L# :

statement

```
start              :   program;

program            :   program unit
                   |   unit
                   ;
```

just_in=0

func_definition

↳ RPAREN { }

just_in=1

```
unit               :   var_declaration
                   |   func_declaration
                   |   func_definition
                   ;

func_declaration   :   type_specifier ID LPAREN parameter_list RPAREN SEMICOLON
                   |   type_specifier ID LPAREN RPAREN SEMICOLON
                   ;
```

{ } → Enter_scope();
symbolTable.insert

```
func_definition    :   type_specifier ID LPAREN parameter_list RPAREN { }
                       compound_statement
                   |   type_specifier ID LPAREN RPAREN compound_statement
                   ;
```

vector<obj> → name, type, size → is_array

```
parameter_list     :   parameter_list COMMA type_specifier ID
                   |   parameter_list COMMA type_specifier
                   |   type_specifier ID
                   |   type_specifier
                   ;
```

if(just_in=0)
Enter_scope;
else just_in=0;

{ symbolTable.curr varoff = var_off; }

```
compound_statement :   LCURL statements RCURL
                   |   LCURL RCURL
                   ;
```

→ { }
↳ var_off =symbolTable.curr_sope.var_off.

"ADD SP, . . "

if(return-present=0)
{

```
var_declaration    :   type_specifier declaration_list SEMICOLON
                   ;
type_specifier     :   INT
                   |   FLOAT
                   |   VOID
                   ;
```

~~func~~
ID2PAR P.. RPAREN
COMPOUND

```
declaration_list   :   declaration_list COMMA ID
                   |   declaration_list COMMA ID LTHIRD CONST_INT RTHIRD
                   |   ID
                   |   ID LTHIRD CONST_INT RTHIRD
                   ;

statements         :   statement
                   |   statements statement
                   ;
```

→ { }

```
statement            :   var_declaration
                     |   expression_statement
                     |   compound_statement
                     |   FOR LPAREN expression_statement expression_statement expression
                         RPAREN statement
                     |   IF LPAREN expression RPAREN statement
                     |   IF LPAREN expression RPAREN statement ELSE statement
                     |   WHILE LPAREN expression RPAREN statement
                     |   PRINTLN LPAREN ID RPAREN SEMICOLON
                     |   RETURN expression SEMICOLON
                     ;

expression_statement :   SEMICOLON
                     |   expression SEMICOLON
                     ;

variable             :   ID
                     |   ID LTHIRD expression RTHIRD
                     ;

expression           :   logic_expression
                     |   variable ASSIGNOP logic_expression
                     ;

logic_expression     :   rel_expression
                     |   rel_expression LOGICOP rel_expression
                     ;

rel_expression       :   simple_expression
                     |   simple_expression RELOP simple_expression
                     ;

simple_expression    :   term
                     |   simple_expression ADDOP term
                     ;
term                 :   unary_expression
                     |   term MULOP unary_expression
                     ;

unary_expression     :   ADDOP unary_expression
                     |   NOT unary_expression
                     |   factor
                     ;
```

int parameter_variable

retur ~~0~~ i,

ADD SP, i

1 POP BP

RET parameter-varible*2

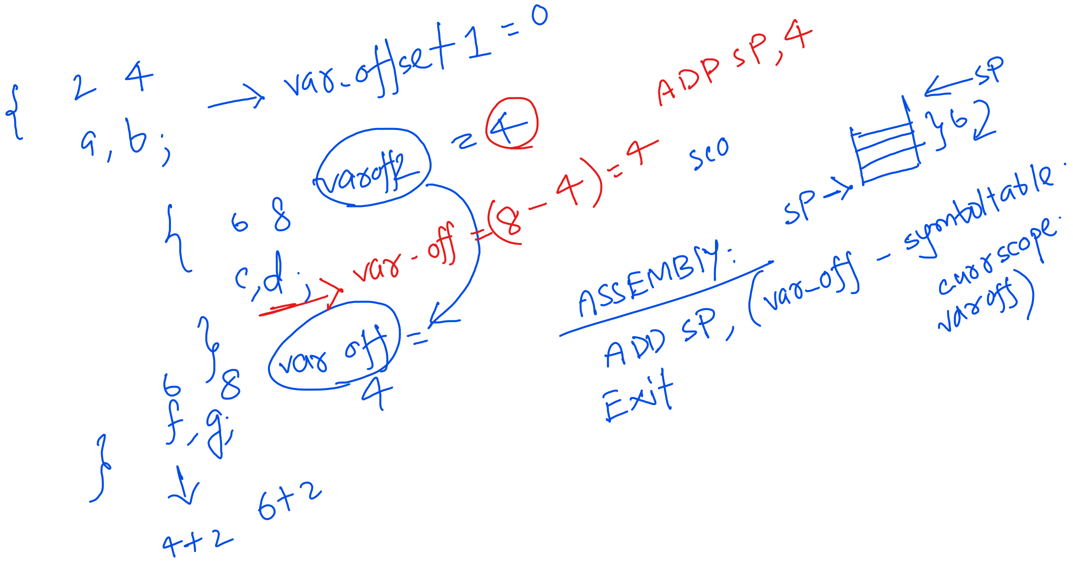
ADD SP, 8

func
→ No Return ✗ → "RET"
→ Return
→ main

MOV AX, 4CH
INT 21H

```
       factor   :  variable
                |  ID LPAREN argument_list RPAREN
                |  LPAREN expression RPAREN
                |  CONST_INT
                |  CONST_FLOAT
                |  variable INCOP
                |  variable DECOP
                ;

argument_list   :  arguments
                |
                ;
    arguments   :  arguments COMMA logic_expression
                |  logic_expression
                ;
```


{ 2 4
a, b; → var_offset1 = 0
varoff2 = 4
ADP SP, 4
{ 6 8
c, d; → var - off = (8 - 4) = 4
sco
var off = 4
} 8
6
f, g;
}
4+2 6+2
SP
SP →
} 6 2
ASSEMBLY:
ADD SP, (var_off - symboltable.currscope.varoff)
Exit