB2 necesita como dato únicamente la constante ya evaluada. Esta es evaluada dentro de la rutina RN desde donde se llama a TAB2 que simplemente almacena la constante en la primera posición libre del vector y devuelve control a RN.

##### 4.4.2.3. - Salida.

Como salida se emite en CARCUR únicamente el código de constante la localización de la constante determinada dentro del vector la puede realizar el analizador sintáctico mediante un Pointer que señale a la posición de la primera constante todavía no procesada por dicho analizador, una vez analizada dicha constante se aumentará en 1 la posición señalada por el Pointer.

#### 4.4.3. - Tabla de Strings.

El principal problema para la construcción de una tabla de cadenas es la gran cantidad de memoria que se ocupa, hemos exigido que una cadena pueda tener a lo sumo 60 caracteres (incluyendo ' ( ' y ' ) ' ) y que en un programa pueda haber a lo sumo 10 cadenas con esto el número de posiciones ocupadas por la tabla será 100.

El proceso empleado será análogo al utilizado en la tabla de números y por tanto no lo vamos a repetir, la única diferencia es que ahora no se hace una evaluación, sin embargo habrá que almacenar de forma "compacta" (6 caracteres por palabra de memoria) la cadena reconocida en un vector que se mandará a la rutina TAB3 cuya función será análoga a la de la TAB2 (aun cuando ahora construirá una matriz 10x10 en lugar de un vector).

Para la discusión de la salida también se aplicará lo que se ha dicho en 4.4.2.3.

## APENDICE

Lista de abreviaturas empleadas en la gramática:

ID = resto de identificador

RN = resto número

RPPS = resto posible pal, res, o string



CP = caracter puntuación  
 RI = resto de resto de identificador  
 PPD = posible parte decimal  
 RP = resto caracter que empieza por .  
 RA = resto caracter que empieza por  
 RB = resto caracter que empieza por /  
 RPA = resto caracter que empieza por (  
 RPD = resto parte decimal  
 RS = resto string  
 A = resto palabra reservada que empieza por A  
 B = resto palabra reservada que empieza por B  
 E = resto palabra reservada que empieza por E  
 EQ = resto palabra reservada que empieza por EQ  
 PFS = posible fin de string  
 SIMBOLO-´ = cualquier símbolo que no sea ´  
 SIMBOLO-) = cualquier símbolo que no sea )  
 RPFS = resto posible fin de string

#### Representaciones internas de los caracteres:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
17	18	19	20	21	22	23	24	25	33	34	35	36	37	38	39	40	41	50	51	52	53	54	55	56	57

b = '\$'

48 43 12 En los demás signos de puntuación coinciden representación interna y código.

#### LISTA DE CODIGOS

NUMERO = 13	.	.	= 264
IDENTIFICADOR = 10	.	,	= 265
STRING = 30	.	=	= 266
IGNORE = 99	(	/	= 267
,	'BEGIN'		= 151
+	'BOOLEAN'		= 152
-	'AND'		= 153
)	'ARRAY'		= 154
(	'COMENT'		= 155
.	'DO'		= 156
/	'ELSE'		= 157
* *	'END'		= 158
//	'EQ'		= 159
*	'EQV'		= 160
/)	'UNTIL'		= 179