***TOKENS:***

***Tokens con RegEx***:

* id: [a-zA-Z][0-9a-zA-Z\_]\*
* cte\_int: ^[0-9]\*$
* cte\_float: ^[0-9]\*\.[0-9]{1,4}$
* cte\_string: \"[^"]\*\"

***Palabras Reservadas***:

* program - ^program$
* main
* end
* var
* int
* float
* print
* void
* if
* else
* while
* do

***Signos*:**

* ;
* ,
* :
* {
* }
* (
* )
* [ = \[
* ]
* +
* -
* \*
* /
* =
* <
* >
* !=

CONTEXT FREE GRAMMAR

< PROGRAMA>

PROGRAMA → program id ; VARS FUNCS main BODY end

<VARS>

VARS → var VARS’

<VARS’>

VARS’ → VARS’’:TYPE;

VARS’ → VARS’’:TYPE;VARS’

<VARS’’>

VARS’’ → id

VARS’’ → id, COM’

<TYPE>

TYPE → int

TYPE → float

<BODY>

BODY → {STATEMENT}

<STATEMENT>

STATEMENT → ε

STATEMENT → ASSIGN STATEMENT

STATEMENT → CONDITION STATEMENT

STATEMENT → CYCLE STATEMENT

STATEMENT → F\_CALL STATEMENT

STATEMENT → PRINT STATEMENT

<ASSIGN>

ASSIGN → id = EXPRESSION ;

<PRINT>

PRINT → print(PRINT’);

<PRINT’>

PRINT’ → cte.string PRINT’

PRINT’ → EXPRESSION PRINT’

<PRINT’’>

PRINT’’ → , cte.string PRINT’

PRINT’’ → , EXPRESSION PRINT’

<CYCLE>

CYCLE → while(EXPRESSION) do BODY

<CONDITION>

CONDITION → if(EXPRESSION) BODY COND;

<COND>

COND → ε

COND → else BODY

<EXPRESSION>

EXPRESSION→ EXP EXPRESION’

<EXPRESSION’>

EXPRESSION’→ COMP EXP

EXPRESSION’→ ε

<COMP>

COMP → >

COMP → <

COMP → !=

<EXP>

EXP → TERMINO EXP’

<EXP’>

EXP’ → SIGN EXP

EXP’ → ε

<SIGN>

SIGN → +

SIGN → -

SIGN → ε

<TERMINO>

TERMINO → FACTOR TERMINO’

<TERMINO’>

TERMINO’ → ε

TERMINO’ → OPER TERMINO

<OPER>

OPER → \*

OPER → /

OPER → ε

<FACTOR>

FACTOR → FACT FACTO

FACTOR → EXPRESSION

<FACTOR’>

FACTOR’ → SIGN

FACTOR’ → ε

<FACTOR’’>

FACTOR’’ → id

FACTOR’’ → CTE

<CTE>

CTE → cte\_int

CTE → cte\_float

<F\_CALL>

F\_CALL → id (CALL);

<CALL>

CALL → ε

CALL → EXPRESSION CALL’

<CALL’>

CALL’ → ε

CALL’ → , EXPRESSION CALL’

<FUNCS>

FUNCS → void id ( FUNC ) [FUNC\_VARS BODY];

<FUNC>

FUNC → FUNC’

FUNC → ε

<FUNC’>

FUNC’ → id: TYPE FUNC’’

<FUNC’’>

FUNC’’ → ε

FUNC’’ → , FUNC’