# Características del lenguaje

El lenguaje que se nos ha asignado cuenta de los siguientes componentes:

* Según el formato de PASCAL
* Palabras reservadas en inglés
* Listas
* Funciones
* for (formato PASCAL)

# Descripción del lenguaje en EBNF

<Programa> ::= <Cabecera\_programa> <bloque>

<bloque> ::= <Inicio\_de\_bloque>

<Declar\_de\_variables\_locales>

<Declar\_de\_subprogs>

<Sentencias>

<Fin\_de\_bloque>

<Declar\_de\_subprogs> ::= <Declar\_de\_subprogs> <Declar\_subprog>

|

<Declar\_subprog> ::= <Cabecera\_subprog> <bloque>

<Declar\_de\_variables\_locales> ::= <Marca\_ini\_declar\_variables>

<Variables\_locales>

<Marca\_fin\_declar\_variables>

|

<Marca\_ini\_declar\_variables> ::= VAR

<Marca\_fin\_declar\_variables> ::= ENDVAR

<Cabecera\_programa> ::= PROGRAM <identificador>

<Inicio\_de\_bloque> ::= BEGIN

<Fin\_de\_bloque> ::= END

<Variables\_locales> ::= <Variables\_locales> <Cuerpo\_declar\_variables>

| <Cuerpo\_declar\_variables>

<Cuerpo\_declar\_variables> ::= <Cuerpo\_declar\_variables> <lista\_identificadores> : <tipo\_dato>;

|

<lista\_identificadores> ::= <lista\_identificadores>, <identificador>

| <identificador>

<tipo\_dato> ::= <tipo\_dato\_A>

| <tipo\_dato\_B>

<tipo\_dato\_A> ::= INTEGER

| REAL

| CHAR

| BOOLEAN

<tipo\_dato\_B> ::= LIST OF <tipo\_dato\_A>

<Cabecera\_subprog> ::= FUNCTION <identificador> (<Variables\_locales>) : <tipo\_dato>;

<bloque>;

<sentencia\_return> ::= RETURN <expresion>

<Sentencias> ::= <Sentencias> ; <Sentencia>

| <Sentencia>

<Sentencia> ::= <bloque>

| <sentencia\_asignacion>

| <sentencia\_if>

| <sentencia\_while>

| <sentencia\_for>

| <sentencia\_entrada>

| <sentencia\_salida>

| <sentencia\_return>

<sentencia\_asignacion> ::= <identificador> := <expresion>

<sentencia\_if> ::= <alternativa\_simple>

| <alternativa\_doble>

<alternativa\_simple> ::= IF <expresion> THEN <Sentencia>

<alternativa\_doble> ::= IF <expresion> THEN <Sentencia> ELSE <Sentencia>

<sentencia\_while> ::= WHILE <expresion> DO <Sentencia>

<sentencia\_for> ::= FOR <sentencia\_asignacion> TO <expresion> DO <Sentencia>

<sentencia\_entrada> ::= <nomb\_entrada> <lista\_variables>

<sentencia\_salida> ::= <nomb\_salida> <lista\_expresiones\_o\_cadena>

<expresion> ::= ( <expresion> )

| <op\_unario> <expresion>

| <expresion> <op\_binario> <expresion>

| <expresion> <op\_ter2> <expresion>

| <expresion> <op\_ter1> <expresion> <op\_ter2> <expresion>

| <identificador>

| <constante>

| <agregado>

| <funcion>

<nomb\_entrada> ::= SCAN

<nomb\_salida> ::= PRINT

<lista\_variables> ::= <identificador>, <lista\_variables>

| <identificador>

<lista\_expresiones\_o\_cadena> ::= <lista\_expresiones\_o\_cadena> <exp\_cad>

|

<exp\_cad> ::= <expresion>

| <cadena>

<lista\_expresiones> ::= <expresion>, <lista\_expresiones>

| <expresion>

<agregado> ::= [<lista\_constantes>]

<lista\_constantes> ::= <constante> <lista\_constantes>

| <constante>

<funcion> ::= <identificador> (<lista\_expresiones>)

<op\_unario> ::= #

| ?

| >>

| <<

| $

| +

| -

| NOT

<op\_binario> ::= +

| \*

| -

| /

| --

| \*\*

| %

| @

| AND

| OR

| =

| <>

| >

| <

| <=

| >=

<op\_ter1> ::= ++

<op\_ter2> ::= @

<constante> ::= '[a-z][0-9][A-Z]'

| [0-9]+

| [0-9]+.[0-9]+

| TRUE

| FALSE

<identificador> ::= [a-Z]([a-Z][0-9])\*

<cadena> ::= "([a-Z][0-9])\*"

# Definición de la semántica en lenguaje natural

1. **Introducción**

Vamos a realizar una descripción básica de nuestro lenguaje de programación basado en el lenguaje Pascal y añadiendo algunas modificaciones.

1. **Estructura de un programa**

|  |  |
| --- | --- |
| **Objetivo** | Definir un nuevo programa |
| **Formato** | PROGRAM <identificador> <bloque> |
| **Descripción** | Para comenzar un nuevo programa debemos indicarlo mediante la palabra clave PROGRAM, seguido de un identificador con el nombre que queramos asignarle. Por último, indicaremos el bloque de código que lo compondrá. |
| **Ejemplo** | **PROGRAM** ejemplo  BEGIN  VAR  a, b : Integer;  ENDVAR  FUNCTION Sumar (n1, n2 : Integer) : Integer;  BEGIN  return n1 + n2;  END  a := 1;  b := 2;  return Sumar (a, b);  END |

* 1. **Encabezamiento**

|  |  |
| --- | --- |
| **Objetivo** | Definir el encabezamiento del programa |
| **Formato** | program <identificador> |
| **Descripción** | En la cabecera del programa se especifican el nombre y parámetros del programa. Es meramente informativa y se sitúa en la primera línea del programa encabezada por la palabra reservada program. |
| **Ejemplo** | **PROGRAM** ejemplo; |

* 1. **Definición de un bloque**

|  |  |
| --- | --- |
| **Objetivo** | Definir un nuevo bloque de código |
| **Formato** | BEGIN  [**VAR** Declaración de variables locales **ENDVAR**]  [Declaración de subprogramas]  Sentencias del programa  END |
| **Descripción** | Para indicar la declaración de un nuevo bloque de código, utilizamos la palabra clave BEGIN.  A continuación, y de manera opcional, pasamos a la declaración de variables locales y subprogramas. Una vez realizadas las declaraciones, introducimos el conjunto de sentencias a ejecutar.  Por último, cerramos el bloque con la palabra clave END. |
| **Ejemplo** | **BEGIN**  VAR  mi\_var : Integer;  ENDVAR  mi\_var := 1;  return mi\_var;  **END** |

* 1. **Declaración de variables locales**

|  |  |
| --- | --- |
| **Objetivo** | Declarar una o varias variables |
| **Formato** | <identificador> (, <identificador>)\* : <tipo\_dato>; |
| **Descripción** | Mediante esta sentencia se pueden declarar una o más variables de diferentes tipos usando los siguientes tipos de datos:  Integer : Define una variable de tipo entero  Char : Define una variable de tipo carácter  Real : Define una variable de tipo real  Boolean : Define una variable de tipo lógica |
| **Ejemplo** | numero : Integer;  letra1, letra2 : Char; |

* 1. **Declaración de subprogramas**

|  |  |
| --- | --- |
| **Objetivo** | Declarar un subprograma |
| **Formato** | FUNCTION <identificador> ([Declaración de variables]) : <tipo\_dato>;  <bloque>; |
| **Descripción** | Para declarar un subprograma, especificamos su cabecera mediante la palabra clave FUNCTION, seguida del nombre del subprograma. A continuación especificamos las variables que servirán como argumentos, y por último, el tipo de dato devuelto.  A continuación, proporcionamos el bloque de instrucciones que compondrá el subprograma. |
| **Ejemplo** | **FUNCTION** funcion\_ej (a, b : Integer; c : Char) : Integer;  BEGIN  VAR  res : Integer;  ENDVAR  print [c];  res = a + b;  return res;  END |

1. **Tipos de datos**
   1. **Integer (Números enteros)**
   2. **Char (Caracteres)**
   3. **Real (Números reales)**
   4. **Boolean (Valores lógicos: TRUE y FALSE)**
   5. **List (explicado en otro apartado)**
2. **Operadores y expresiones**
   1. **Aritméticos**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Nombre** | **Tipo** | **Lexema** | **Operandos** | **Ejemplo** | **Resultado** |
| Suma | binario | + | dos valores de tipo entero o real | 3 + 3  4.0 + 3.4 | Devuelve la suma de los dos valores |
| Resta | binario | - | dos valores de tipo entero o real | 3 - 3  4.0 - 3.4 | Devuelve la resta de los dos valores |
| Multiplicación | binario | \* | dos valores de tipo entero o real | 4 \* 2  5.0 \* 2 | Devuelve la multiplicación de los dos valores |
| División | binario | / | dos valores de tipo entero o real | 4 / 2  5.0 / 2 | Devuelve la división de el primer valor entre el segundo |
| Módulo | binario | % | dos valores de tipo entero o real | 4 % 2  5.0 % 2 | Devuelve el resto entero de realizar la división de el primer valor entre el segundo |

* 1. **Lógicos**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Nombre** | **Tipo** | **Lexema** | **Operandos** | **Ejemplo** | **Resultado** |
| AND | binario | AND | dos valores booleanos | TRUE AND FALSE | Devuelve el resultado (TRUE o FALSE) de realizar la operación lógica AND |
| OR | binario | OR | dos valores booleanos | TRUE OR FALSE | Devuelve el resultado (TRUE o FALSE) de realizar la operación lógica OR |
| NOT | unario | NOT | valor booleano | NOT TRUE | Devuelve el resultado (TRUE o FALSE) de realizar la operación lógica NOT |
| Igual | binario | = | dos valores del mismo tipo | 3 = 3  3.5 = 3.5  ‘a’ = ‘b’  [1, 2] = [2, 1] | Devuelve el resultado (TRUE o FALSE) de realizar la comparación de igualdad |
| Distinto | binario | <> | dos valores del mismo tipo | 3 <> 3  3.5 <> 3.5  ‘a’ <> ‘b’  [1, 2] <> [2, 1] | Devuelve el resultado (TRUE o FALSE) de realizar la comparación de diferencia |
| Mayor estricto | binario | > | dos valores de tipo entero o real | 1 > 10 | Devuelve el resultado (TRUE o FALSE) de realizar la operación lógica “>” |
| Menor estricto | binario | < | dos valores de tipo entero o real | 6 < 2 | Devuelve el resultado (TRUE o FALSE) de realizar la operación lógica “<” |
| Mayor o igual | binario | >= | dos valores de tipo entero o real | 3 >= 5 | Devuelve el resultado (TRUE o FALSE) de realizar la operación lógica “>=” |
| Menor o igual | binario | <= | dos valores de tipo entero o real | 3 <= 5 | Devuelve el resultado (TRUE o FALSE) de realizar la operación lógica “<=” |

1. **Operadores de entrada y salida**

|  |  |
| --- | --- |
| **Objetivo** | Obtener datos de entrada a través del teclado |
| **Formato** | **SCAN** <nombre\_variable>**;** |
| **Descripción** | Entrada de datos del programa mediante el uso del teclado. Se captará todo carácter o número introducido hasta pulsar la tecla return. |
| **Ejemplo** | **SCAN** x**;** |

|  |  |
| --- | --- |
| **Objetivo** | Imprimir variables o una cadena por pantalla |
| **Formato** | **PRINT** <lista\_expresiones\_o\_cadena>**;** |
| **Descripción** | Es la salida del sistema. Nos permite imprimir por pantalla una lista de expresiones o cualquier cadena de caracteres. |
| **Ejemplo** | **PRINT** “Hola mundo!”;  x := 10;  **PRINT** x; |

1. **Estructuras de control**
   1. **IF**

|  |  |
| --- | --- |
| **Objetivo** | Introducir una instrucción condicional |
| **Formato** | **IF** **(** <expresión\_lógica> **)**  **THEN** <sentencia>  Con la opción else:    **IF (** <expresión\_lógica> **)**  **THEN** <sentencia\_o\_bloque\_1>  **ELSE** <sentencia\_o\_bloque\_2> |
| **Descripción** | La expresión que sigue al IF representa la condición a evaluar. Si la condición es TRUE se ejecuta la parte THEN, si es FALSE se ejecuta la parte ELSE o ninguna parte si no la hay.  Notar que después de THEN y ELSE no debe escribirse el punto y coma. Si se escribe precediendo a ELSE, se terminará allí la sentencia IF y se producirá un error de compilación. |
| **Ejemplo** | **IF** seguir  **THEN**  BEGIN  **IF** (B <> 1)  **THEN**  C := 1  **ELSE**  D := 1  END; |

* 1. **WHILE**

|  |  |
| --- | --- |
| **Objetivo** | Introducir una instrucción bucle de tipo while |
| **Formato** | **WHILE** **(** <expresión\_lógica> **)** **DO** <sentencia\_o\_bloque> |
| **Descripción** | Es una sentencia de control iterativa. Repite se sentencia de ejecución mientras se cumpla la condición de nuestra expresión lógica. La sentencia que se repite puede ser compuesta.  Notar que la expresión se evalúa antes de ejecutar la sentencia. Si el valor inicial es FALSE, no se ejecutará la sentencia ninguna vez.  La sentencia que se repite debe modificar el valor de la expresión, si no resulta un bloque sin salida.  Para ejecutar un grupo de sentencias, hay que construir una sentencia compuesta con los delimitadores BEGIN y END. |
| **Ejemplo** | **WHILE** (i < N) **DO**  BEGIN  potencia := potencia \* x;  i := i+1  END |

* 1. **FOR**

|  |  |
| --- | --- |
| **Objetivo** | Introducir una instrucción bucle de tipo for |
| **Formato** | **FOR** <sentencia\_asignación> **TO** <expresion> **DO** <sentencia\_o\_bloque> |
| **Descripción** | Permite construir bucles repetitivos controlador por un contador inicializado en la sentencia de asignación.  La variable de control, el valor inicial y el valor final deben ser todos del mismo tipo ordinal. Las sentencias que se repiten (rango del bucle) no deben modificar el valor de la variable de control. |
| **Ejemplo** | **FOR** i := 0 **TO** 10 **DO**  PRINT“HOLA”; |

1. **Listas**

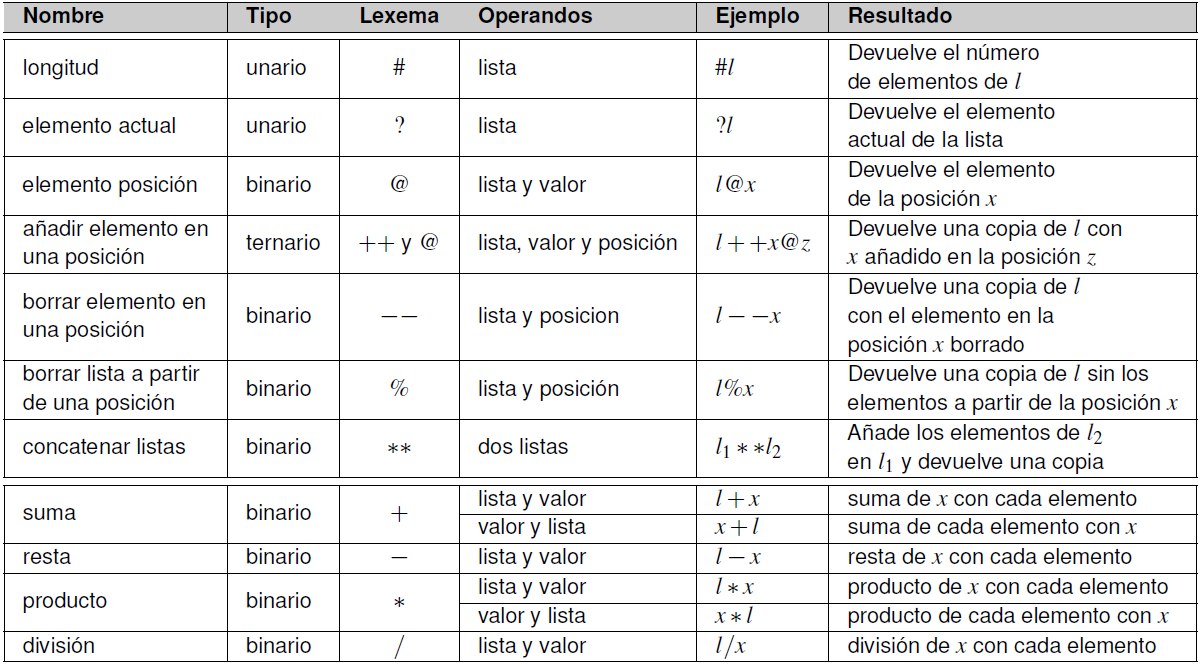
|  |  |
| --- | --- |
| **Objetivo** | **Crear** una variable de tipo lista |
| **Formato** | <identificador> (,<identificador>)\* **: LIST OF** <tipo\_basico>**;** |
| **Descripción** | No podemos crear listas recursivas. Solo podemos crear listas de tipos básicos. En la misma definición no podemos inicializar la lista. |
| **Ejemplo** | **LIST OF** integer x; |

|  |  |
| --- | --- |
| **Objetivo** | **Asignar** valores a una variable de tipo lista |
| **Formato** | **<identificador> := [** <valor>,(<valores>)\***];** |
| **Descripción** | No podemos crear listas recursivas. Solo podemos crear listas de tipos básicos. En la misma definición no podemos inicializar la lista.  Los valores de la lista pueden ser valores, variables o constantes. |
| **Ejemplo** | x1 := [1,2,3]**;** |



Se definen los siguientes operadores, teniendo en cuenta que son formas de expresión

y no de sentencia, salvo que se indique como sentencia, es decir, se devuelve un nuevo elemento del tipo base de la lista o una nueva lista modificada tras realizar la operación correspondiente.



Tal y como muestra la tabla anterior, los operadores unarios tienen como argumento una lista de cualquier tipo base. Los operadores binarios manejan la lista de tipo base y, bien un elemento que debe ser del mismo tipo base o bien la posición que debe ser de tipo entero o ambos (operador ternario).

Existen tres operadores unarios para la realización del recorrido de una lista. Dos de ellos hacen desplazamientos, uno de retroceso y otro de avance y el tercero sitúa el cursor al comienzo de la lista.

Por último, la concatenación de listas debe actuar sobre listas que posean el mismo tipo base con independencia del número de elementos que en ellas se alojen.

# Identificación de tokens

Identificación de lexemas:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| identificador | % | , | FOR | BEGIN | <> |
| constante | @ | ; | = | END | > |
| cadena | ? | PRINT | RETURN | PROGRAM | < |
| + | >> | SCAN | FUNCTION | VAR | <= |
| \* | << | WHILE | LIST OF | ENDVAR | >= |
| - | $ | DO | INTEGER | NOT | ‘ |
| / | # | IF | REAL | AND | “ |
| -- | ( | THEN | CHAR | OR | TO |
| \*\* | ) | ELSE | BOOLEAN | := | : |
| ++ | [ | ] |  |  |  |

Identificación de lexemas con igual función semántica:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| identificador | % | , | FOR | BEGIN | <> |
| constante | @ | ; | = | END | > |
| cadena | ? | PRINT | RETURN | PROGRAM | < |
| + | >> | SCAN | FUNCTION | VAR | <= |
| \* | << | WHILE | LIST OF | ENDVAR | >= |
| - | $ | DO | INTEGER | NOT | ‘ |
| / | # | IF | REAL | AND | “ |
| -- | ( | THEN | CHAR | OR | TO |
| \*\* | ) | ELSE | BOOLEAN | := | : |
| ++ | [ | ] |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **Nombre** | **Código** | **Atributos** | **Expr. Regular** |
| MASMENOS | 256 | 0: +  1: - | "+" | "-" |
| OPBIN | 257 | 0: \*  1: /  2: --  3: \*\*  4: %  6: =  7: AND  8: OR  9: <>  10: >  11: <  12: >=  13: <= | "\*" | "/" | "--" | "\*\*" | "%" | "=" | "AND" | "OR" | "<>" | ">" | "<" | "<=" | ">=" |
| OPUNA | 258 | 0: ?  1: >>  2: <<  3: $  4: # | "?" | ">>" | "<<" | "$" | "#" |
| OPTER1 | 288 |  | “++” |
| OPTER2 | 289 |  | “@” |
| IDENT | 259 |  | [a-Z]+ |
| CONST | 260 | 0: "'[a-z][0-9][A-Z]'"  1: [0-9]+  2: [0-9]+.[0-9]+  3: "TRUE"  4: "FALSE" | "'[a-z][0-9][A-Z]'" | [0-9]+ | [0-9]+.[0-9]+ | "TRUE" | "FALSE" |
| CAD | 261 |  | ""([a-Z][0-9])\*"" |
| PARI | 262 |  | "(" |
| PARD | 263 |  | ")" |
| CORI | 290 |  | “[“ |
| CORD | 291 |  | “]” |
| COMA | 264 |  | "," |
| PYC | 265 |  | ";" |
| PRINT | 268 |  | "PRINT" |
| SCAN | 269 |  | "SCAN" |
| IF | 270 |  | "IF" |
| THEN | 271 |  | "THEN" |
| ELSE | 272 |  | "ELSE" |
| WHILE | 273 |  | "WHILE" |
| DO | 274 |  | "DO" |
| FOR | 275 |  | "FOR" |
| TO | 276 |  | "TO" |
| TIPOBASICO | 277 | 0: INTEGER  1: REAL  2: CHAR  3: BOOLEAN | "INTEGER" | "REAL" | "CHAR" | "BOOLEAN" |
| LISTOF | 278 |  | "LIST OF" |
| BEG | 279 |  | "BEGIN" |
| END | 280 |  | "END" |
| VAR | 281 |  | "VAR" |
| ENDVAR | 282 |  | "ENDVAR" |
| PROG | 283 |  | "PROGRAM" |
| FUNCTION | 284 |  | "FUNCTION" |
| RETURN | 285 |  | "RETURN" |
| ASIG | 286 |  | “:=” |
| DOSPU | 287 |  | “:” |

# Definición de la gramática abstracta

<Programa> ::= <Cabecera\_programa> <bloque>

<bloque> ::= <BEG>

<Declar\_de\_variables\_locales>

<Declar\_de\_subprogs>

<Sentencias>

<END>

<Declar\_de\_subprogs> ::= <Declar\_de\_subprogs> <Declar\_subprog>

|

<Declar\_subprog> ::= <Cabecera\_subprog> <bloque>

<Declar\_de\_variables\_locales> ::= <VAR>

<Variables\_locales>

<ENDVAR>

|

<Cabecera\_programa> ::= <PROG> <IDENT>

<Variables\_locales> ::= <Variables\_locales> <Cuerpo\_declar\_variables>

| <Cuerpo\_declar\_variables>

<Cuerpo\_declar\_variables> ::= <Cuerpo\_declar\_variables> <lista\_identificadores> <DOSPU> <tipo\_dato> <PYC>

|

<lista\_identificadores> ::= <lista\_identificadores> <COMA> <IDENT>

| <IDENT>

<tipo\_dato> ::= <TIPOBASICO>

| <LISTOF>

<TIPOBASICO> ::= INTEGER

| REAL

| CHAR

| BOOLEAN

<tipo\_dato\_B> ::= <LISTOF> <TIPOBASICO>

<Cabecera\_subprog> ::= <FUNCTION> <IDENT> (<Variables\_locales>) <DOSPU> <tipo\_dato> <PYC>

<bloque> <PYC>

<sentencia\_return> ::= <RETURN> <expresion>

<Sentencias> ::= <Sentencias> <PYC> <Sentencia>

| <Sentencia>

<Sentencia> ::= <bloque>

| <sentencia\_asignacion>

| <sentencia\_if>

| <sentencia\_while>

| <sentencia\_for>

| <sentencia\_entrada>

| <sentencia\_salida>

| <sentencia\_return>

<sentencia\_asignacion> ::= <IDENT> <ASIG> <expresion>

<sentencia\_if> ::= <alternativa\_simple>

| <alternativa\_doble>

<alternativa\_simple> ::= <IF> <expresion> <THEN> <Sentencia>

<alternativa\_doble> ::= <IF> <expresion> <THEN> <Sentencia> <ELSE> <Sentencia>

<sentencia\_while> ::= <WHILE> <expresion> <DO> <Sentencia>

<sentencia\_for> ::= <FOR> <sentencia\_asignacion> <TO> <expresion> <DO> <Sentencia>

<sentencia\_entrada> ::= SCAN <lista\_variables>

<sentencia\_salida> ::= PRINT <lista\_expresiones\_o\_cadena>

<expresion> ::= <PARI> <expresion> <PARD>

| <OPUNA> <expresion>

| <expresion> <OPBIN> <expresion>

| <expresion> <OPTER1> <expresion>

| <expresion> <OPTER1> <expresion> <OPTER2> <expresion>

| <IDENT>

| <CONST>

| <agregado>

| <funcion>

<lista\_variables> ::= <lista\_variables> <COMA> <IDENT>

| <IDENT>

<lista\_expresiones\_o\_cadena> ::= <lista\_expresiones\_o\_cadena> <exp\_cad>

|

<exp\_cad> ::= <expresion>

| <CAD>

<lista\_expresiones> ::= <lista\_expresiones> <COMA> <expresion>

| <expresion>

<agregado> ::= <CORI> <lista\_constantes> <CORD>

<lista\_constantes> ::= <lista\_constantes> <COMA> <CONST>

| <CONST>

<funcion> ::= <IDENT> <PARI> <lista\_expresiones> <PARD>

<OPUNA> ::= #

| ?

| >>

| <<

| $

| +

| -

| NOT

<OPBIN> ::= +

| \*

| -

| /

| --

| \*\*

| %

| @

| AND

| OR

| =

| <>

| >

| <

| <=

| >=

<OPTER1> ::= ++

<OPTER2> ::= @

<CONST> ::= '<alfanum>'

| <numero>

| <numero>.<numero>

| TRUE

| FALSE

<numero> ::= <numero><digito>

| <digito>

<IDENT> ::= <IDENT><alfanum>

| <letra>

<CAD> ::= "<cadena>"

<cadena> ::= <cadena><caracter>

|

<BEG> ::= BEGIN

<END> ::= END

<PROG> ::= PROGRAM

<DOSPU> ::= :

<PYC> ::= ;

<COMA> ::= ,

<LISTOF> ::= LIST OF

<FUNCTION> ::= FUNCTION

<RETURN> ::= RETURN

<ASIG> ::= :=

<IF> ::= IF

<THEN> ::= THEN

<ELSE> ::= ELSE

<WHILE> ::= WHILE

<DO> ::= DO

<FOR> ::= FOR

<TO> ::= TO

<PARI> ::= (

<PARD> ::= )

<CORI> ::= [

<CORD> ::= ]

# Definición de la gramática abstracta

<Programa> ::= <Cabecera\_programa> <bloque>

<bloque> ::= BEG

<Declar\_de\_variables\_locales>

<Declar\_de\_subprogs>

<Sentencias>

END

<Declar\_de\_subprogs> ::= <Declar\_de\_subprogs> <Declar\_subprog>

|

<Declar\_subprog> ::= <Cabecera\_subprog> <bloque>

<Declar\_de\_variables\_locales> ::= VAR

<Variables\_locales>

ENDVAR

|

<Marca\_ini\_declar\_variables> ::= VAR

<Marca\_fin\_declar\_variables> ::= ENDVAR

<Cabecera\_programa> ::= PROG IDENT

<Inicio\_de\_bloque> ::= BEGIN

<Fin\_de\_bloque> ::= END

<Variables\_locales> ::= <Variables\_locales> <Cuerpo\_declar\_variables>

| <Cuerpo\_declar\_variables>

<Cuerpo\_declar\_variables> ::= <Cuerpo\_declar\_variables> <lista\_identificadores> DOSPU <tipo\_dato> PYC

|

<lista\_identificadores> ::= <lista\_identificadores> COMA IDENT

| IDENT

<tipo\_dato> ::= TIPOBASICO

| LISTOF

<tipo\_dato\_A> ::= INTEGER

| REAL

| CHAR

| BOOLEAN

<tipo\_dato\_B> ::= LIST OF TIPOBASICO

<Cabecera\_subprog> ::= FUNCTION IDENT (<Variables\_locales>) DOSPU <tipo\_dato> PYC

<bloque> PYC

<sentencia\_return> ::= RETURN <expresion>

<Sentencias> ::= <Sentencias> PYC <Sentencia>

| <Sentencia>

<Sentencia> ::= <bloque>

| <sentencia\_asignacion>

| <sentencia\_if>

| <sentencia\_while>

| <sentencia\_for>

| <sentencia\_entrada>

| <sentencia\_salida>

| <sentencia\_return>

<sentencia\_asignacion> ::= IDENT ASIG <expresion>

<sentencia\_if> ::= <alternativa\_simple>

| <alternativa\_doble>

<alternativa\_simple> ::= IF <expresion> THEN <Sentencia>

<alternativa\_doble> ::= IF <expresion> THEN <Sentencia> ELSE <Sentencia>

<sentencia\_while> ::= WHILE <expresion> DO <Sentencia>

<sentencia\_for> ::= FOR <sentencia\_asignacion> TO <expresion> DO <Sentencia>

<sentencia\_entrada> ::= SCAN <lista\_variables>

<sentencia\_salida> ::= PRINT <lista\_expresiones\_o\_cadena>

<expresion> ::= PARI <expresion> PARD

| OPUNA <expresion>

| <expresion> OPBIN <expresion>

| <expresion> OPTER1 <expresion>

| <expresion> OPTER1 <expresion> OPTER2 <expresion>

| IDENT

| CONST

| <agregado>

| <funcion>

<nomb\_entrada> ::= SCAN

<nomb\_salida> ::= PRINT

<lista\_variables> ::= IDENT COMA <lista\_variables>

| IDENT

<lista\_expresiones\_o\_cadena> ::= <lista\_expresiones\_o\_cadena> <exp\_cad>

|

<exp\_cad> ::= <expresion>

| CAD

<lista\_expresiones> ::= <expresion> COMA <lista\_expresiones>

| <expresion>

<agregado> ::= CORI <lista\_constantes> CORD

<lista\_constantes> ::= CONST <lista\_constantes>

| CONST

<funcion> ::= IDENT PARI <lista\_expresiones> PARD

<op\_unario> ::= #

| ?

| >>

| <<

| $

| +

| -

| NOT

<op\_binario> ::= +

| \*

| -

| /

| --

| \*\*

| %

| @

| AND

| OR

| =

| <>

| >

| <

| <=

| >=

<op\_ter1> ::= ++

<op\_ter2> ::= @

<constante> ::= '[a-z][0-9][A-Z]'

| [0-9]+

| [0-9]+.[0-9]+

| TRUE

| FALSE

<identificador> ::= [a-Z]([a-Z][0-9])\*

<cadena> ::= "([a-Z][0-9])\*"