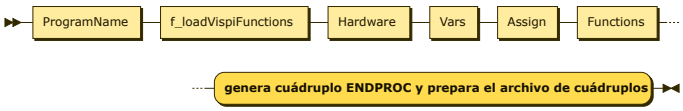
Program:




```
Program  ::= ProgramName f_loadVispiFunctions Hardware Vars Assign Functions 'genera cuádruplo ENDPROC y prepara el archivo de cuádruplos'
```

no references

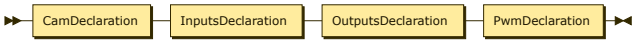
ProgramName:


```
ProgramName
         ::= 'program' 'id' '\n' 'genera el GOTO main y guarda el nombre del programa'
```

referenced by:

- Program

Hardware:

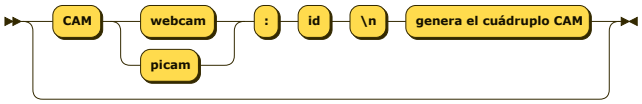



```
Hardware ::= CamDeclaration InputsDeclaration OutputsDeclaration PwmDeclaration
```

referenced by:

- Program

CamDeclaration:

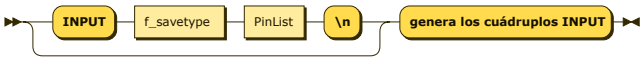



```
CamDeclaration
         ::= ( 'CAM' ( 'webcam' | 'picam' ) ':' 'id' '\n' 'genera el cuádruplo CAM' )?
```

referenced by:

- Hardware

InputsDeclaration:

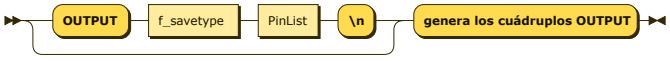



```
InputsDeclaration
         ::= ( 'INPUT' f_savetype PinList '\n' )? 'genera los cuádruplos INPUT'
```

referenced by:

- Hardware

OutputsDeclaration:

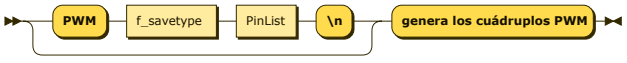



```
OutputsDeclaration
         ::= ( 'OUTPUT' f_savetype PinList '\n' )? 'genera los cuádruplos OUTPUT'
```

referenced by:

- Hardware

PwmDeclaration:




```
PwmDeclaration
         ::= ( 'PWM' f_savetype PinList '\n' )? 'genera los cuádruplos PWM'
```

referenced by:

- Hardware

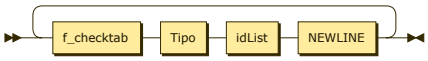
PinList:


```
PinList  ::= 'c_int' ':' 'id' ( ',' 'c_int' ':' 'id' )* 'guarda los id de pines como var. globales'
```

referenced by:

- InputsDeclaration
- OutputsDeclaration
- PwmDeclaration

Vars:

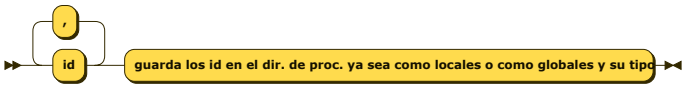



```
Vars     ::= ( f_checktab Tipo idList NEWLINE )+
```

referenced by:

- Program
- Statement

idList:

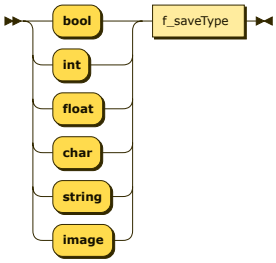



```
idList   ::= 'id' ( ',' 'id' )* 'guarda los id en el dir. de proc. ya sea como locales o como globales y su tipo'
```

referenced by:

- Vars

Tipo:

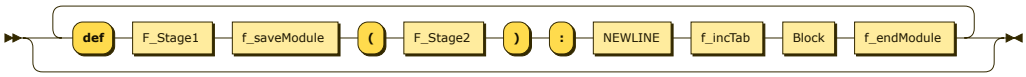



```
Tipo     ::= ( 'bool' | 'int' | 'float' | 'char' | 'string' | 'image' ) f_saveType
```

referenced by:

- F_Stage1
- F_Stage2
- Vars

Functions:

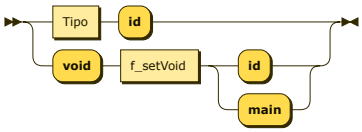



```
Functions
         ::= ( 'def' F_Stage1 f_saveModule '(' F_Stage2 ')' ':' NEWLINE f_incTab Block f_endModule )*
```

referenced by:

- Program

F_Stage1:

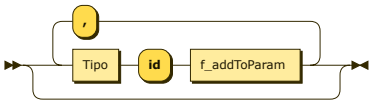



```
F_Stage1 ::= Tipo 'id'
           | 'void' f_setVoid ( 'id' | 'main' )
```

referenced by:

- Functions

F_Stage2:




```
F_Stage2 ::= ( Tipo 'id' f_addToParam ( ',' Tipo 'id' f_addToParam )* )?
```

referenced by:

- Functions

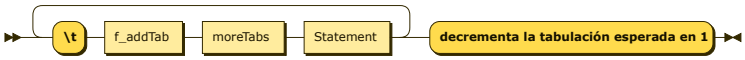
Assign:


```
Assign   ::= ( f_checkTab 'id' f_checkID '=' f_isAssign Expression NEWLINE f_generateEqual )+
```

referenced by:

- Program
- Statement

Block:

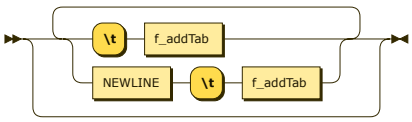



```
Block    ::= ( '\t' f_addTab moreTabs Statement )+ 'decrementa la tabulación esperada en 1'
```

referenced by:

- Condition
- Cycle
- DoCycle
- Functions

moreTabs:

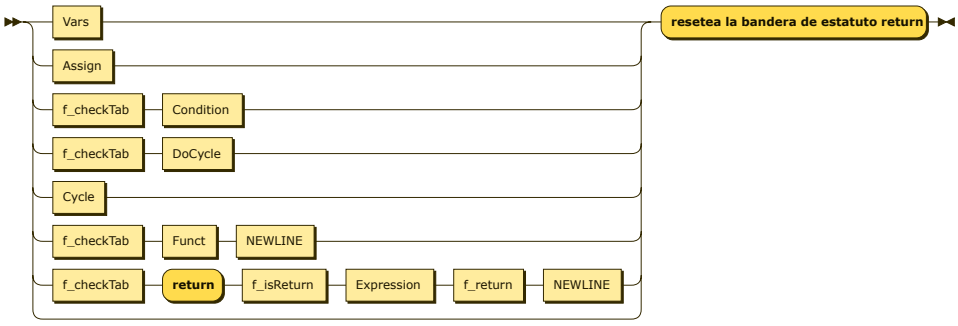



```
moreTabs ::= ( '\t' f_addTab | NEWLINE '\t' f_addTab )*
```

referenced by:

- Block

Statement:

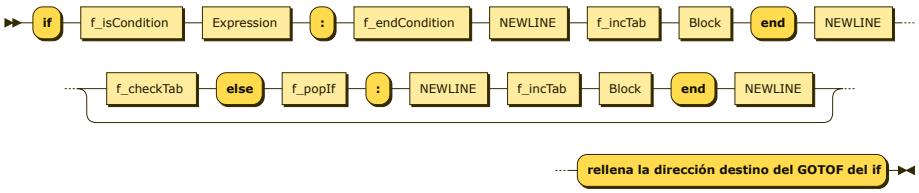



```
Statement
        ::= ( Vars | Assign | f_checkTab Condition | f_checkTab DoCycle | Cycle | f_checkTab Funct NEWLINE | f_checkTab 'return' f_isReturn Expression f_return NEWLINE )? 'resetea la
```

referenced by:

- Block

Condition:

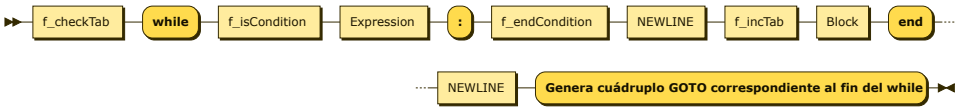



```
Condition
        ::= 'if' f_isCondition Expression ':' f_endCondition NEWLINE f_incTab Block 'end' NEWLINE ( f_checkTab 'else' f_popIf ':' NEWLINE f_incTab Block 'end' NEWLINE )? 'rellena la d
```

referenced by:

- Statement

Cycle:

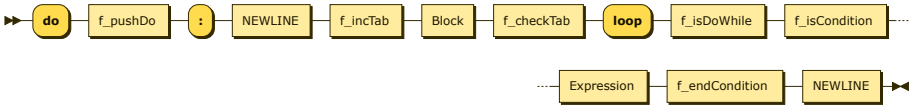



```
Cycle    ::= f_checkTab 'while' f_isCondition Expression ':' f_endCondition NEWLINE f_incTab Block 'end' NEWLINE 'Genera cuádruplo GOTO correspondiente al fin del while'
```

referenced by:

- Statement

DoCycle:

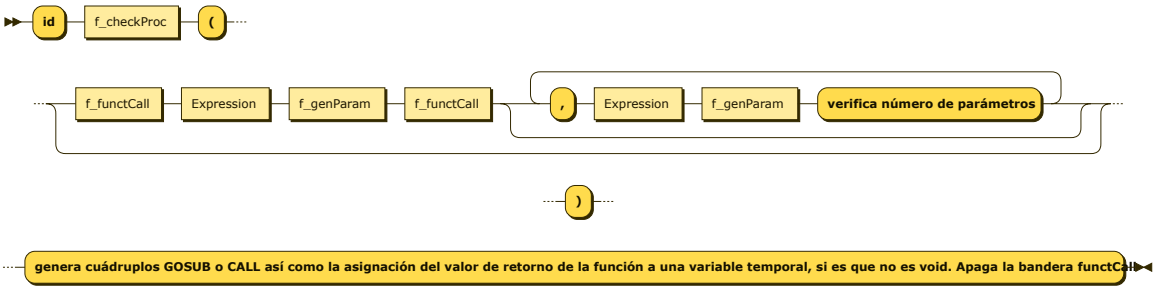



```
DoCycle  ::= 'do' f_pushDo ':' NEWLINE f_incTab Block f_checkTab 'loop' f_isDoWhile f_isCondition Expression f_endCondition NEWLINE
```

referenced by:

- Statement

Funct:

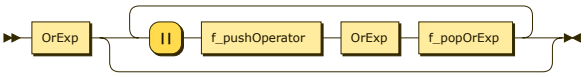



```
Funct    ::= 'id' f_checkProc '(' ( f_functCall Expression f_genParam f_functCall ( ',' Expression f_genParam 'verifica número de parámetros' )* )? ')' 'genera cuádruplos GOSUB o CALL
```

referenced by:

- Factor
- Statement

Expression:

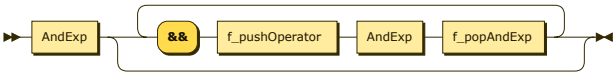



```
Expression
         ::= OrExp ( '||' f_pushOperator OrExp f_popOrExp )*
```

referenced by:

- Assign
- Condition
- Cycle
- DoCycle
- Factor
- Funct
- Statement

OrExp:




```
OrExp    ::= AndExp ( '&&' f_pushOperator AndExp f_popAndExp )*
```

referenced by:

- Expression

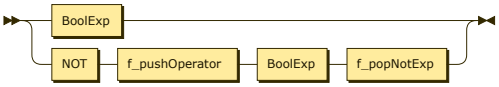
AndExp:


```
AndExp   ::= NotExp
```

referenced by:

- OrExp

NotExp:

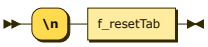



```
NotExp   ::= BoolExp
           | NOT f_pushOperator BoolExp f_popNotExp
```

referenced by:

- AndExp

NEWLINE:

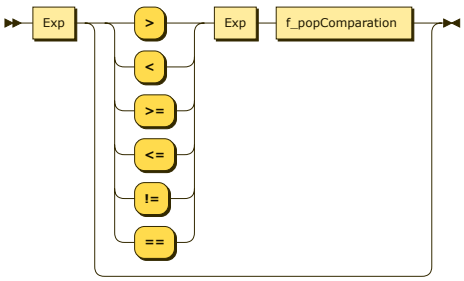



```
NEWLINE  ::= '\n' f_resetTab
```

referenced by:

- Assign
- Condition
- Cycle
- DoCycle
- Functions
- Statement
- Vars
- moreTabs

BoolExp:

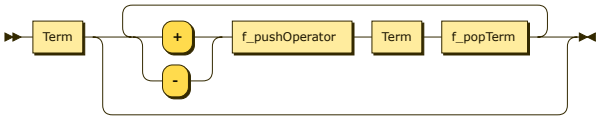



```
BoolExp  ::= Exp ( ( '>' | '<' | '>=' | '<=' | '!=' | '==' ) Exp f_popComparation )?
```

referenced by:

- NotExp

Exp:

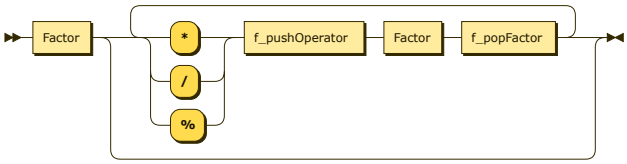



```
Exp      ::= Term ( ( '+' | '-' ) f_pushOperator Term f_popTerm )*
```

referenced by:

- BoolExp

Term:

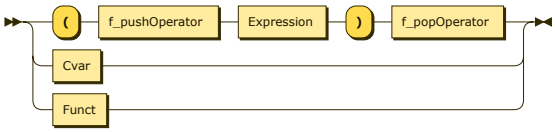



```
Term     ::= Factor ( ( '*' | '/' | '%' ) f_pushOperator Factor f_popFactor )*
```

referenced by:

- Exp

Factor:

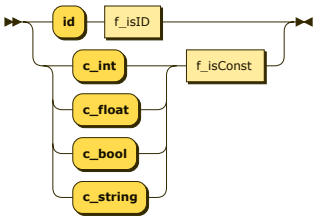



```
Factor   ::= '(' f_pushOperator Expression ')' f_popOperator
           | Cvar
           | Funct
```

referenced by:

- Term

Cvar:




```
Cvar     ::= 'id' f_isID
           | ( 'c_int' | 'c_float' | 'c_bool' | 'c_string' ) f_isConst
```

referenced by:

- Factor

f_loadVispiFunctions:


```
f_loadVispiFunctions
         ::= 'carga las funciones predefinidas de nuestro lenguaje en el dir. de procedimientos'
```

referenced by:

- Program

f_saveType:


```
f_saveType
         ::= 'guarda el tipo de dato correspondiente al HW o variable primitiva declarada para su uso futuro'
```

referenced by:

- Tipo

f_setVoid:


```
f_setVoid
         ::= 'guarda "void" en el tipo de dato'
```

referenced by:

- F_Stage1

f_saveModule:


```
f_saveModule
         ::= 'verifica si el id definido ya existe. Declara un nuevo procedimiento en el directorio'
```

referenced by:

- Functions

f_addToParam:


```
f_addToParam
         ::= 'agrega el parametro declarado a su lugar en el dir. de procedimientos'
```

referenced by:

- F_Stage2

f_resetTab:


```
f_resetTab
         ::= 'reinicia el contador de tabs a 0'
```

referenced by:

- NEWLINE

f_incTab:


```
f_incTab ::= 'incrementa en 1 la tabulación esperada a partir de este punto'
```

referenced by:

- Condition
- Cycle
- DoCycle
- Functions

f_decTab:


```
f_decTab ::= 'decrementa en 1 la tabulación esperada a partir de este punto'
```

no references

f_endModule:


```
f_endModule
        ::= 'guarda el tamaño de la funcion declarada, reinicia contadores de variables y genera el cuad. RET'
```

referenced by:

- Functions

f_checkTab:


```
f_checkTab
        ::= 'verifica que el contador de tabs corresponda con lo esperado'
```

referenced by:

- Assign
- Condition
- Cycle
- DoCycle
- Statement

f_checkID:


```
f_checkID
        ::= 'verifica que el id ya haya sido declarado como local o global. Si sí, inserta operando y tipo en sus pilas respectivas'
```

referenced by:

- Assign

f_isAssign:


```
f_isAssign
        ::= 'enciende una bandera para saber que se procesa una asignación'
```

referenced by:

- Assign

f_generateEqual:


```
f_generateEqual
        ::= 'hace 2 pop de la pila de tipos y de operandos, verifica tipos y genera el cuádruplo de la asignación'
```

referenced by:

- Assign

f_addTab:


```
f_addTab ::= 'incrementa en 1 el contador de tabs y verifica que corresponda con la tabulación esperada'
```

referenced by:

- Block
- moreTabs

f_return:


```
f_return ::= 'marca error si hay return en función void. Verifica que el valor de retorno (top de pila) sea del tipo adecuado. Genera el cuádruplo RETURN'
```

referenced by:

- Statement

f_isReturn:


```
f_isReturn
        ::= 'enciende una bandera que indica que el estatuto es un return'
```

referenced by:

- Statement

f_isCondition:


```
f_isCondition
        ::= 'enciende una bandera para indicar que se está procesando una condición o ciclo. Si es del tipo while, se inserta el número de cuádruplo en pilaDeBranch'
```

referenced by:

- Condition
- Cycle
- DoCycle

f_popIf:


```
f_popIf  ::= 'dado que vino un else, genera el GOTO del final del if e inserta en una pilaDeBranch el número del siguiente cuádruplo'
```

referenced by:

- Condition

f_pushDo:


```
f_pushDo ::= 'genera un cuádruplo DO que indica el inicio de un do..loop e inserta su número en la pilaDeBranch'
```

referenced by:

- DoCycle

f_isDoWhile:


```
f_isDoWhile
        ::= 'enciende una bandera que indica que se está procesando un do..loop'
```

referenced by:

- DoCycle

f_checkProc:


```
f_checkProc
        ::= 'verifica que el módulo llamado esté definido y genera el cuádruplo ERA'
```

referenced by:

- Funct

f_genParam:


```
f_genParam
        ::= 'dados los parámetros de la función, genera los cuádruplos PARAM verificando el tipo'
```

referenced by:

- Funct

f_popOrExp:


```
f_popOrExp
        ::= 'si la operación es || verifica tipos en el cubo semántico y genera el cuádruplo'
```

referenced by:

- Expression

f_popAndExp:


```
f_popAndExp
        ::= 'si la operación es && verifica tipos en el cubo semántico y genera el cuádruplo'
```

referenced by:

- OrExp

f_popNotExp:


```
f_popNotExp
        ::= 'si la operación es ! verifica el tipo en el cubo semántico y genera el cuádruplo'
```

referenced by:

- NotExp

f_popComparation:


```
f_popComparation
        ::= 'si se hizo una comparación: verifica tipo en el cubo semántico, genera una temporal con el resultado (a la pilaOperandos) y genera el cuádruplo correspondiente'
```

referenced by:

- BoolExp

f_popTerm:


```
f_popTerm
        ::= 'si la operación fue + o -: verifica tipos en el cubo semántico, genera una temporal con el resultado (a la pila pilaOperandos) y genera el cuádruplo'
```

referenced by:

- Exp

f_popFactor:

```
f_popFactor
        ::= 'si es * / o % la operación: verifica tipos en el cubo semántico y genera el cuádruplo'
```

referenced by:

- Term

f_pushOperator:


```
f_pushOperator
        ::= 'siempre que la bandera de condición esté apagada inserta el operador en su pila'
```

referenced by:

- Exp
- Expression
- Factor
- NotExp
- OrExp
- Term

f_popOperator:


```
f_popOperator
        ::= 'hace pop del operador en la cima del stack'
```

referenced by:

- Factor

f_isID:


```
f_isID   ::= 'verifica que el id esté declarado como local o global'
```

referenced by:

- Cvar

f_isConst:


```
f_isConst
        ::= 'guarda la constante en la tabla de constantes junto con su tipo y los inserta en la pila de operandos y tipos'
```

referenced by:

- Cvar

f_endCondition:


```
f_endCondition
        ::= 'verifica que tipo del operando en stack.top sea bool o int. Genera GOTOF y hace push a pilaDeBranch si es if o while, o hace pop a pilaDeBranch y genera GOTOT si es do'
```

referenced by:

- Condition
- Cycle
- DoCycle

f_functCall:


```
f_functCall
        ::= 'enciende una bandera que indica que se procesa una llamada a función'
```

referenced by:

- Funct
