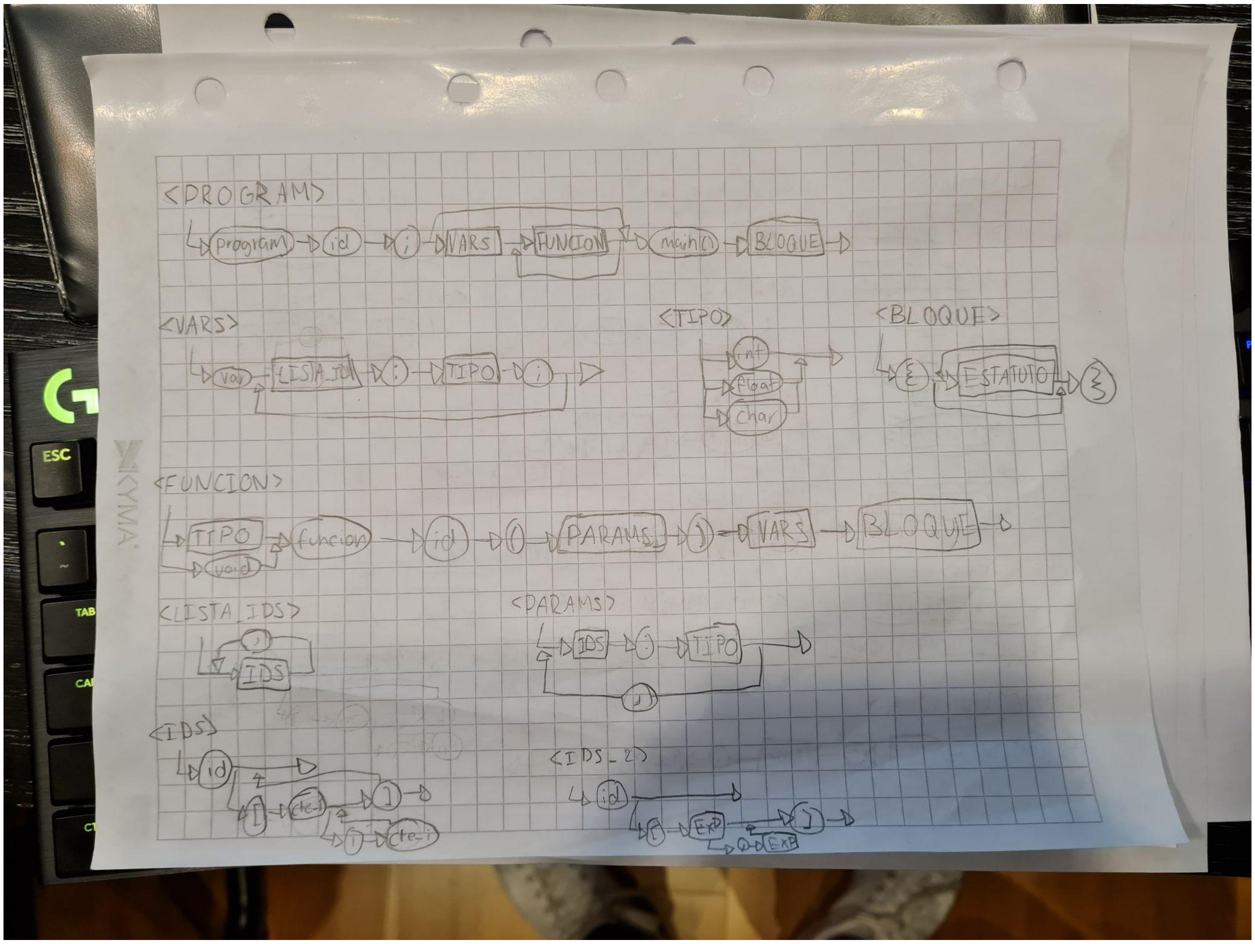
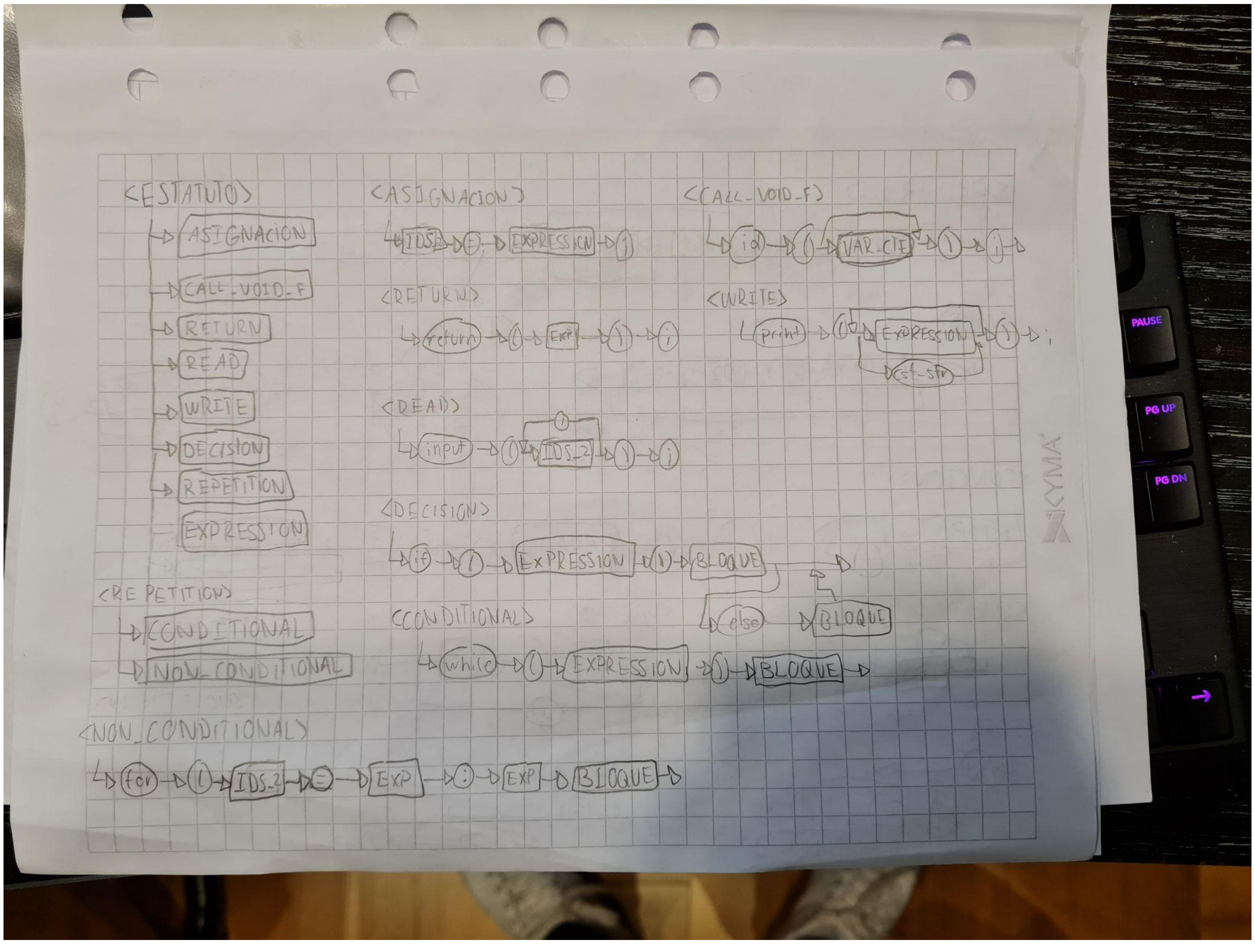
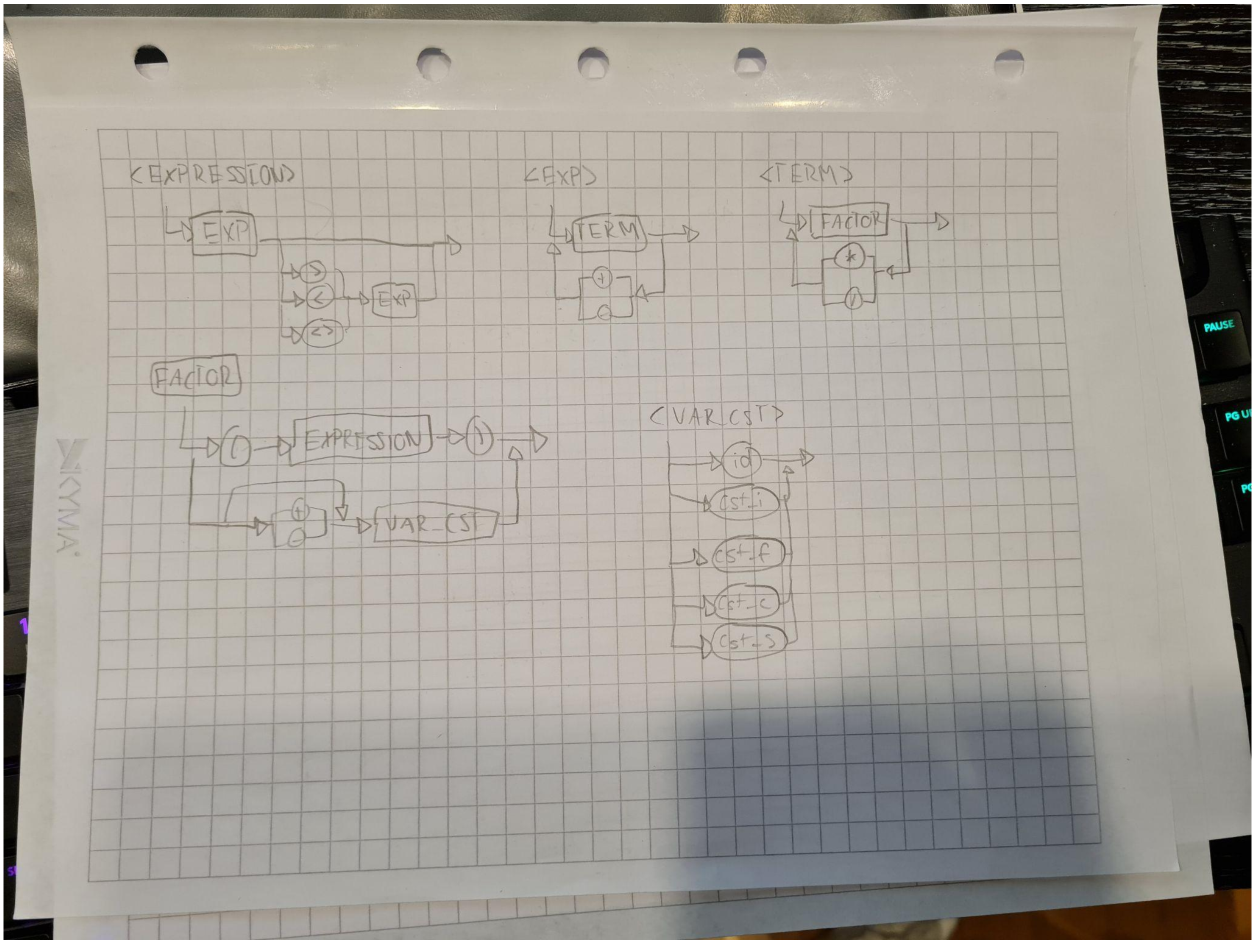

<PROGRAM>
program
id
;
VARS
FUNCION
main()
BLOQUE
<VARS>
var
LISTA_ID
:
TIPO
;
<TIPO>
int
float
char
<BLOQUE>
{
ESTATUTO
}
<FUNCION>
TIPO
void
funcion
id
(
PARAMS
)
VARS
BLOQUE
<LISTA_IDS>
,
IDS
<PARAMS>
IDS
:
TIPO
,
<IDS>
id
[
cte-i
,
cte-i
]
<IDS_2>
id
[
EXP
,
EXP
]


<ESTATUTO>
ASIGNACION
CALL_VOID_F
RETURN
READ
WRITE
DECISION
REPETITION
EXPRESSION
<ASIGNACION>
IDS_2
=
EXPRESSION
;
<CALL_VOID_F>
id
(
VAR_CTE
)
;
<RETURN>
return
(
EXP
)
;
<WRITE>
print
(
EXPRESSION
cte_str
)
;
<READ>
input
(
,
IDS_2
)
;
<DECISION>
if
(
EXPRESSION
)
BLOQUE
else
BLOQUE
<REPETITION>
CONDITIONAL
NON_CONDITIONAL
<CONDITIONAL>
while
(
EXPRESSION
)
BLOQUE
<NON_CONDITIONAL>
for
(
IDS_2
=
EXP
:
EXP
BLOQUE
KYMA



Palabras reservadas:

program = ‘program’

function = ‘function’

var = ‘var’

int = ‘int’

float = ‘float’

Char = ‘char’

Void = ‘void’

print = ‘print’

input = ‘input’

if = ‘if’

else = ‘else’

main = ‘main’

return = ‘return

while = ‘while’

for = ‘for’

null = ‘null’

Expresiones regulares

```
# Regexpr para tokens simples

t_SEMICOLON    = r';'

t_COLON        = r':'

t_COMA         = r','

t_LBRACE       = r'\{'

t_RBRACE       = r'\}'

t_EQUAL        = r'='

t_PLUS         = r'\+'

t_MINUS        = r'-'

t_MULTIPLY     = r'\*'

t_DIVIDE       = r'/'

t_LPAREN       = r'\('
t_RPAREN        = r'\)'

t_LBRACKET = r’\[’

t_RBRACKET = r’\]’

t_CST_STRING    = r'("(\\"|[^"])*")'

t_CST_INT       = r'[0-9]+'

t_CST_FLOAT     = r'[0-9]+\.[0-9]+'

t_CST_CHAR = r’[a-zA-Z]’

t_LT            = r'<'

t_GT            = r'>'

t_NE          = r’<>’
```

Gramatica:

PROGRAMA → program id t_SEMICOLON VARS FUNCTIONS main t_LPAREN t_RPAREN BLOCK| program id t_SEMICOLON main t_LPAREN t_RPAREN BLOCK

VARS → var VARS’

VARS’ → LIST_IDS t_COLON TYPE t_SEMICOLON VARS’’

VARS’’ → VARS’ | null

FUNCTIONS → FUNCTION | FUNCTIONS’

FUNCTIONS’ → FUNCTIONS | null

LIST_IDS → IDS LIST_IDS’

LIST_IDS’ → t_COMA IDS LIST_IDS’ | null

IDS → id | id t_LBRACKET t_CST_INT t_RBRACKET | id t_LBRACKET t_CST_INT t_RBRACKET t_LBRACKET t_CST_INT t_RBRACKET

IDS_2 → id | id t_LBRACKET EXP t_RBRACKET | id t_LBRACKET EXP t_RBRACKET t_LBRACKET EXP t_RBRACKET

TYPE→ int | float | char

RETURN_TYPE → TYPE | void

FUNCTION → RETURN_TYPE function id t_LPAREN PARAMS t_RPAREN VARS BLOCK

PARAMS → IDS t_COLON TIPO PARAMS’

PARAMS’ → PARAMS t_COMA | null

BLOCK → t_LBRACE STATUTES t_RBRACE

STATUTES → STATUTE | STATUTES’

STATUTES’ → STATUTES | null

STATUTE→ ASSIGNMENT | CONDITION | WRITE | CALL_VOID_F | RETURN | READ | DECISION | REPETITION | EXPRESSION

ASSIGNMENT → IDS_2 t_EQUAL EXPRESSION t_SEMICOLON

WRITE → print t_LPAREN WRITE’ t_RPAREN t_SEMICOLON

WRITE’→ EXPRESSION WRITE’’ | cst_String WRITE’’

WRITE’’ → t_COMA WRITE’ | null

CALL_VOID_F → id t_LPAREN CALL_VOID_F’ t_RPAREN t_SEMICOLON

CALL_VOID_F’ → VAR_CTE | null

RETURN → return t_LPAREN EXP t_RPAREN t_SEMICOLON

READ → input t_LPAREN READ’ t_RPAREN t_SEMICOLON

READ’ → IDS_2 READ’’

READ’’ → t_COMA READ’ | null

REPETITION → CONDITIONAL | NON-CONDITIONAL

DECISION → if t_LPAREN EXPRESSION t_RPAREN BLOCK ELSE t_SEMICOLON | if t_LPAREN EXPRESSION t_RPAREN BLOCK

CONDITIONAL → while t_LPAREN EXPRESSION t_RPAREN BLOCK

NON-CONDITIONAL → for t_LPAREN IDS_2 t_EQUAL EXP T_COLON EXP BLOCK

EXPRESSION → EXP EXPRESSION’

EXPRESSION’ → GT EXP | LT EXP | NE EXP | null

EXP → TERM EXP’

EXP’ → t_PLUS EXP | t_MINUS EXP | null

ELSE→ else BLOCK | null

TERM → FACTOR TERM’

TERM’ → t_MULTIPLY TERM | t_DIVIDE TERM | null

FACTOR → t_LPAREN EXPRESSION t_RPAREN| FACTOR’

FACTOR’ → t_PLUS VAR_CST | t_MINUS VAR_CST | VAR_CST

VAR_CST → id | t_CST_FLOAT t |t_CST_INT | t_CST_CHAR | t_CST_STRING