Tecnológico de Monterrey

Propuesta de Compiladores

Patito++

La Propuesta propuesta

29/03/2020

Oscar Lerma A01380817

Cesar Buenfil A01207499

Objetivo del Lenguaje
Cumplir con las expectativas y funciones descritas en el proyecto en parejas: *Lenguaje Patito++*

Requerimientos de Lenguaje

I) Elementos Básicos (tokens, palabras reservadas)
Palabras reservadas

PROGRAMA →'Programa'
PRINCIPAL →'principal'
FLOAT → 'float'
FUNCION →'funcion'
ENTONCES → 'entonces'
SINO → 'sino'
MIENTRAS → 'mientras'
HAZ → 'haz'
HASTA → 'hasta'
STRING →'string'
VOID→'void'
VAR →'var'
INT →'int'
CHAR →'char'
Sí → 'si'
LEE →'lee'
REGRESA → 'regresa'
ESCRIBE →'escribe'
DESDE → 'desde'
HACER → 'hacer'
NULL → 'null'

Tokens

COMMENT → '%%.*\n'
L_PAREN → '\('
R_PAREN → '\)'
L_BRACKET → '{'
R_BRACKET → '}'
AND → '&&'
OR → '||'
TRANS_ARR →'¡'
COMPARE → '=='
CTE_I → '(\+|-)?[0-9]+'
CTE_F → '(\+|-)?[0-9]+(\.[0-9]+)?f
CTE_STRING → '".*"'
CTE_CH → '\'[A-Za-z]\''
DOT → '\.'
PLUS → '\+'
MINUS → '-'
INV_ARR → '?'
DOTS → ':'

DIV → ‘/’	ID → ‘[A-Za-z]([A-Za-z]|[0-9]_)*’
MULT → ‘*’	EQUAL → ‘=’
MORE → ‘>’	DIFFERENT → ‘!=’
LESS → ‘<’	LESSEQUAL → ‘<=’
NOT → ‘!’	MOD → ‘%’
MOREEQUAL → ‘>=’	DET_ARR →’$’
DOTCOMA →’;’	COMA → ‘,’
LSTAPLE →’[’	RSTAPLE →’]’

II) Descripción de funciones

- escribe(**cte_string** , **EXPRESION, id,** ...) : Retorna a pantalla (standard output) el parámetro, (en caso de expresión, todo valor de retorno). En caso de múltiples parámetros, se concatenan de manera inmediata en pantalla (standard output)
- lee(**id**, ...) : La función espera input del usuario (standard input) y intenta mapear el mismo en las variable especificada como parámetro. Si existen múltiples parámetros, la función espera inputs del usuario para cada parámetro separados por ‘\n’ (linea nueva).

III) Tipos de Datos

- Int
- Float
- Char
- String
- Arreglos
- Matrices

IV) Sistema operativo y lenguaje.

- Linux (basado en debian)
- Python 3, analizador léxico y sintáctico PLY (3.1.1)

VI) Gramáticas libres de contexto.

PROGRAM → programa id ; VARS FUNCTIONS MAIN
MAIN →principal ( ) VARS BLOQUE
VARS → var VAR_AUX | ϵ

VAR_AUX →TIPO IDS VAR_AUX | ϵ
TIPO → int | float | char | string
IDS → id ARRDIM ; | id ARRDIM , IDS
ARRDIM → [EXPRESION] | [EXPRESION] [EXPRESION] | [EXPRESION, EXPRESION] | ϵ
FUNCTIONS → FUNCTION FUNCTIONS | ϵ
FUNCTION → funcion TIPO id (PARAM) VARS BLOQUE | funcion void id (PARAM) VARS BLOQUE
PARAM → TIPO id PARENTESIS PARAM_AUX
PARA_AUX → , PARAM | ϵ
PARENTESIS → [ ] | [ ] [ ] | ϵ
BLOQUE → { ESTATUTOS }
ESTATUTOS → ESTATUTO ESTATUTOS | ϵ
ESTATUTO → ASIGNACION ; | FUN ; | COND | WRITE ; | READ ; | RETURN ;
ASIGNACION → id ARRDIM = EXPRESION | id ARRDIM = CTE_ARR
EXPRESION → SUBEXP && SUBEXP | SUBEXP || SUBEXP | SUBEXP
SUBEXP → EXP | EXP COMPARACION EXP
COMPARACION → > | < | == | != | >= | <=
EXP → TERMINO | TERMINO + EXP | TERMINO - EXP
TERMINO → FACTOR | FACTOR * TERMINO | FACTOR / TERMINO | FACTOR % TERMINO
FACTOR → (EXPRESION) | + CTE | - CTE | !CTE | CTE ARROP | CTE
CTE → cte_i | cte_f | ct_ch | cte_string | FUN | id ARRDIM
ARROP→ $ | i | ?
FUN → id (FUN_AUX)
FUN_AUX → EXPRESION , FUN_AUX | EXPRESION | EMPTY
COND → IF | FOR | WHILE

IF → si (EXPRESION) entonces BLOQUE SI_AUX | si (EXPRESION) entonces COND
IF_AUX → sino BLOQUE | ϵ
WHILE → mientras (EXPRESION) WHILE_AUX BLOQUE | mientras (EXPRESION) WHILE_AUX COND
WHILE→ haz | empty
FOR → desde ASIGNACION hasta EXPRESION hacer BLOQUE | desde ASIGNACION hasta EXPRESION hacer COND
WRITE → escribe (WRITE_AUX)
WRITE_AUX → EXPRESION WRITE_AUXSUB
WRITE_AUXSUB → , WRITE_AUX | ϵ
READ → lee (READ_AUX)
READ_AUX→ id ARRDIM READ_AUXSUB
READ_AUXSUB →, READ_AUX | ϵ
RETURN → regresa (EXPRESION) | regresa (NULL)
CTE_ARR → { CTE_ARR_AUX } | { CTE_ARR_AUX2 }
CTE_ARR_AUX → CTE | CTE_ARR_AUXSUB
CTE_ARR_AUXSUB →, CTE_ARR_AUX | ϵ
CTE_ARR_AUX2 → { CTE_ARR_AUX } CTE_ARR_AUX2SUB
CTE_ARR_AUX2SUB → , CTE_ARR_AUX2 | ϵ

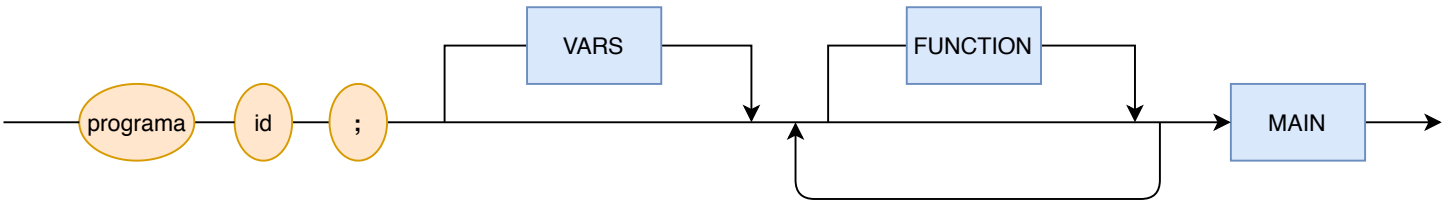
VI) Diagramas sintácticos

Ceasar Buenfil A01207499
Oscar Lerma A01380817

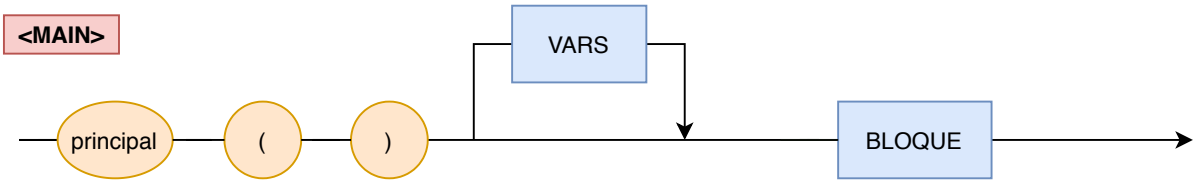
Diagramas Sintácticos

Patito++

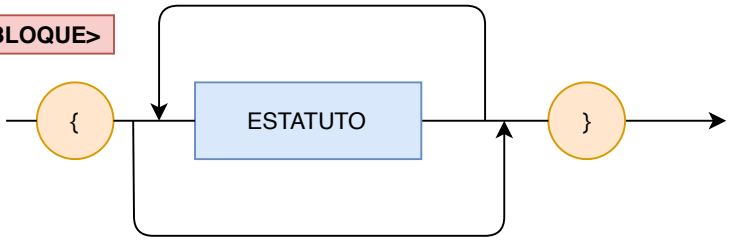
<PROGRAM>



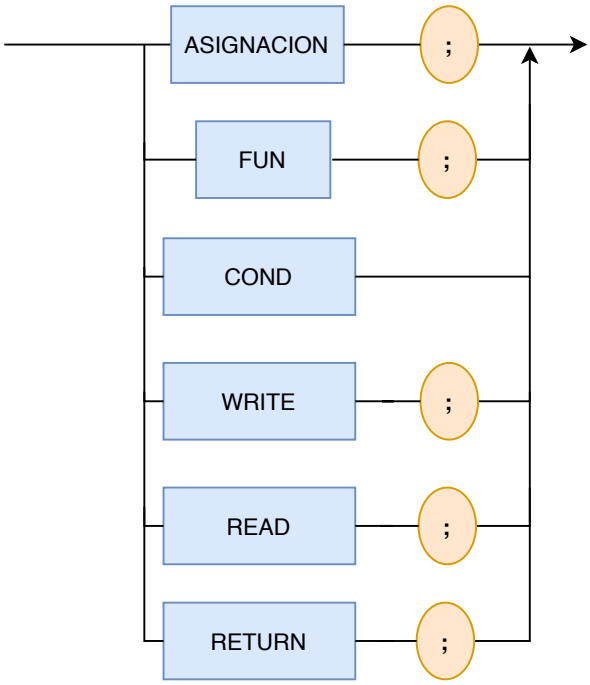
<MAIN>



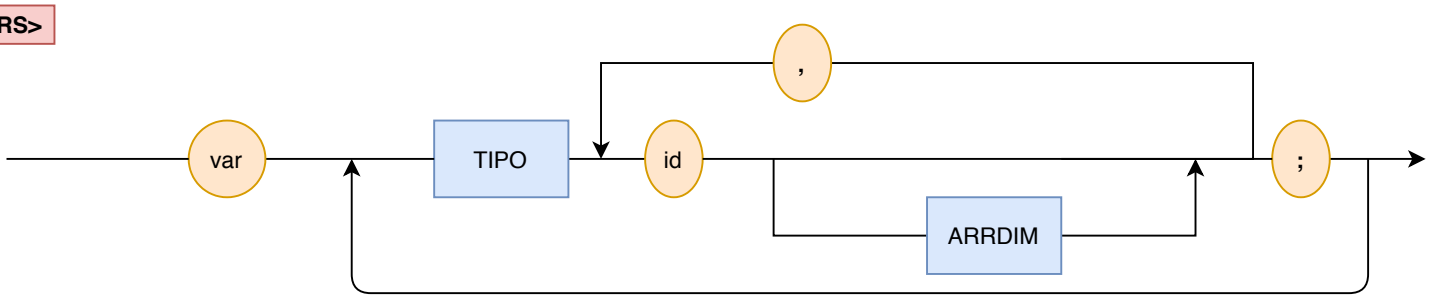
<BLOQUE>



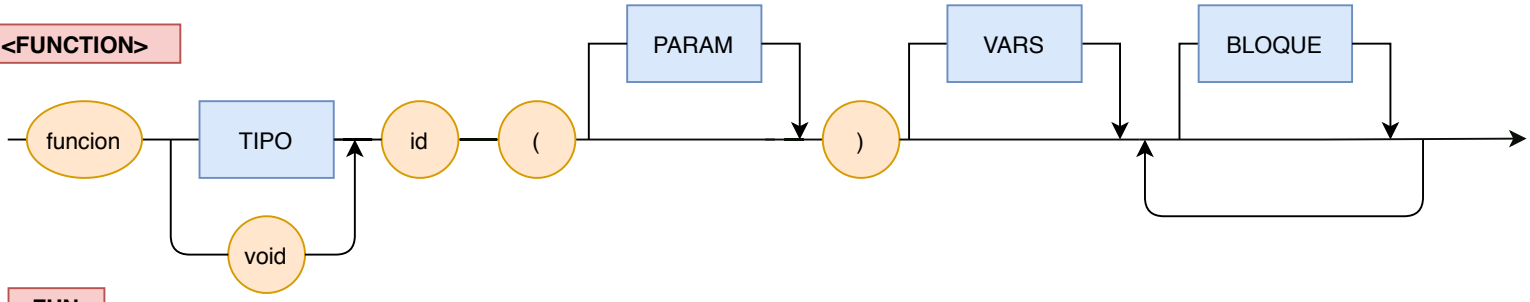
<ESTATUTO>



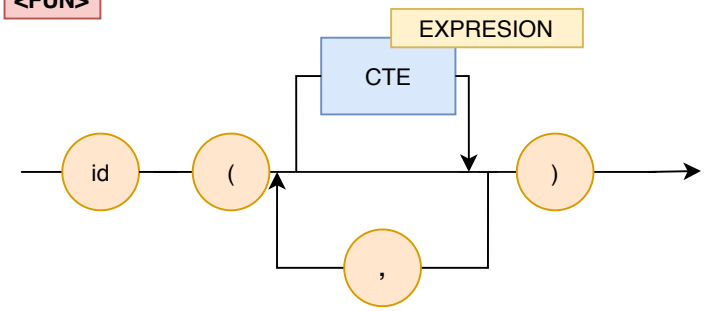
<VARS>



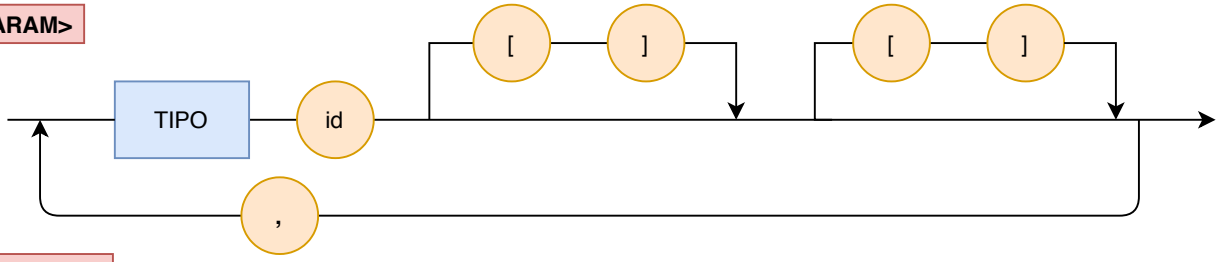
<FUNCTION>



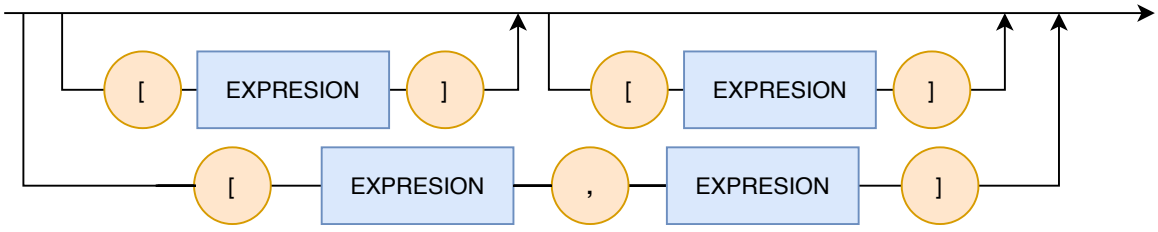
<FUN>

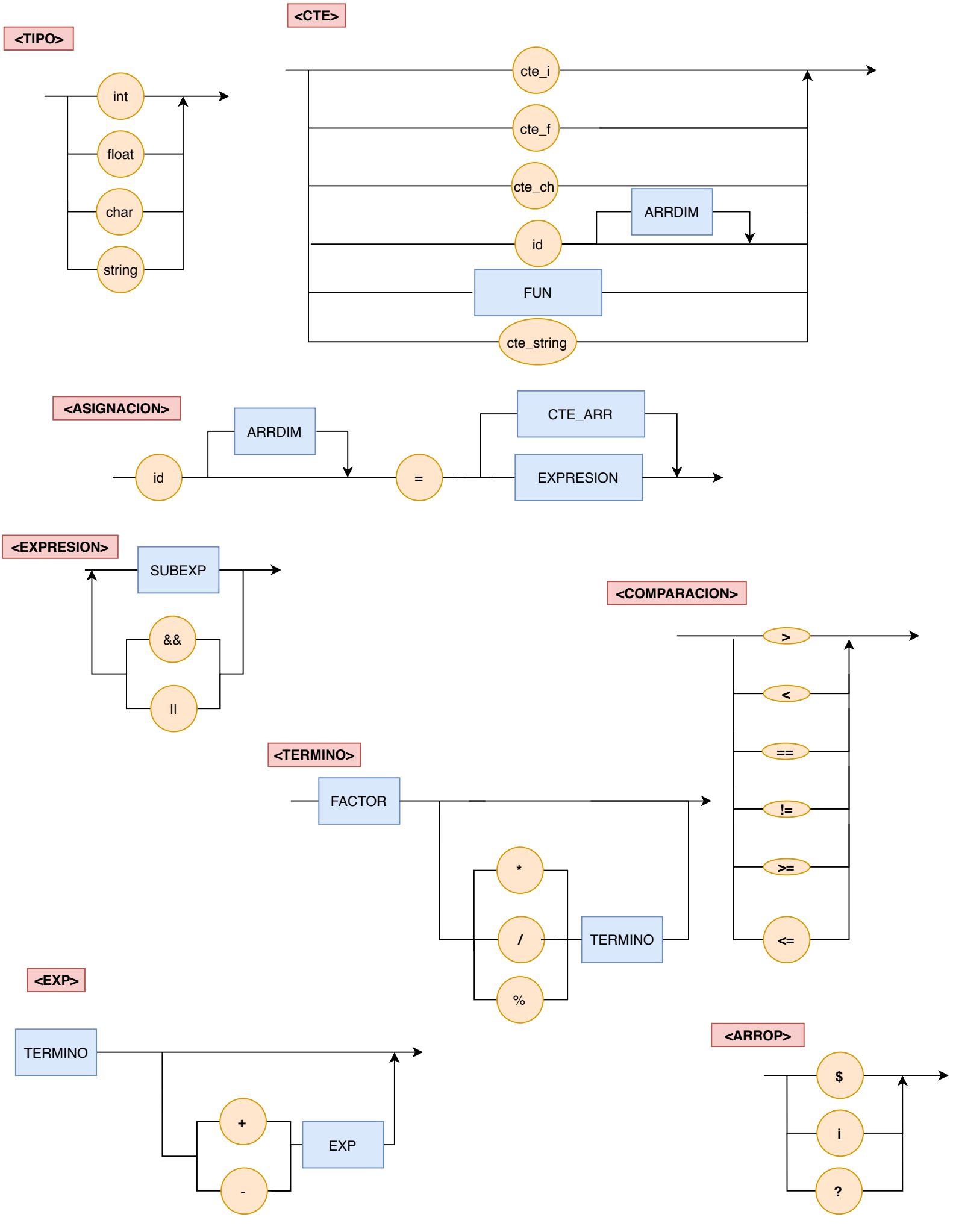


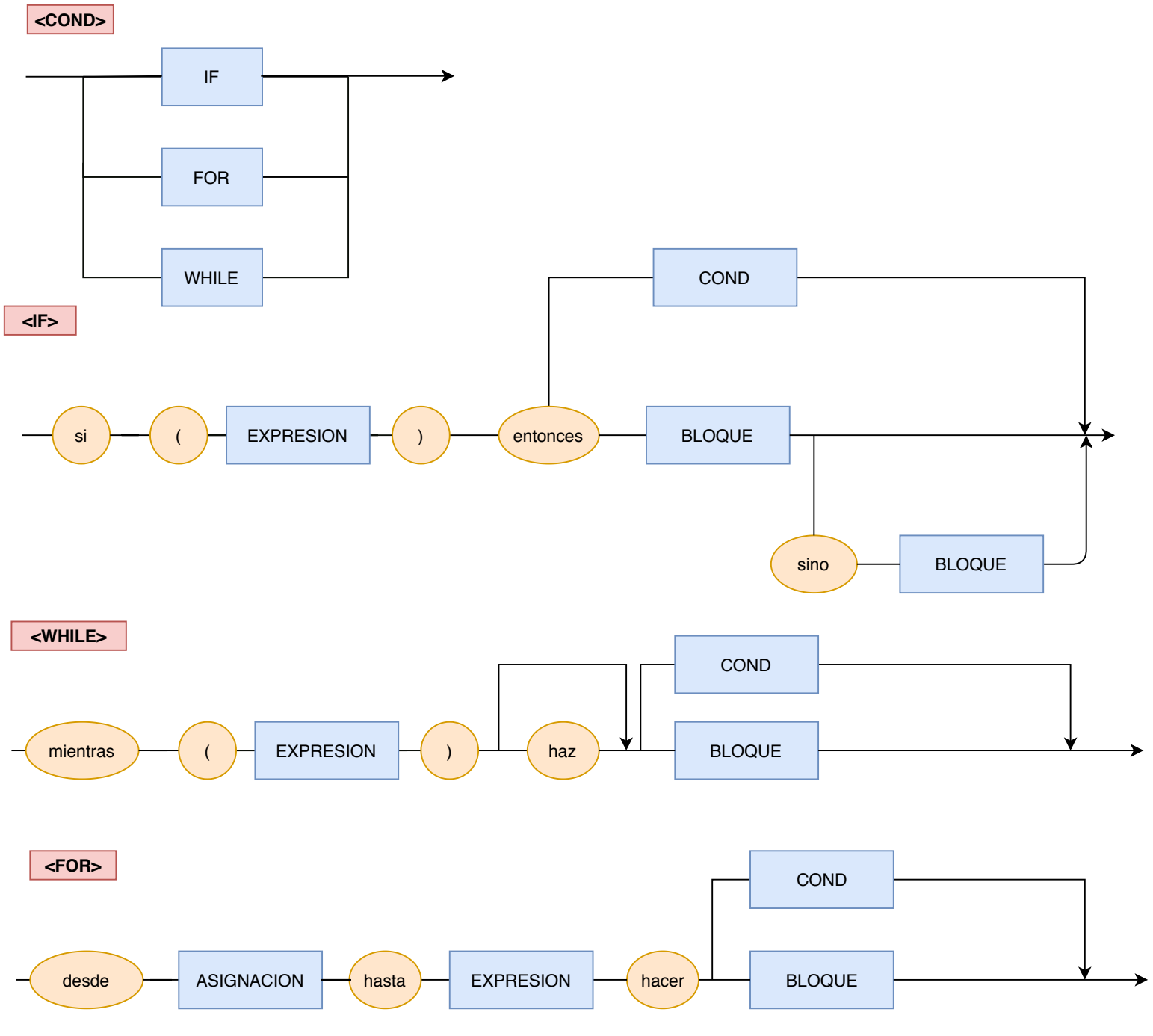
<PARAM>



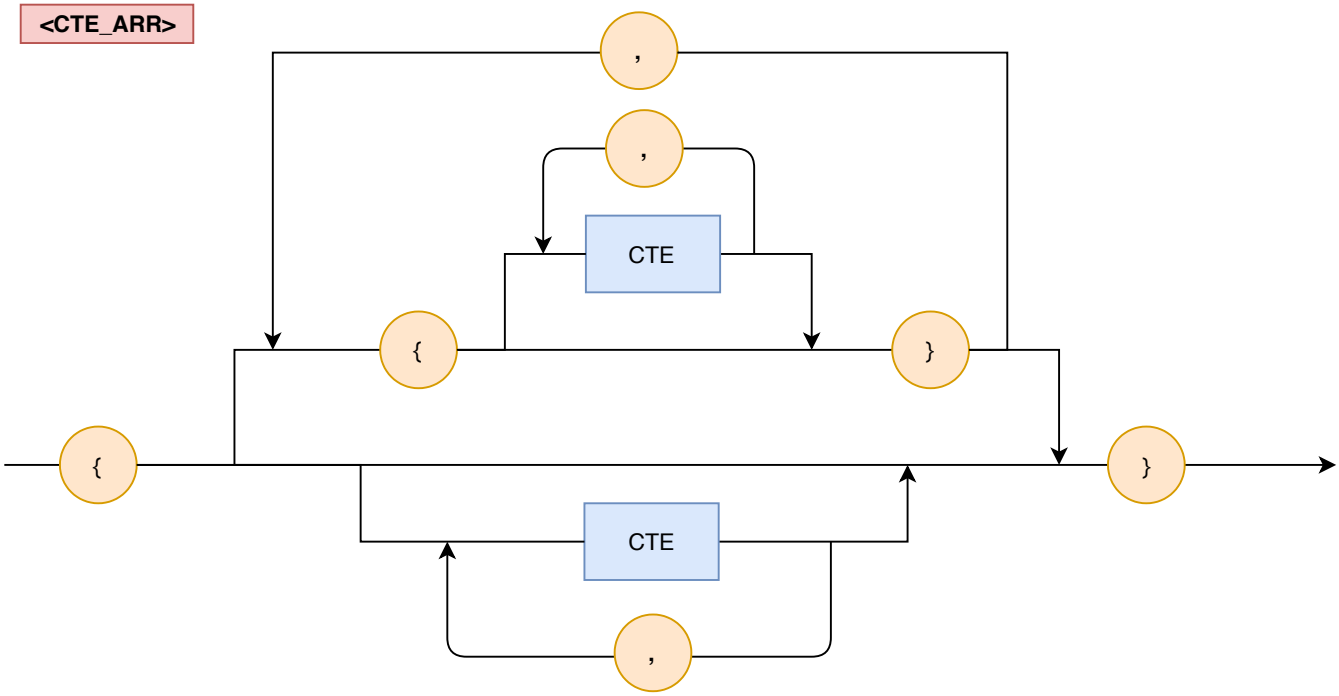
<ARRDIM>



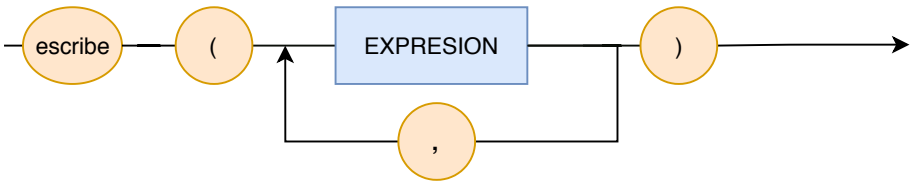

<TIPO>
int
float
char
string
<CTE>
cte_i
cte_f
cte_ch
ARRDIM
id
FUN
cte_string
<ASIGNACION>
ARRDIM
id
=
CTE_ARR
EXPRESION
<EXPRESION>
SUBEXP
&&
||
<COMPARACION>
>
<
==
!=
>=
<=
<TERMINO>
FACTOR
*
/
%
TERMINO
<EXP>
TERMINO
+
-
EXP
<ARROP>
$
i
?


<COND>
IF
FOR
WHILE
<IF>
COND
si
(
EXPRESION
)
entonces
BLOQUE
sino
BLOQUE
<WHILE>
COND
mientras
(
EXPRESION
)
haz
BLOQUE
<FOR>
COND
desde
ASIGNACION
hasta
EXPRESION
hacer
BLOQUE

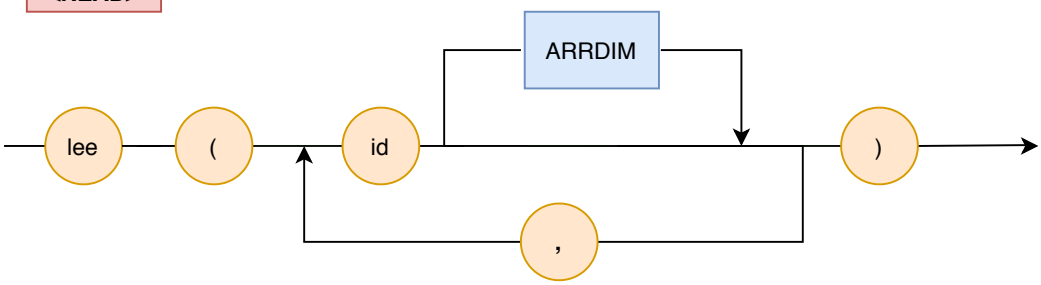
<CTE_ARR>



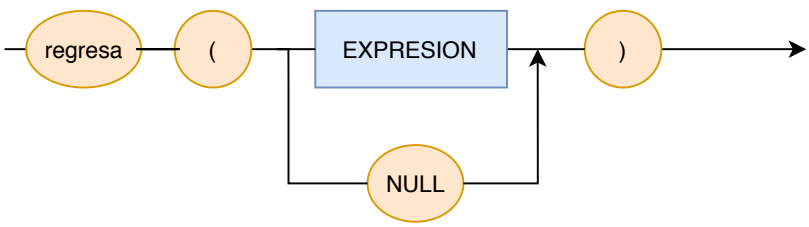
<WRITE>

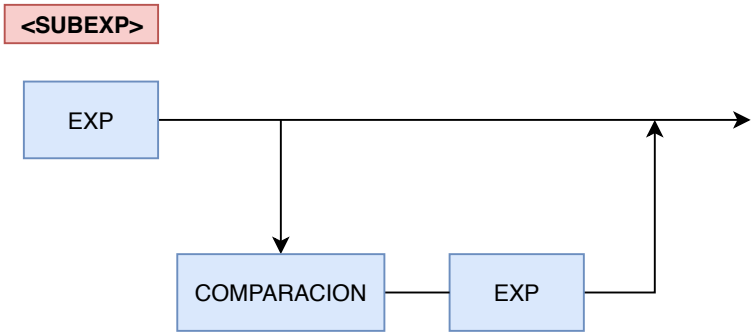


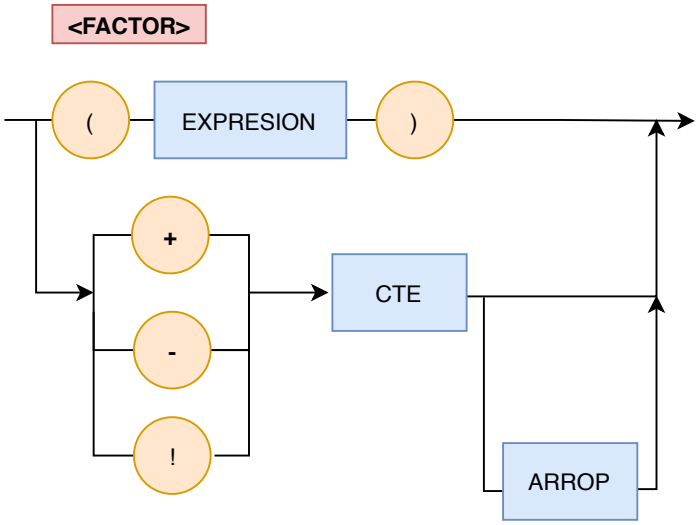
<READ>



<RETURN>




<SUBEXP>
EXP
COMPARACION
EXP


<FACTOR>
(
EXPRESION
)
+
-
!
CTE
ARROP