Diseño de Compiladores

Grupo 1

Profesores

Elda Guadalupe Quiroga González

Héctor Gibrán Ceballos Cancino

Avance 0

Tokens, Diagramas de Sintaxis y Gramática

A01234029 Jazmín Yolistli Santibáñez de la Rosa

8 de octubre de 2022

Lista de Tokens

- Palabras reservadas:

  ## <PROGRAMA>
  'main' : 'MAIN',

  ## <DEC_VAR>
  'var' : 'VAR',

  ## <TIPO_S>
  'int' : 'INT',
  'float' : 'FLOAT',
  'char' : 'CHAR',

  ## <TIPO_C>
  'dataframe' : 'DATAFRAME'
  'file' : 'FILE'

  ## <FUNCS>
  'func' : 'FUNC',
  'void' : 'VOID',
  'return' : 'RETURN',

  ## <ESTATUTOS>
  'read' : 'READ',
  'print' : 'PRINT',

  'if' : 'IF',
  'else' : 'ELSE',

  'while' : 'WHILE',

  'for' : 'FOR',
  'to' : 'TO',
  'step' : 'STEP'

- Definiciones simples de tokens:

  OP_ASSIGN : '='

  ## Airthmetic operators
  OP_ADD: '+'
  OP_SUBTR : '-'
  OP_MULT : '*'
  OP_DIV : '/'

```
## Logical operators
OP_AND = '\&'
OP_OR =


## Relational operators
OP_EQ : '=='
OP_NEQ : '!='
OP_LT : '<'
OP_GT : '>'
OP_LTE : '<='
OP_GTE : '>='

## Separators
lPAREN : '('
rPAREN : ')'
lBRACE : '{'
rBRACE : '}'
lBRACKET : '['
rBRACKET : ']'

SEP_SEMICOLON : ';'
SEP_COMMA : ','

ignore : '\t'

CTE_F : '[0-9]+(\.[0-9]+)?'
CTE_I : '[0-9]+'
CTE_CHAR : ' ( \'[a-zA-Z0-9]\' ) '
LETRERO : ' ( \" [^(\"|\')]* \" ) '
COMENTARIO : '\#.*'
```

Gramática Formal

programa → aux_prog aux_prog2 MAIN lPAREN rPAREN cuerpo;
aux_prog → dec_var | ε ;
aux_prog2 → dec_func aux_prog2 | ε ;

cuerpo → aux_cuerpo bloque
aux_cuerpo → dec_var | ε ;

dec_var → VAR aux_dv;
aux_dv → aux_dv2 aux_dv3 ;
aux_dv2 → tipo_s | tipo_c ;
aux_dv3 → ID aux_dv4 aux_dv6 SEP_SEMICOLON aux_dv7
aux_dv4 → arr aux_dv5 | ε ;
aux_dv5 → arr | ε ;
auxdv6 → SEP_COMMA aux_dv3 | ε ;
aux_dv7 → aux_dv | ε ;

arr → lBRACKET aux_arr rBRACKET ;
aux_arr → ID | CTE_I ;

call_var → ID aux_cv
aux_cv → arr aux_cv2 | ε ;
aux_cv2 → arr | ε ;

tipo_s → INT | FLOAT | CHAR

tipo_c → DATAFRAME | FILE

params → tipo_s call_var aux_params
aux_params → SEP_COMMA params | ε ;

dec_func → func_void | func_return

func_void → FUNC VOID ID lPAREN aux_fv rPAREN cuerpo
aux_fv → params | ε ;

func_return → FUNC tipo_s ID lPAREN aux_fr rPAREN
lBRACE aux_fr2 bloque_return rBRACE
aux_fr → params | ε ;
aux_fr2 → dec_var | ε ;

return → RETURN lPAREN h_exp rPAREN SEP_SEMICOLON

call_func → ID lPAREN aux_cf rPAREN
aux_cf → h_exp aux_cf2 | ε ;
aux_cf2 → SEP_COMMA aux_cf | ε ;

bloque → lBRACE estatuto SEP_SEMICOLON aux_bloque aux_bloque2 rBRACE
aux_bloque → comentario | ε ;
aux_bloque2 → estatuto SEP_SEMICOLON aux_bloque aux_bloque2 | ε ;

bloque_return → estatuto SEP_SEMICOLON aux_bloque aux_bloqu2 return

estatuto → asignacion | call_func | leer | escribir | condicion | ciclo_while | ciclo_for |
COMENTARIO

asignacion → call_var OP_ASSIGN h_exp

leer → READ lPAREN call_var aux_leer rPAREN
aux_leer → SEP_COMMA call_var aux_leer | ε ;

escribir → PRINT lPAREN aux_escribir aux_escribir2 rPAREN
aux_escribir → h_exp | LETRERO | CTE_CHAR
aux_escribir2 → SEP_COMMA aux_escribir aux_escribir2 | ε ;

condicion → IF lPAREN h_exp rPAREN bloque aux_condicion
aux_condicion → ELSE bloque | ε ;

ciclo_while → WHILE lPAREN h_exp rPAREN bloque

ciclo_for → FOR lPAREN ID OP_ASSIGN h_exp TO h_exp rPAREN aux_ciclofor bloque
aux_ciclo → STEP h_exp | ε ;

h_exp → s_exp aux_hexp
aux_hexp → OP_AND h_exp | OP_OR h_exp | ε ;

s_exp → exp aux_sexp
aux_sexp → aux_sexp2 exp | ε ;
aux_sexp2 → OP_EQ | OP_NEQ | OP_LT | OP_GT | OP_LTE | OP_GTE

exp → termino aux_exp
aux_exp → OP_ADD exp | OP_SUBTR exp | ε ;

termino → factor aux_termino

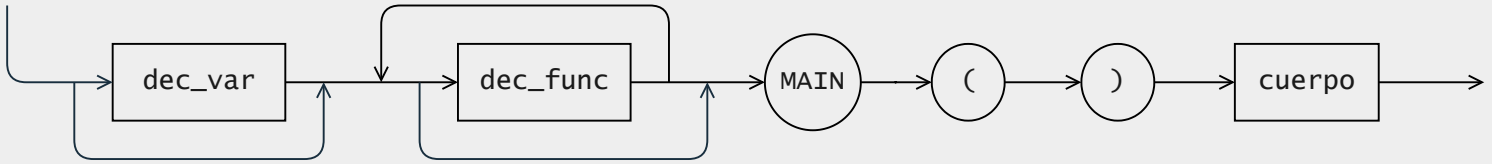
aux_termino → OP_MULT termino | OP_DIV termino | ε ;

factor → aux_factor aux_factor2
aux_factor → OP_ADD | OPP_SUBTR | ε ;
aux_factor2 → CTE_I | CTE_F | h_exp | call_func | call_var

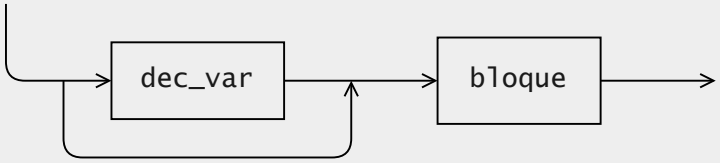
Principales consideraciones semánticas

- Hay variables globales y locales (dentro del main y dentro de cada función).
- Las variables no se inicializan en su declaración.
- Los arreglos son de 1 o 2 dimensiones (arreglos y matrices).
- Las dimensiones de los arreglos sólo pueden ser enteros (o variables de tipo entero).
- Solo puede haber arreglos de enteros y flotantes de tipo simple (tipo_s).
- Las funciones unicamente regresan valores de tipo simple y sus parámetros son de tipo simple.
- Las funciones void no regresan valores.
- Las funciones void no pueden ser llamadas en asignaciones ni expresiones.
- Orden de prescedencia de operadores (de mayor a menor)
    1. *, /
    2. +, -
    3. Operadores relacionales: <, >, ==, !=
    4. Operadores lógicos: & and |
    5. Asignación =

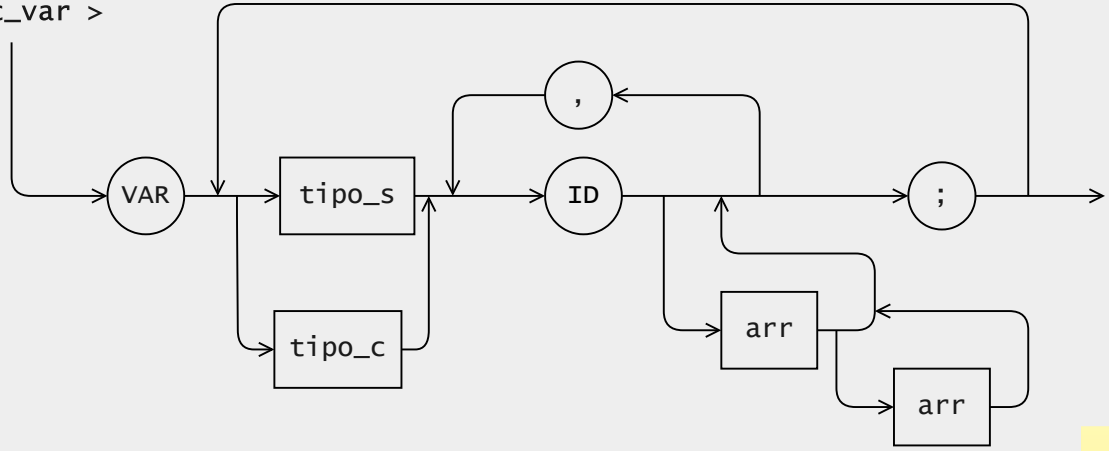
< programa >



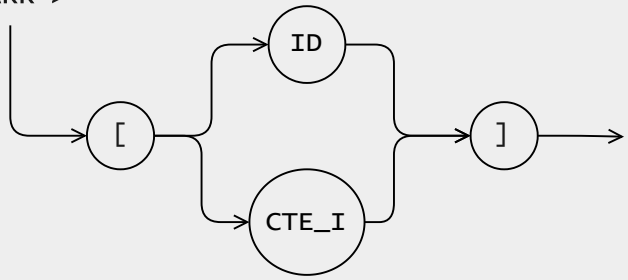
< cuerpo >



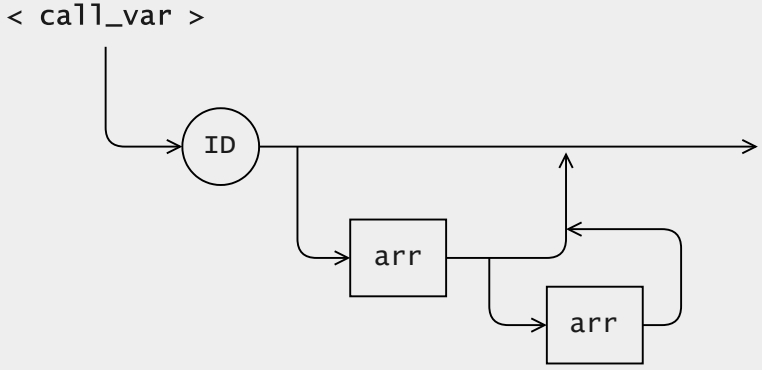
< dec_var >

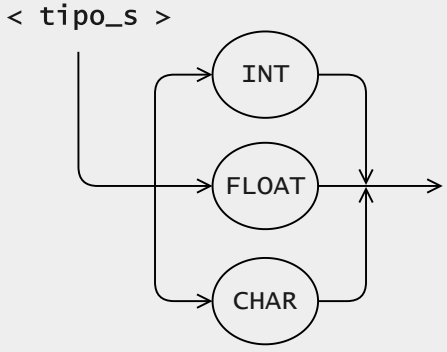


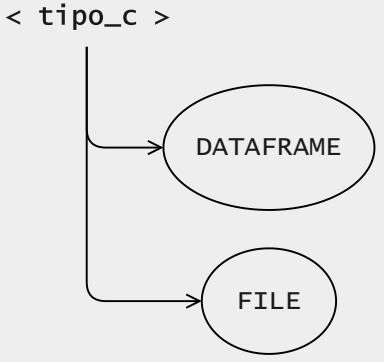
< ARR >

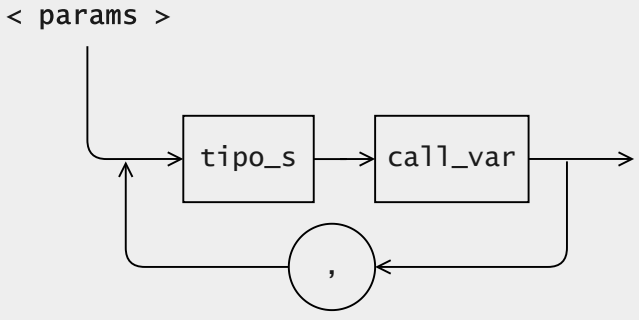


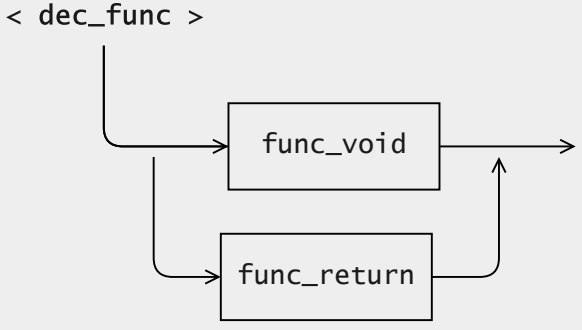
¿Cómo manejar declaración de variables con inicialización?


< call_var >
ID
arr
arr

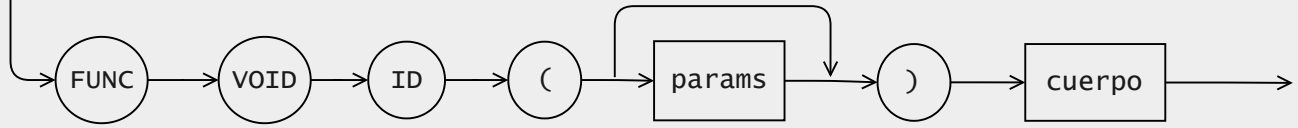

< tipo_s >
INT
FLOAT
CHAR


< tipo_c >
DATAFRAME
FILE

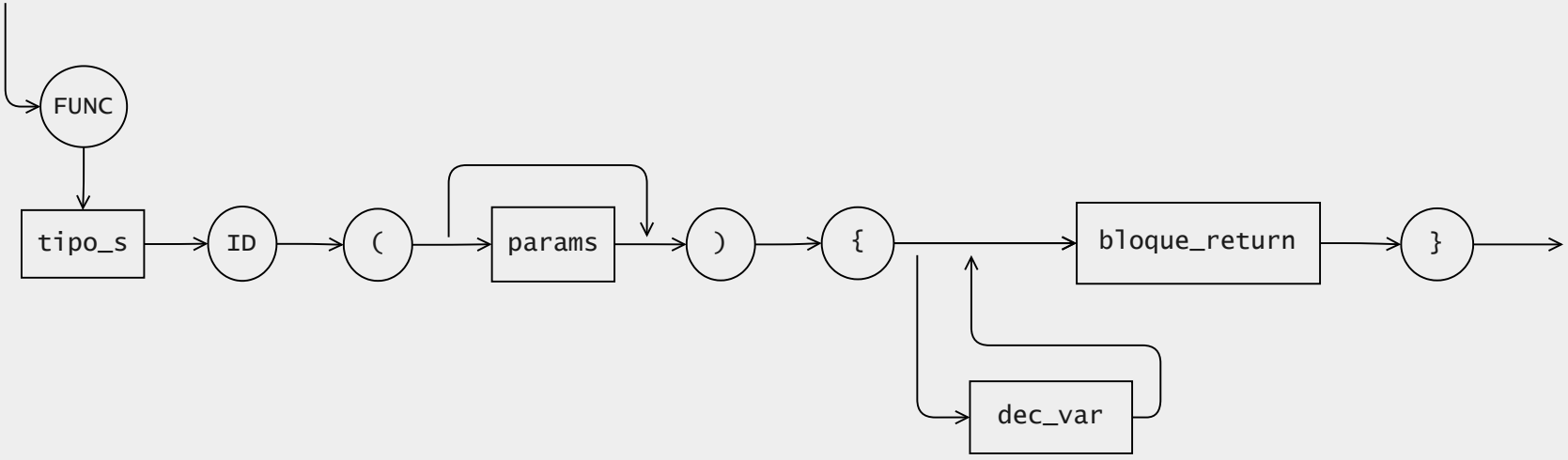

< params >
tipo_s
call_var
,


< dec_func >
func_void
func_return

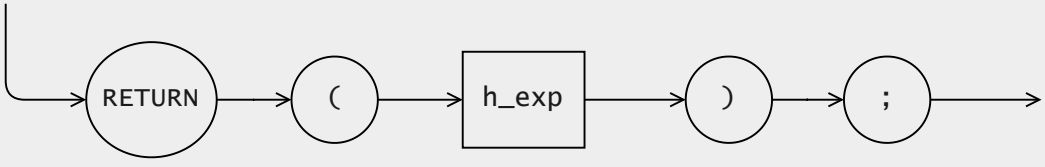
< func_void >

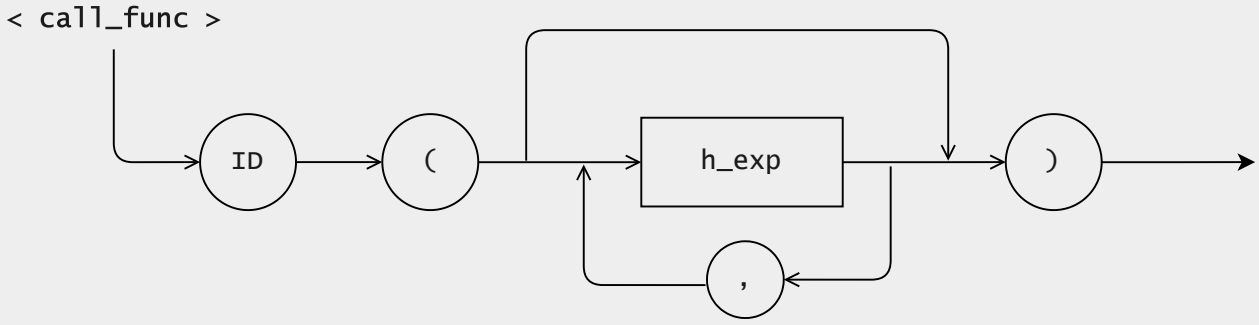


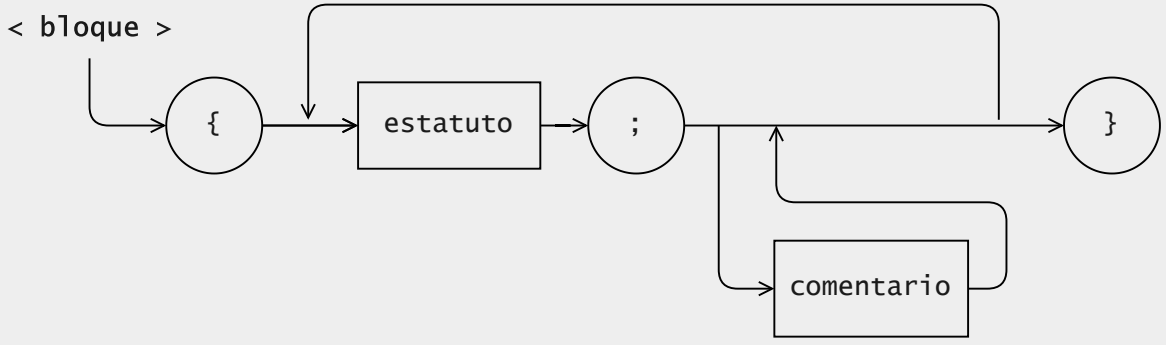
< func_return >

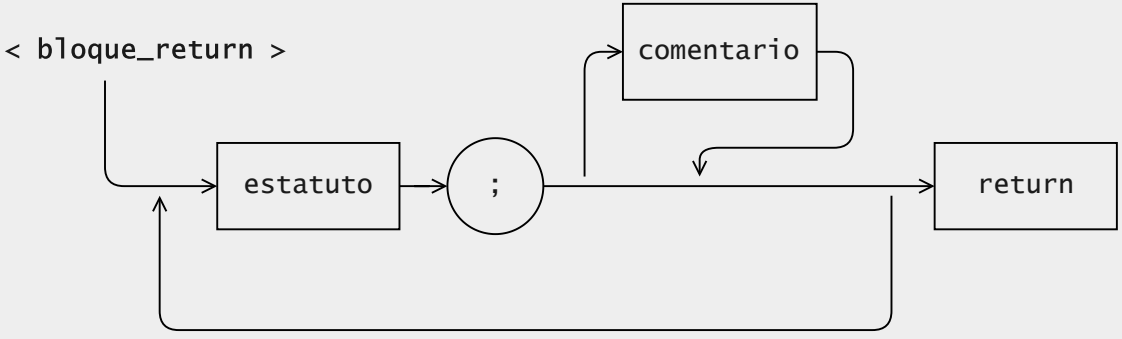


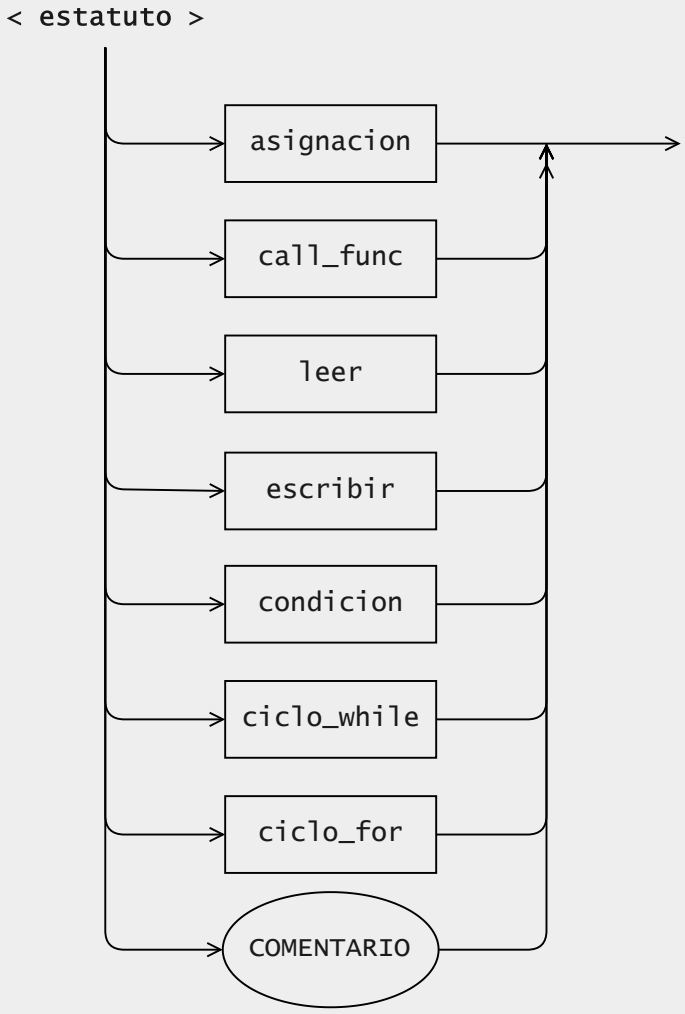
< return >

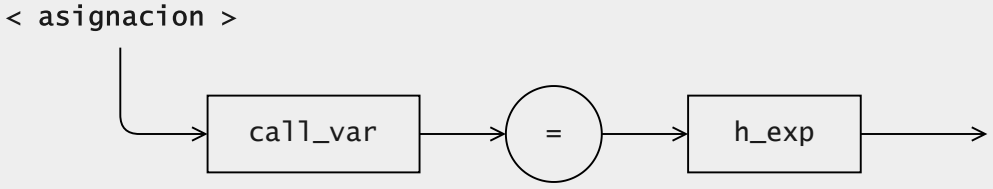


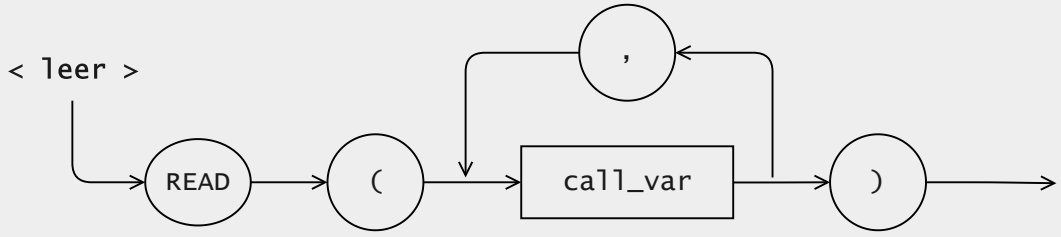

< call_func >
ID
(
h_exp
)
,

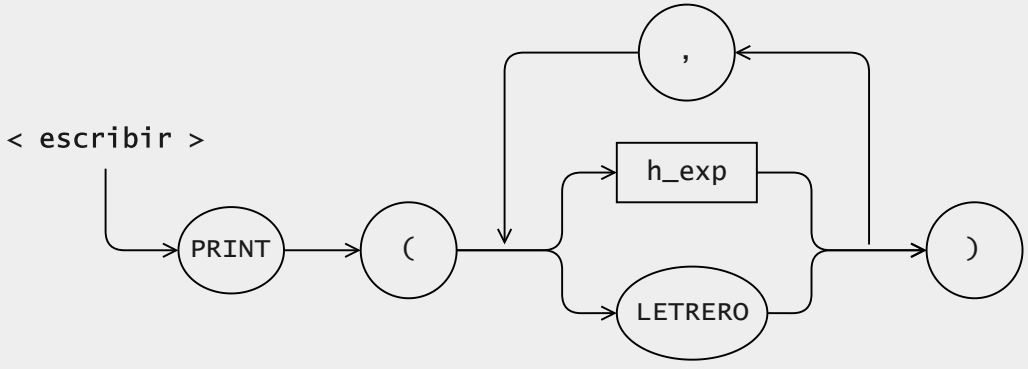

< bloque >
{
estatuto
;
}
comentario

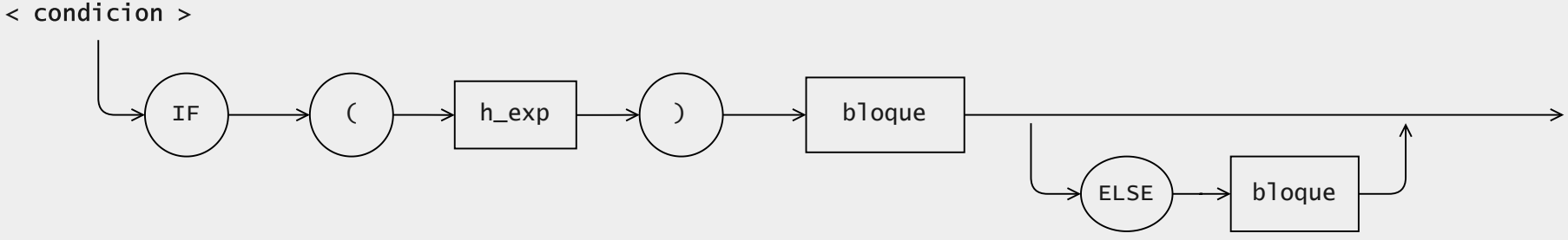

< bloque_return >
comentario
estatuto
;
return


< estatuto >
asignacion
call_func
leer
escribir
condicion
ciclo_while
ciclo_for
COMENTARIO

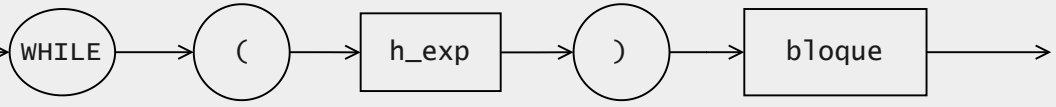

< asignacion >
call_var
=
h_exp


< leer >
READ
(
,
call_var
)

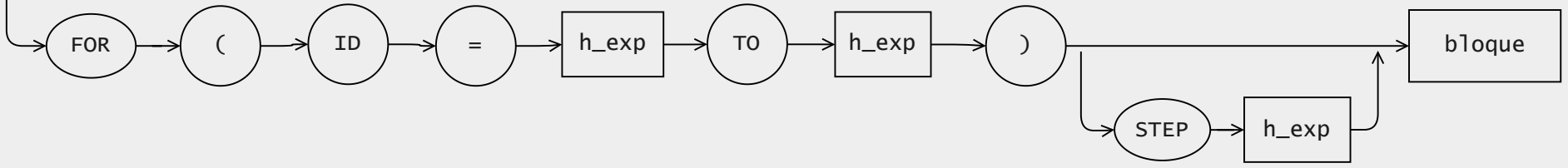

< escribir >
PRINT
(
,
h_exp
LETRERO
)

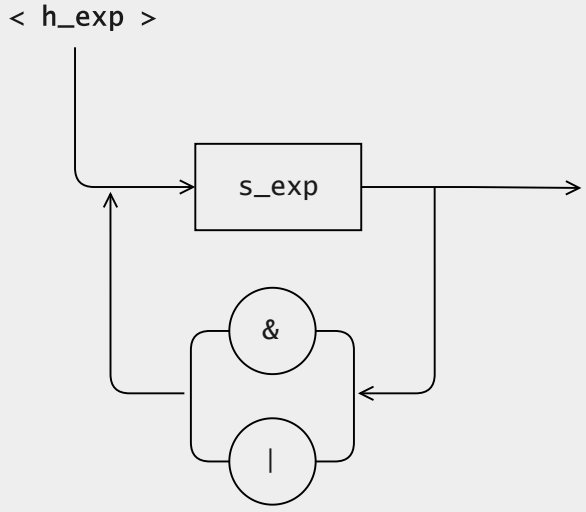

< condicion >
IF
(
h_exp
)
bloque
ELSE
bloque

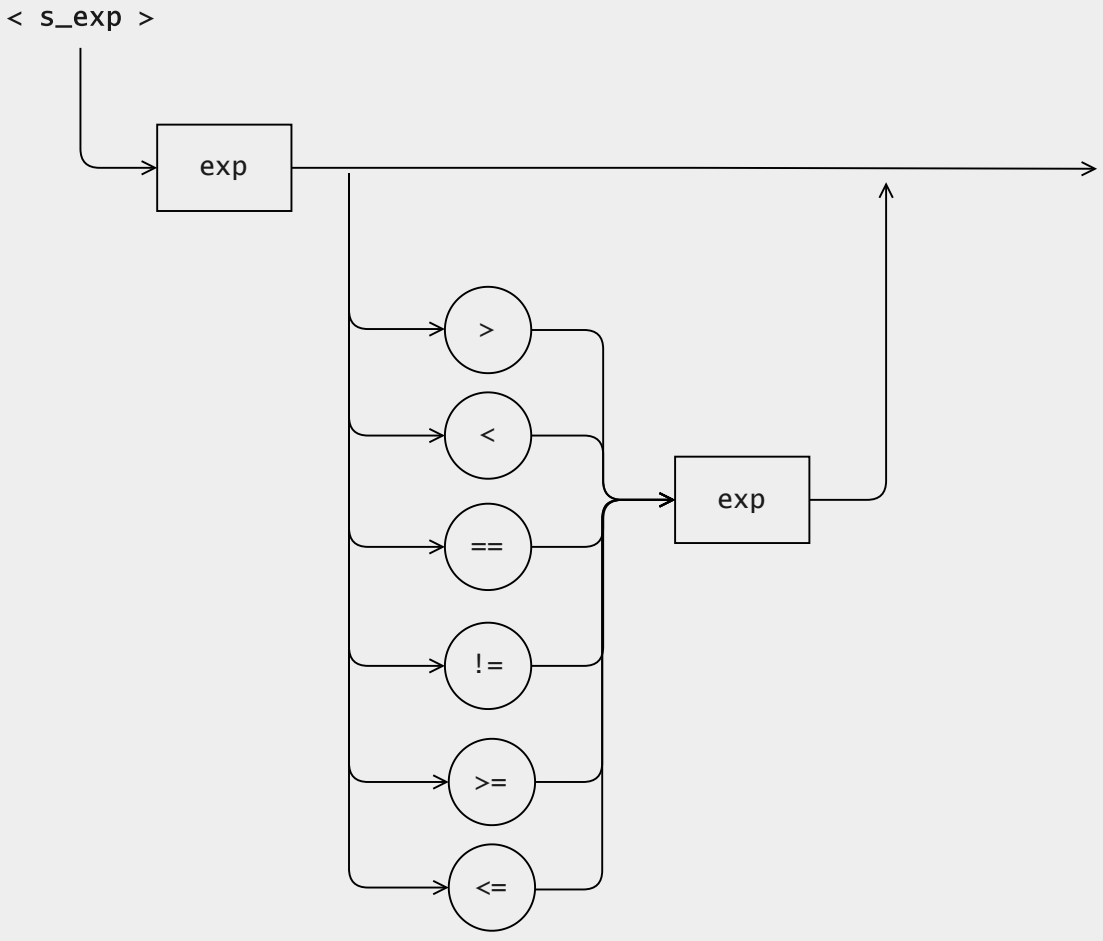
< ciclo_while >

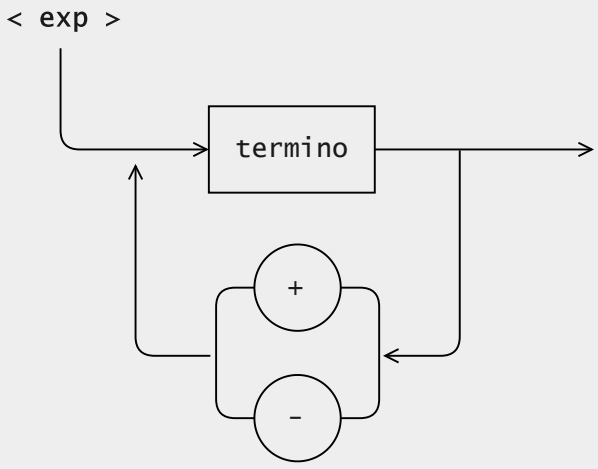


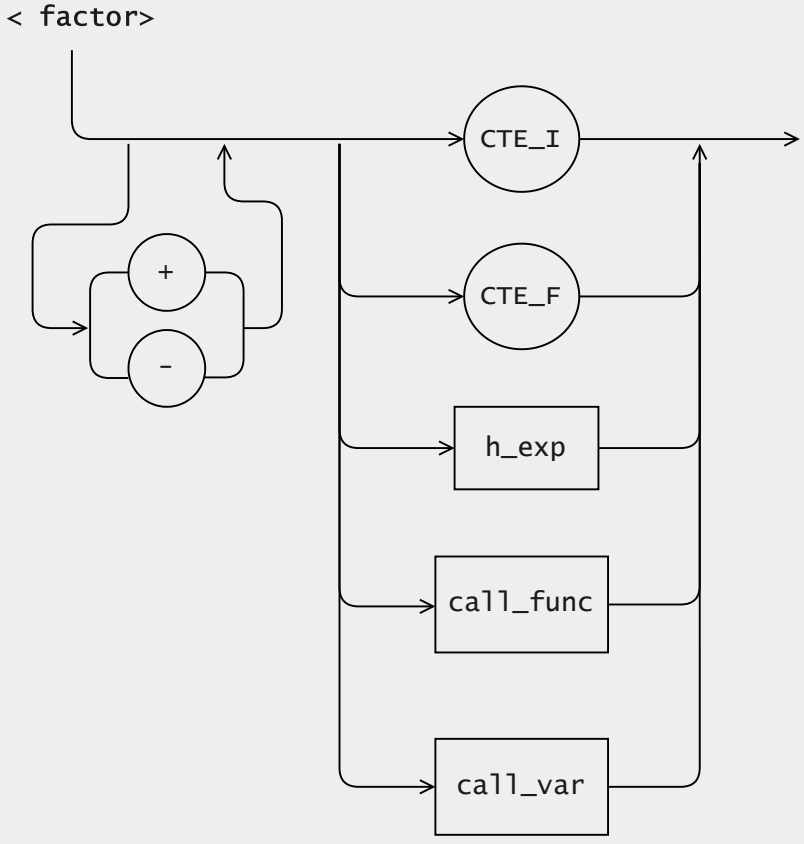
< ciclo_for >

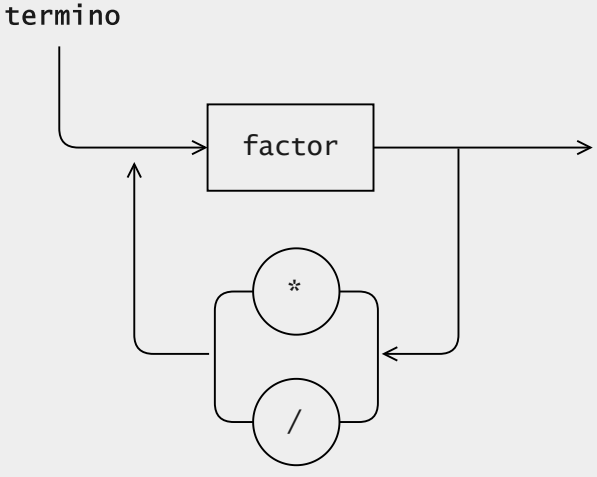



< h_exp >
s_exp
&
|


< s_exp >
exp
>
<
==
!=
>=
<=
exp


< exp >
termino
+
-


< factor>
CTE_I
+
-
CTE_F
h_exp
call_func
call_var


termino
factor
*
/