**Universidad de Granada**

*DEPARTAMENTO DE LENGUAJES Y SISTEMAS INFORMÁTICOS*

Práctica 3 – Procesadores de Lenguajes

**Diseño del Lenguaje**

**Autores**: Alexander Moreno Borrego

Carlos Jesus Fernandez Basso

Francisco Santolalla Quiñonero

**Correos:** [alexmobo@correo.ugr.es](mailto:alexmobo@correo.ugr.es)

[karloos@correo.ugr.es](mailto:karloos@correo.ugr.es)

[fransan@correo.ugr.es](mailto:fransan@correo.ugr.es)

**DNI:** 39906263-K

75927137-C

76439251-Q

**Profesor**:Salvador Villena Morales

Contenido

[Introducción 3](#_Toc356074913)

[Descripción formal 5](#_Toc356074914)

[Semántica en lenguaje natural 9](#_Toc356074915)

[Programa 9](#_Toc356074916)

[Declaración de variables 9](#_Toc356074917)

[Declaración de subprogramas (funciones) 9](#_Toc356074918)

[Sentencias del programa 9](#_Toc356074919)

[Identificación de los tokens 11](#_Toc356074920)

[Tabla de tokens 12](#_Toc356074921)

[Especificación Lex 14](#_Toc356074922)

[Especificación YACC 16](#_Toc356074923)

Introducción

El lenguaje a implementar será asignado por el profesor de prácticas y tendrá las siguientes características mínimas:

* Ser un subconjunto de un lenguaje de programación estructurado.
* Los identificadores debe ser declarados antes de ser usados.
* Los tipos de datos mínimos son: entero, real, carácter y booleano. Se definirán las operacionestípicas para cada uno de ellos, según se puede ver en la siguiente tabla:

|  |  |
| --- | --- |
| Tipo de dato | Operaciones |
| entero, real | suma, resta, producto, división, operaciones de relación |
| booleano | and, or, not, xor |

* Poseerá la sentencia de asignación para todos los tipos de expresiones.
* Permitirá expresiones aritméticas lógicas.
* Tendrá una sentencia de entrada y otra de salida (se utilizará como dispositivo de entrada el tecladoy de salida la pantalla). Además, la sentencia de entrada deberá permitir leer sobre una lista de identificadoresy la sentencia de salida deberá permitir escribir una lista de expresiones y/o constantesde tipo cadena. A diferencia de los lenguajes conocidos y usados como referencia, estas sentenciasno representan llamada a subprograma.
* Dispone de las estructuras de control siguientes:
  1. IF-THEN-ELSE
  2. WHILE
* Con independencia del tipo de lenguaje asignado:
  1. La estructura sintáctica del programa es:

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

* 1. En cualquier parte se podrán definir bloques como en C, es decir, tendremos una estructurasintáctica como la que se muestra a continuación:

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

<Declar\_de\_variables\_locales>

<Declar\_de\_subprogs>

<Sentencias>

<Fin\_de\_bloque>

* 1. Una sentencia puede ser, un bloque, por lo que se permite el anidamiento de bloques y subprogramas.
  2. La comprobación de tipos será como la del Pascal, es decir, fuertemente tipado.
  3. Para los argumentos de un subprograma, el mecanismo de paso de parámetros es por valor.
  4. No se permiten declaraciones fuera de los bloques. Las declaraciones deben ir entre una marcade inicio y otra de final de las declaraciones1.
  5. La estructura sintáctica de un subprograma será el siguiente:

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

* 1. El lenguaje debe admitir tanto las letras mayúsculas como las minúsculas, exceptuando aquellosque tengan lenguaje C que ofrece sensibilidad en este sentido, mientras que para el caso dePascal no sucede.
* El lenguaje debe incluir **cinco elementos nuevos**, además de los enumerados anteriormente:
  1. Sintaxis inspirada en un lenguaje de programación.
     + **Pascal.** fue diseñado y publicado para que los programadores pudieran aprenderlo fácilmente utilizando programación estructurada y estructuras de datos.

Se caracteriza por ser un lenguaje de programación **estructurado fuertemente tipado**. Lo que implica dos aspectos:

* El código está dividido en porciones fácilmente legibles (funciones o procedimientos).
* El tipo de dato de todas las variables debe ser declarado previamente para que su uso quede habilitado.
  1. Palabras reservadas.
     + **Inglés**
  2. Estructura de datos considerada como tipo elemental.
     + **Conjuntos** con las operaciones para manejo de conjuntos. Debe definirse la constante de tipoconjunto y un término para indicar el conjunto vacío.
  3. Subprogramas.
     + **Funciones.** A diferencia d los procedimientos, las funciones, devuelven un valor.
  4. Estructura de control adicional.
     + **FOR**

Descripción formal

La siguiente especificación corresponde con la descripción formal del lenguaje en BNF.

<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\_subprograma>

<bloque>

| ""

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

<Variables\_locales>

<Marca\_fin\_declar\_variables>

|""

<Cabecera\_programa> ::= "program" <identificador> ";"

| ""

<Marca\_ini\_declar\_variables> ::= "var"

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

<Inicio\_de\_bloque> ::= "begin"

<Fin\_de\_bloque> ::= "end"

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

| <Cuerpo\_declar\_variables>

<Cuerpo\_declar\_variables> ::=<lista\_variables>": set of integer " ";"

| <lista\_variables>": set of real " ";"

| <lista\_variables>": set of character " ";"

| <lista\_variables>": set of boolean " ";"

| <lista\_variables>": integer " ";"

| <lista\_variables>": real " ";"

| <lista\_variables>": character " ";"

| <lista\_variables>": boolean " ";"

<variable> ::= "," <identificador><variable>

|""

<Cabecera\_subprog> ::= "function" <identificador> "(" <lista\_argumentos> ")" ": set of integer " ";"

| "function" <identificador> "(" <lista\_argumentos> ")" ": set of real " ";"

| "function" <identificador> "(" <lista\_argumentos> ")" ": set of character " ";"

| "function" <identificador> "(" <lista\_argumentos> ")" ": set of boolean " ";"

| "function" <identificador> "(" <lista\_argumentos> ")" ": integer " ";"

| "function" <identificador> "(" <lista\_argumentos> ")" ": real " ";"

| "function" <identificador> "(" <lista\_argumentos> ")" ": character " ";"

| "function" <identificador> "(" <lista\_argumentos> ")" ": boolean " ";"

<lista\_argumentos> ::= <lista\_argumentos> ";" <Cuerpo\_declar\_argumentos>

| <Cuerpo\_declar\_argumentos>

<Cuerpo\_declar\_argumentos> ::= <lista\_variables>": set of integer"

| <lista\_variables>": set of real"

| <lista\_variables>": set of character"

| <lista\_variables> ": set of boolean"

| <lista\_variables> ": integer"

| <lista\_variables> ": real"

| <lista\_variables> ": character"

| <lista\_variables> ": boolean"

<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> ::= "if" <expresion> "then" <Sentencia> "else" <Sentencia>

| "if" <expresion> "then" <Sentencia>

| "if" <expresion> "then" <Sentencia><sentencia\_elseif> "else" <Sentencia>

<sentencia\_elseif> ::= "else if" <Sentencia><sentencia\_elseif>

<sentencia\_while> ::= "while" <expresion> "do" <Sentencia>

<sentencia\_for> ::= "for" <identificador>":="<expresion> "to" <expresion> "do" <Sentencias>

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

<lista\_variables> ::= <identificador>

| <identificador> "," <lista\_variables>

<nomb\_entrada> ::= "readln"

| "read"

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

<lista\_expresiones\_o\_cadena> ::= <expresion>

| <lista\_expresiones\_o\_cadena> "," <expresion>

| <frase>

| <lista\_expresiones\_o\_cadena> "," <frase>

<frase> ::= """<identificador>"""

|"""<digito><identificador>"""

<nomb\_salida> ::= "writeln"

|"write"

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

<lista\_expresiones> ::= <expresion>

| <lista\_expresiones> "," <expresion>

<sentencia\_return> ::= <sentencia\_asignacion>

<expresion> ::= "("<expresion>")"

| "+"<expresion>

| "-"<expresion>

| <expresion>"+"<expresion>

| <expresion>"-"<expresion>

| <expresion>"\*"<expresion>

| <expresion>"/"<expresion>

| <expresion> ">" <expresion>

| <expresion> ">=" <expresión

| <expresion> "<" <expresion>

| <expresion> "<=" <expresion>

| <expresion> "=" <expresion>

| <expresion> "¬=" <expresion>

| <expresion> "<>" <expresion>

| "not"<expresion>

| <expresion>"and"<expresion>

| <expresion>"or"<expresion>

| <expresion>"xor"<expresion>

| <expresion>"in"<expresion>

| <identificador>

| <constante>

| <conjunto>

| <funcion>

<identificador> ::= <letra><restoIdentificador>

| "\_"<restoIdentificador>

<restoIdentificador> ::= <restoIdentificador><digito>

| <restoIdentificador><letra>

| <digito>

| <letra>

| ""

<conjunto> ::= "["<restoConjunto>"]"

<restoConjunto> ::= <expresion> "," <restoConjunto>

| <constante>".."<constante>

|<expresion>

| ""

<constante> ::= <constanteReal>

| <constateEntera>

| <caracter>

| <booleano>

<constanteReal> ::= <constanteEntera>"."<constanteEntera>

<constanteEntera> ::= <constanteEntera><digito>

| <digito>

<caracter> ::= "'"<letra>"'"

<digito> ::= "0"|"1"|"2"|"3"|"4"|"5"|"6"|"7"|"8"|"9"

<letra> ::= "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"

|"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"

<booleano> ::= "true"

| "false"

Semántica en lenguaje natural

En este apartado es explicará cómo funciona todas y cada una de las instrucciones del lenguaje que estamos creando. Se irán describiendo de más generales a más concretas.

Programa

Primeramente, todo programa debe definirse. Por tanto debe constar de una cabecera (donde se le da nombre al programa) y un bloque de instrucciones (que hace el programa).

El bloque de instrucciones tiene delimitadores, para saber dónde empieza y dónde acaba, y tres partes principales:

* La declaración de variables.
* La declaración de subprogramas (funciones en nuestro lenguaje).
* Las sentencias del programa.

Declaración de variables

En la declaración de variables tenemos dos partes:

* Delimitadores.
* Variables (nombre y tipo). Para declarar las variables se declaran primero los tipos y luego se añaden los identificadores de las variables para cada tipo.

Declaración de subprogramas (funciones)

En esta parte se declaran los subprogramas o funciones que queremos utilizar en nuestro programa. Tiene dos partes:

* Cabecera. Se describe el nombre de la función y su valor de retorno.
* Bloque. En este bloque es donde colocaremos la declaración de variables (se declaran como se ha visto en el apartado anterior) locales y nuestras instrucciones a ejecutar, ya sean estructuras de control o operaciones (que explicaremos en el apartado siguiente)..

Sentencias del programa

Este apartado es el más extenso de todos por la variedad de opciones que tenemos. Se pueden poner todas las sentencias que queramos, separadas todas ellas por punto y coma (;). Las sentencias son de varios tipos y se clasifican en:

* Sentencias de control del flujo del programa.
* Sentencias de asignación.
* Sentencias de I/O.
* Sentencias de retorno

En las estructuras de control del flujo siempre se evalúa una expresión para decidir qué hacer. Estas expresiones están detalladas en la introducción de esta documentación y dependiendo de los diferentes tipos de variables se pueden hacer unas u otras.

Los tipos se han definido como:

* Carácter. Letras (mayúsculas y minúsculas) e números.
* Booleano. True o false.
* Entero. Número entero.
* Real- Número real, con parte entera y parte real separadas por un punto.
* Conjunto sobre cualquiera de los tipos anteriores. Variables de tipos anteriores separadas por comas y englobadas por corchetes.

La asignación se realiza sobre identificadores (variables).Los identificadores son formaciones de caracteres (letras o \_) y se asigna una expresión.

Las sentencias de I/O son para leer y escribir por los periféricos predeterminados. Leer de teclado y escribir en la pantalla. Aquí aparece una nueva producción la “frase”. La frase es una cadena de caracteres (un string).

Por último, la sentencia de retorno se una para indicar que expresión va a devolver nuestra función.

Identificación de los tokens

Según nuestra gramática hemos podido identificar los siguientes tokens:

* TIPO. Tipos de variables/conjuntos
* OPBI. Operadores binarios
* OPUN. Operadores unarios
* CONS. Constantes carácter, booleano y real
* CONSEN. Constante entera
* IDEN. Identificador
* NOT. Operador negación
* CO. La coma
* PAA.Paréntesis abierto
* PAC. Paréntesis cerrado
* SALIDA. Instrucción de salida
* FRASE. Cadena de caracteres
* COSIM. Comilla simple
* CODOB. Comilla doble
* ENTRADA. Instrucción de entrada
* FOR. Estructura de control para
* IGUAL. Operador de asignación
* TO. Inicio del bloque para
* DO. Final del bloque para
* WHILE. Estructura de control mientras
* IF. Estructura de control si
* THEN. Inicio del bloque si
* ELSE. Inicio del bloque sino
* ELIF. Final del bloque si
* PUNCO. Punto y coma
* FUNCION. Denotación de declaración de función
* BEGIN. Inicio de bloque
* END. Final de bloque
* VAR. Variables
* PROGRAM. Denotación de declaración de programa
* PUN. Punto
* PUN2. Dos puntos
* CORA. Corchete abierto
* CORC. Corchete cerrado

Tabla de tokens

|  |  |  |  |
| --- | --- | --- | --- |
| Token | Expresión Regular | Código del Token | Atributos |
| TIPO | (“set of integer”  | ”set of real”  | “set of character”  | “set of boolean”  | “integer”  | “real”  | “character”  | “boolean”) | 257 | {set of integer,set of real,  set of character,set of boolean  integer,real,character,boolean} |
| **CONS** | (“<carácter>”  | “<booleano>”  | “<constanteReal>”) | 258 | {<carácter>, <booleano>, <constanteReal>} |
| **CONSEN** | “<constanteEntera>” | 259 | {<constanteEntera>} |
| **IDEN** | “<identificador>” | 260 | {<identificador>} |
| CO | “,” | 261 | {,} |
| PAA | “ (“ | 262 | {(} |
| PAC | “)” | 263 | {)} |
| **SALIDA** | “<nomb\_salida>” | 264 | {<nomb\_salida>} |
| **FRASE** | “<frase>” | 265 | {<frase>} |
|  |  |  |  |
|  |  |  |  |
| **ENTRADA** | “<nomb\_entrada>” | 268 | {<nomb\_entrada>} |
| FOR | “for” | 269 | { for } |
| IGUAL | “:=” | 270 | {:=} |
| TO | “to” | 271 | { to } |
| DO | “do” | 272 | { do } |
| WHILE | “while” | 273 | { while } |
| IF | “if” | 274 | { if } |
| THEN | “then” | 275 | { then } |
| ELSE | “else” | 276 | { else } |
|  |  |  |  |
| PUNCO | “;” | 278 | {;} |
| FUNCION | “function” | 279 | { function } |
| BEGIN | “begin” | 280 | { begin } |
| END | “end” | 281 | { end } |
| VAR | “var” | 282 | {var} |
| PROGRAM | “program” | 283 | {program} |
| PUN | “.” | 284 | {.} |
| PUN2 | “..” | 285 | {..} |
| CORA | “[“ | 286 | {[} |
| CORC | “]” | 287 | {]} |
| DOSPUN | “:” | 288 | {:} |
| OPREL | (“>”  | “>=”  | “<”  | “<=”  | “=”  | “¬=”  | “<>“) | 289 | {>, >=, <, <=, =, ¬=, <>} |
| SUMRES | (“+”  | “-“) | 290 | {+, -} |
| MULDIV | (“\*”  | “/”) | 291 | {\*, /} |
| NOT | “not” | 292 | {not} |
| IN | “in“ | 293 | {In} |
| OR | “or” | 294 | {Or} |
| XOR | “xor” | 295 | {Xor} |
| AND | “and” | 296 | {And} |

Especificación Lex

%{

#define MAXLONBUFFER 200

char buffer[MAXLONBUFFER];

%}

retorno \n

blanco " "|\t

letra [a-zA-Z]

digito [0-9]

boolean ("true"|"false")

otros .

%option noyywrap

%%

{retorno} {nlineas ++;}

{blanco}+ ;

":" return DOSPUN;

"set of integer" return TIPO;

"set of real" return TIPO;

"set of char" return TIPO;

"set of boolean" return TIPO;

"integer" return TIPO;

"real" return TIPO;

"char" return TIPO;

"boolean" return TIPO;

">" return OPREL;

">=" return OPREL;

"<" return OPREL;

"<=" return OPREL;

"=" return OPREL;

"¬=" return OPREL;

"<>" return OPREL;

"and" return AND;

"or" return OR;

"xor" return XOR;

"in" return IN;

"\*" return MULDIV;

"/" return MULDIV;

"+" return SUMRES;

"-" return SUMRES;

"not" return NOT;

"," return CO;

"(" return PAA;

")" return PAC;

"for" return FOR;

":=" return IGUAL;

"to" return TO;

"do" return DO;

"while" return WHILE;

"if" return IF;

"then" return THEN;

"else" return ELSE;

\; {return (PUNCO);}

"function" {return (FUNCION);}

"begin" return BEGINN;

"end" {return (END);}

"var" return VAR;

"program" return PROGRAM;

".." return PUN2;

"." return PUN;

"[" return CORA;

"]" return CORC;

("writeln"|"write") return SALIDA;

("readln"|"read") return ENTRADA;

("'"[^"'"]"'"|{boolean}|{digito}+"."{digito}+) return CONS;

{digito}+ return CONSEN;

({letra}|"\_")({letra}|{digito})\* return IDEN;

\'[^\']\*\' return FRASE;

{otros} {snprintf(buffer,MAXLONBUFFER, "Los caracteres '%s' no forman ningun token conocido",yytext);

yyerror(buffer);}

%%

Especificación YACC

%{

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#define YYERROR\_VERBOSE

/\*\* La siguiente declaracion permite que yyerror se pueda invocar desde el

\*\*\* fuente de lex (prueba.l)

\*\*/

void yyerror( char\* msg ) ;

/\*\* La siguiente variable se usará para conocer el numero de la línea

\*\*\* que se esta leyendo en cada momento. También es posible usar la variable

\*\*\* yylineno que también nos muestra la línea actual. Para ello es necesario

\*\*\* invocar a flex con la opción -l (compatibilIDENTIFICADORad con lex).

\*\*/

int nlineas = 1 ;

%}

/\*\* Para uso de mensajes de error sintáctico con BISON.

\*\*\* La siguiente declaración provoca que bison, ante un error sintáctico,

\*\*\* visualice mensajes de error con indicación de los tokens que se esperaban

\*\*\* en el lugar en el que produjo el error (SÓLO FUNCIONA CON BISON>=2.1).

\*\*\* Para Bison<2.1 es mediante

\*\*\*

\*\*\* #define YYERROR\_VERBOSE

\*\*\*

%error-verbose

/\*\* A continuación declaramos los nombres simbólicos de los tokens.

\*\*\* byacc se encarga de asociar a cada uno un código

\*\*/

%token TIPO

%token CONS

%token CONSEN

%token IDEN

%token CO

%token PAA

%token PAC

%token SALIDA

%token FRASE

%token ENTRADA

%token FOR

%token IGUAL

%token TO

%token DO

%token WHILE

%token IF

%token THEN

%token ELSE

%token PUNCO

%token FUNCION

%token BEGINN

%token END

%token VAR

%token PUN

%token PUN2

%token CORA

%token CORC

%token DOSPUN

%token PROGRAM

%left OR

%left AND

%left XOR

%left NOT

%left IN

%left OPREL

%left SUMRES

%left MULDIV

%start Programa

%%

Programa: Cabecera\_programa bloque PUN;

Cabecera\_programa: PROGRAM IDEN PUNCO

| error;

bloque: Declar\_de\_variables\_locales Declar\_de\_subprogs BEGINN Sentencias END;

Declar\_de\_variables\_locales: VAR Variables\_locales

| ;

Variables\_locales: Variables\_locales Cuerpo\_declar\_variables

| Cuerpo\_declar\_variables;

Cuerpo\_declar\_variables: lista\_variables DOSPUN TIPO PUNCO;

lista\_variables: IDEN

| lista\_variables CO IDEN

| error;

Declar\_de\_subprogs: Declar\_de\_subprogs Declar\_subprog

| ;

Declar\_subprog: Cabecera\_subprog bloque PUNCO;

Cabecera\_subprog: FUNCION IDEN PAA lista\_argumentos PAC DOSPUN TIPO PUNCO;

lista\_argumentos: lista\_argumentos PUNCO Cuerpo\_declar\_argumentos

| Cuerpo\_declar\_argumentos;

Cuerpo\_declar\_argumentos: lista\_variables DOSPUN TIPO;

Sentencias: Sentencias Sentencia

| ;

Sentencia: bloque PUNCO

| sentencia\_asignacion

| sentencia\_if

| sentencia\_while

| sentencia\_for

| sentencia\_entrada

| sentencia\_salida

| error;

sentencia\_asignacion: IDEN IGUAL expresion PUNCO;

sentencia\_if:

IF expresion THEN Sentencia

| IF expresion THEN Sentencia ELSE Sentencia;

sentencia\_while: WHILE expresion DO Sentencia;

sentencia\_for: FOR IDEN IGUAL expresion TO expresion DO Sentencia;

sentencia\_entrada: ENTRADA PAA lista\_variables PAC PUNCO;

sentencia\_salida: SALIDA PAA lista\_expresiones\_o\_cadena PAC PUNCO;

lista\_expresiones\_o\_cadena: expresion

| lista\_expresiones\_o\_cadena CO lista\_expresiones\_o\_cadena\_cont

| FRASE;

lista\_expresiones\_o\_cadena\_cont: expresion

| FRASE;

lista\_expresiones: expresion

| lista\_expresiones CO expresion

| ;

expresion: PAA expresion PAC

| SUMRES expresion

| expresion SUMRES expresion

| expresion OPREL expresion

| expresion MULDIV expresion

| expresion OR expresion

| expresion XOR expresion

| expresion IN expresion

| expresion AND expresion

| NOT expresion

| IDEN

| constante

| conjunto

| funcion

| error;

funcion: IDEN PAA lista\_expresiones PAC;

conjunto: CORA restoConjunto CORC;

restoConjunto: restoConjunto CO expresion

| constante PUN2 constante

| expresion

| ;

constante: CONS

| CONSEN;

%%

/\*\* aqui incluimos el fichero generado por el 'lex'

\*\*\* que implementa la función 'yylex'

\*\*/

#ifdef DOSWINDOWS /\* Variable de entorno que indica la plataforma \*/

#include "lexyy.c"

#else

#include "lex.yy.c"

#endif

/\*\* se debe implementar la función yyerror. En este caso

\*\*\* simplemente escribimos el mensaje de error en pantalla

\*\*/

void yyerror( char \*msg )

{

fprintf(stderr,"[Linea %d]: %s\n", nlineas, msg);

}