TC3048.1 Diseño de Compiladores

Tecnológico de Monterrey

EMIL

Ing. Elda Quiroga

Dr. Héctor Ceballos

Nina Sepúlveda Conde
A01194378

Project Description

Project Description

Purpose and Scope
EMIL is an R-like language which has built in mathematical expressions for the simplification of statistical problems.

Requirement Analysis and Main Test Case Description

Requirements

R1	Assignments
R2	Conditions
R3	Cycles
R4	Read and write
R5	Arithmetics, logical and relational
R6	Modules
R7	Arrays and Matrices

Use Cases

UC1	
UC2	

General Process Description

Date	Description	Commit
31/10/23	Initial commit, lexer and parser init.	Init
01/11/23	Grammar functional Expression quads generated Term quads generated Relational quads generated	Grammar E Quads T Quads R Quads
07/11/23	Symbols Table created Constant Directory created	Symbol Table Cte. Dir.
08/11/23	Symbols Table functional Semantic checks	Commit
14/11/23	Conditional quads generated Cyclical quads generated	If-else Quad While Quad
16/11/23	Variable address incorporated	Commit

17/11/23	Function declaration quads created	Funcdecl Quad
18/11/23	Function declarations quads functional	Funcdecl Quad
19/11/23	Function call quads functional	Funccall Quad
21/11/23	VM Created	VM Init
22/11/23	VM - GOTO ERA SUM	Commit

Personal Reflection

Language Description

Language Name

EMIL

Generic description of the main language characteristics

EMIL is an R-like language which has built in mathematical expressions for the simplification of statistical problems.

List of possible errors

- **Compilation Errors**
  - ERROR - Cannot use functions as variables
  - ERROR - Variable no declarada
  - ERROR - Variable already declared
  - ERROR - Cannot name a function after a variable
  - ERROR - Non-Void Functions must have a return
  - ERROR - Función no declarada
  - ERROR - Too many arguments
  - ERROR - Argument mismatch
  - ERROR - Not a boolean expression
  - ERROR - Not a boolean expression
- **Execution Errors**

Compiler Description

Computational Equipment, language and special utility

EMIL was developed in a Windows 10 computer. The lexer and parser were developed using Python, with the help of SLY. I chose SLY for its easy readability and its help with grammar.

Lexical Analysis

Tokens

TOKEN	REGEX
CTE_FLT	r'-?\d+\.\d+'
CTE_NUM	r'-?\d+'
CTE_STR	r'\"[^\"\n]*\"'

ID	r'[a-z][a-z0-9_]*'
SEMICLN	r'\;'
COLON	r'\:'
SUM	r'\+'
SUB	r’\-’
MULT	r’*’
DIV	r’\/’
EQUAL_TO	r'=='
LESS_OR_EQ_THAN	r'<='
MORE_OR_EQ_THAN	r'>='
ASS	r'='
DIFFERENT_TO	r'<>'
LESS_THAN	r'<'
MORE_THAN	r'>'
COMMA	r','
LPAREN	r'\('
RPAREN	r'\)'
LSQUARE	r'\['
RSQUARE	r'\]'
LCURLY	r'\{'
RCURLY	r'\}'
VARS	r'VARS'
MAIN	ID['main']
PROGRAM	ID[‘program’]
INT	ID[‘int’]
CHAR	ID[‘char’]
FLOAT	ID[‘float’]

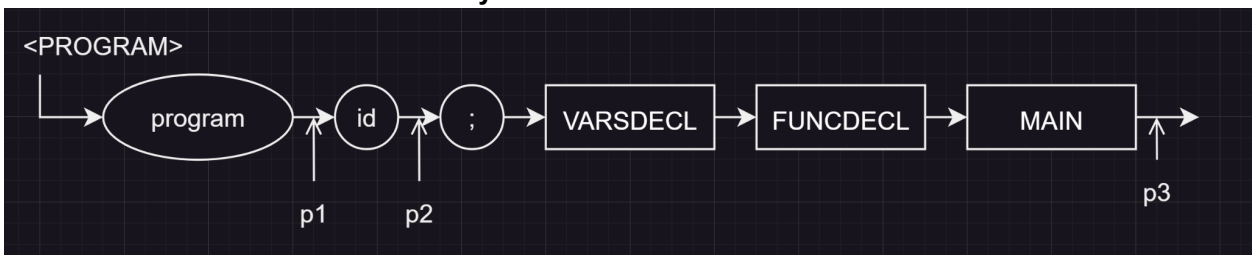
BOOL	ID[‘bool’]
READ	ID[‘read’]
WRITE	ID[‘write’]
IF	ID[‘if’]
ELSE	ID[‘else’]
WHILE	ID[‘while’]
FUNC	ID[‘func’]
TRUE	ID[‘true’]
FALSE	ID[‘false’]
AND	ID[‘and’]
OR	ID[‘or’]
END	ID[‘end’]
RETURN	ID[‘return’]
VOID	ID[‘void’]

Syntactical Analysis

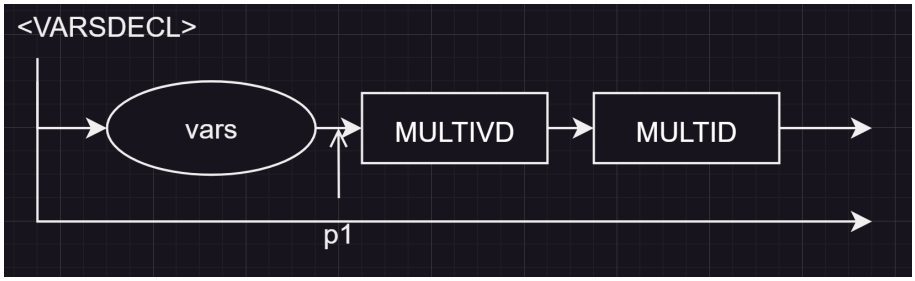
PROGRAM	program id semicolon VARSDECL FUNCDECL MAIN
VARSDECL	vars MULTIVD MULTID | ε
MULTIVD	TIPO colon id ARR MULTID semicln MULTIVD | ε
MULTID	comma id ARR multid | ε
TIPO	int | float | char | bool
ARR	lsquare EXP rsquare | ε
FUNCDECL	Func TIPOFUNC id lparen PARAM rparen lcurly VARSDECL STMNT FUNCDECL | ε
TIPOFUNC	TIPO | void
PARAM	TIPO colon id MULTIPARAM | ε
MULTIPARAM	comma PARAM | ε
MAIN	main lparen rparen STMNT

STMNT	ASS_STMNT STMNT| FUNC_STMNT STMNT | RET_STMNT STMNT | READ_STMNT STMNT | WRITE_STMNT STMNT| IF_STMNT STMNT| WHILE_STMNT STMNT | ε
ASS_STMNT	id ARR ass LOGIC semicln | id ARR ass FUNC_STMNT semicln
FUNC_STMTN	id lparen ARG rparen semicln
ARG	LOGIC MULTIARG | ε
MULTIARG	comma ARG MULTIARG| ε
RET_STMNT	return lparen LOGIC rparen semicln
READ_STMTNT	read lparen LOGIC MULTIO rparen semicln
WRITE_STMNT	write lparen LOGIC MULTIO rparen semicln
MULTIO	comma LOGIC MULTIO | ε
LOGIC	REL | REL and LOGIC | REL or LOGIC
REL	EXP | EXP RELOP REL
RELOP	more_than | less_than | more_or_eq_than | less_or_eq_than | different_to | equal_to
EXP	TERM | TERM sum EXP | TERM sub EXP
TERM	FACTOR | FACTOR mult TERM | FACTOR div TERM
FACTOR	id | id lparen LOGIC MULTIEXP rparen | cte_num | cte_flt | cte_str | true | false
MULTIEXP	comma LOGIC MULTIEXP | ε
IF_STMNT	if lparen LOGIC rparen STMNT ELSE_STMNT end
ELSE_STMNT	else STMNT | ε
WHILE_STMNT	while lparen LOGIC rparen STMNT end

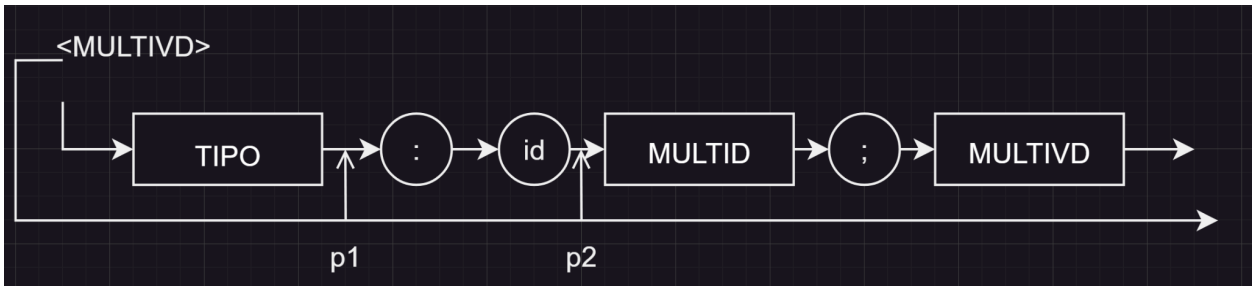
Intermediate Code and Semantic Analysis



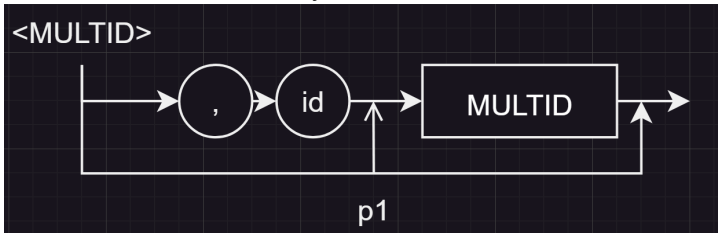
- P1 → Create Function Directory, create first quad GOTO Main
- P2 → Add program's name into the FuncDir and save it to a variable
- P3 → Generate last quad, ENDPROG



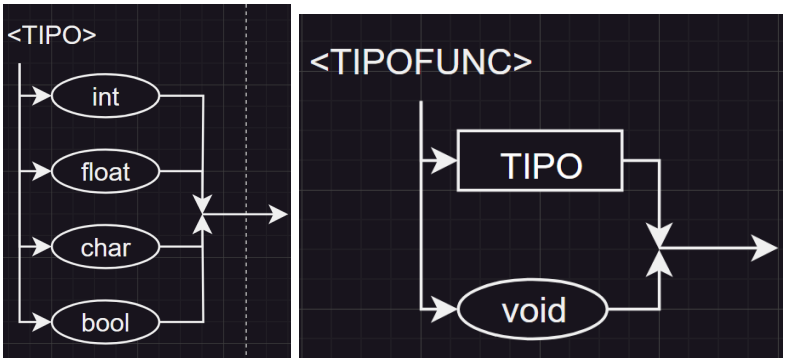
- P1 → Create VarDir for the current scope

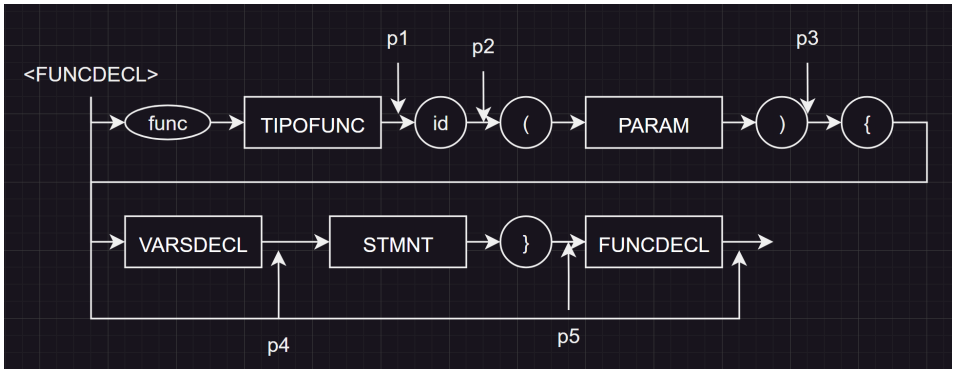


- P1 → Assign type to currType
- P2 → Check if ID already exists, insert ID into VarDir with its type

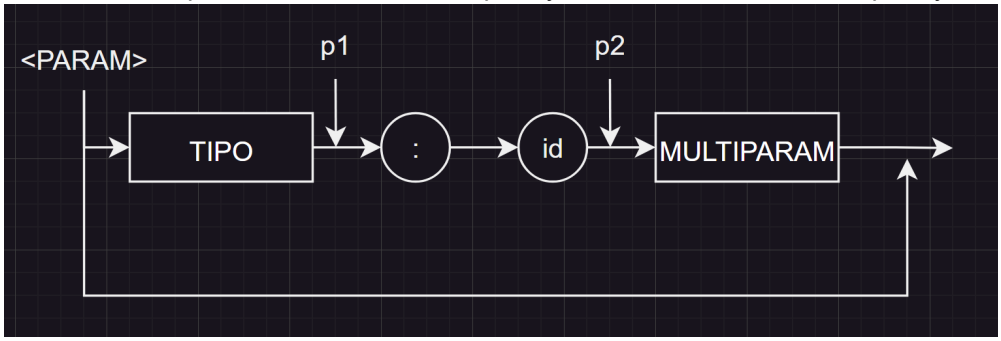


- P1 → Check if ID already exists, insert ID into VarDir with its type

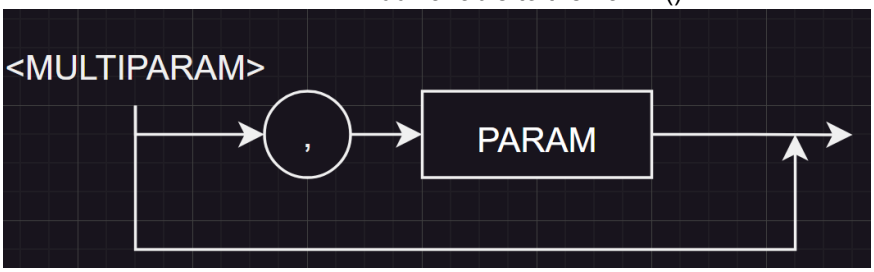


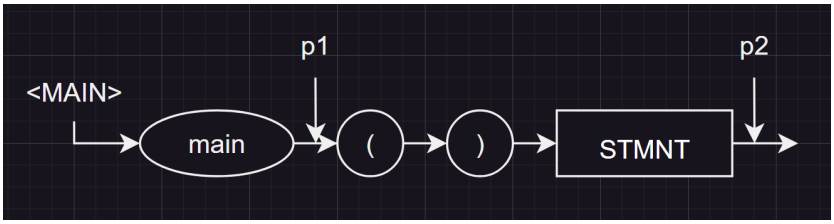


- P1 → Assign type to funcType
- P2 → Check if the ID hasn't been previously declared
- P3 → Set parameter count
- P4 → Set variable count, set quadruple count
- P5 → Reset parameter count, set temporary count, reset local and temporary variables

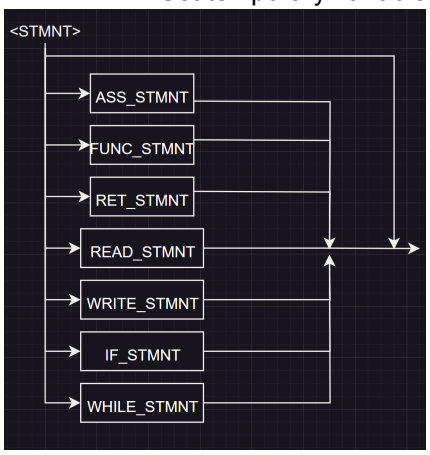


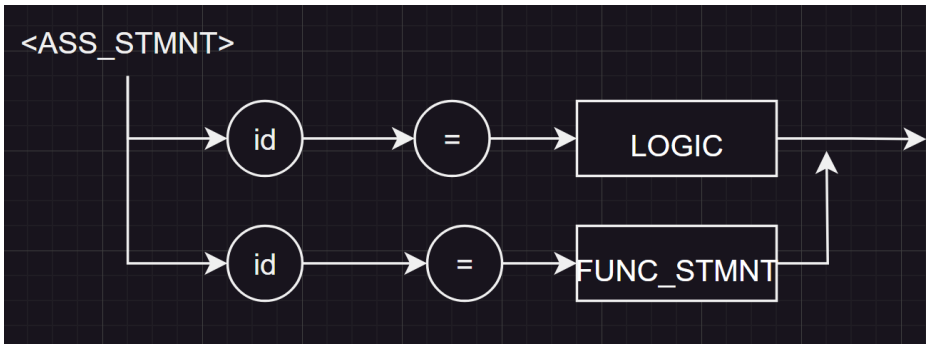
- P1 → Save parameter type to currType
- P2 → Add variable to the VarDir()

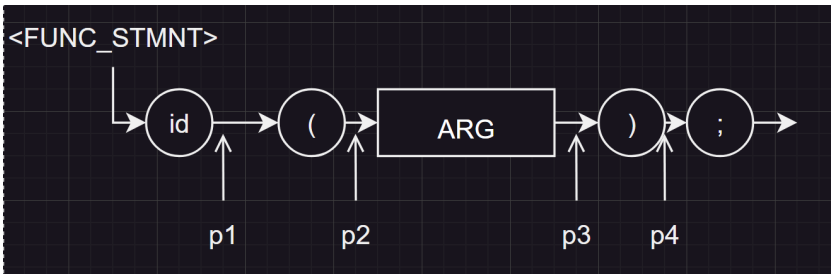


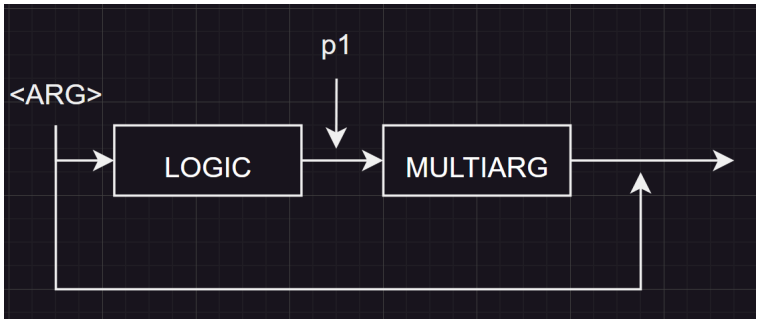


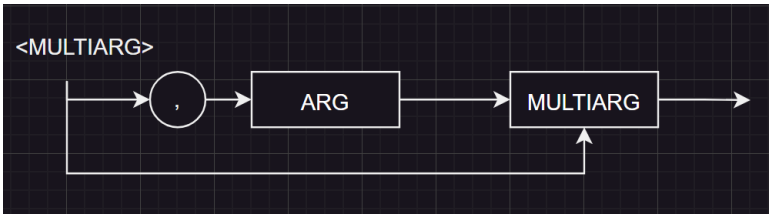
- P1 → Save scopeName as main, generate VarDir for this scope, append quadruple address to the first GOTO
- P2 → Set temporary variables

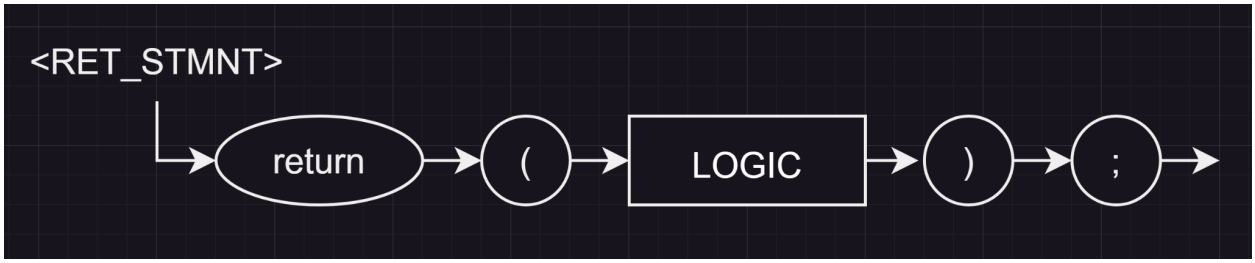


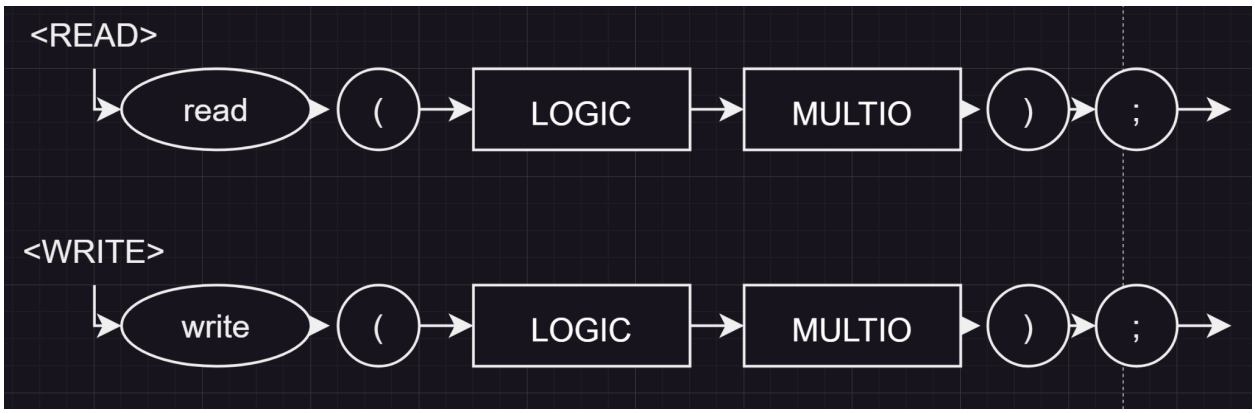


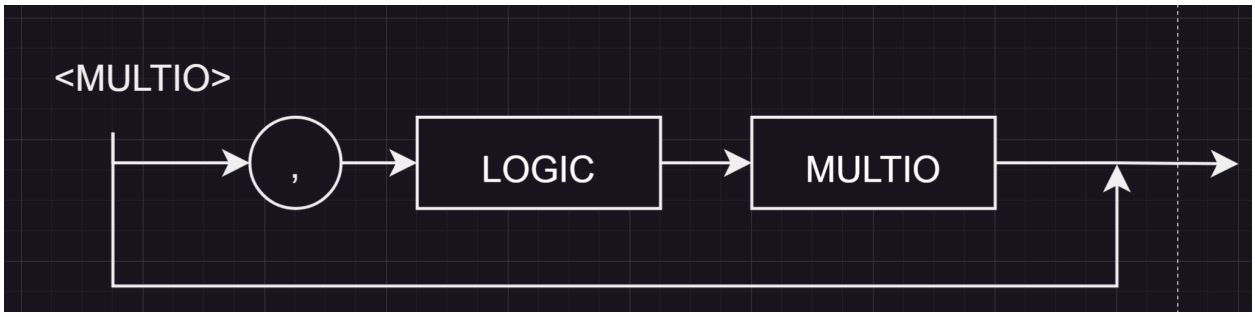


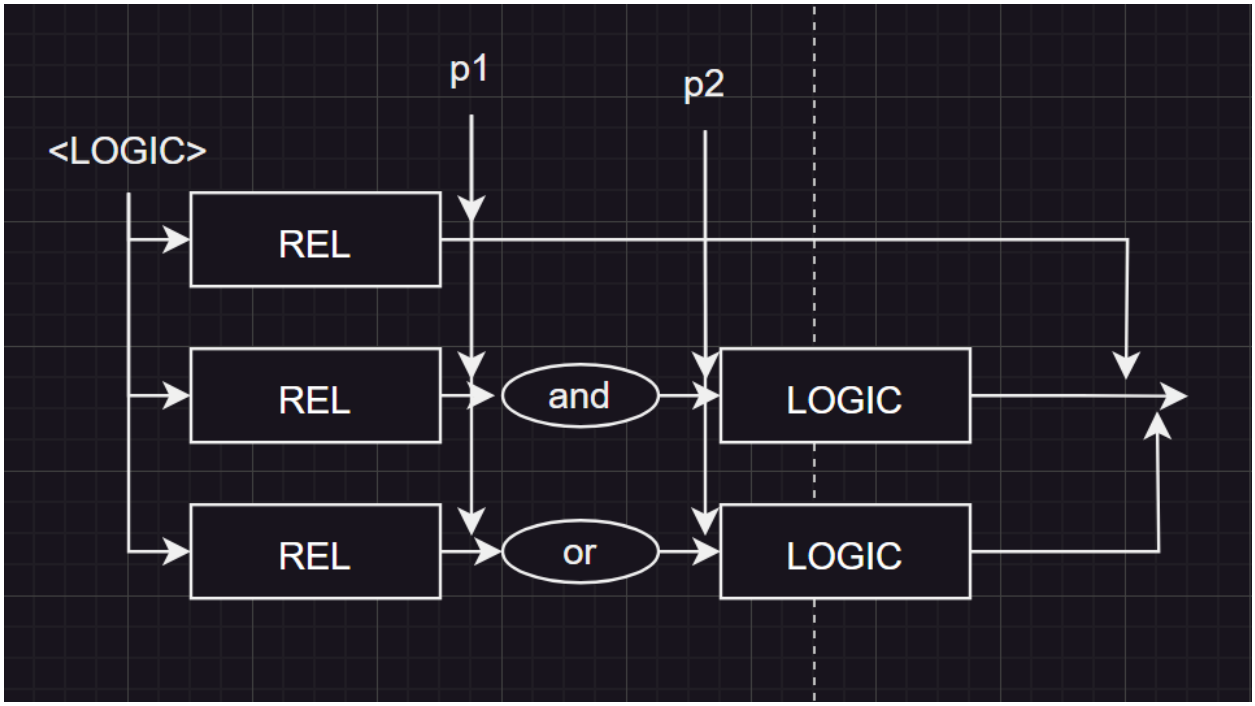


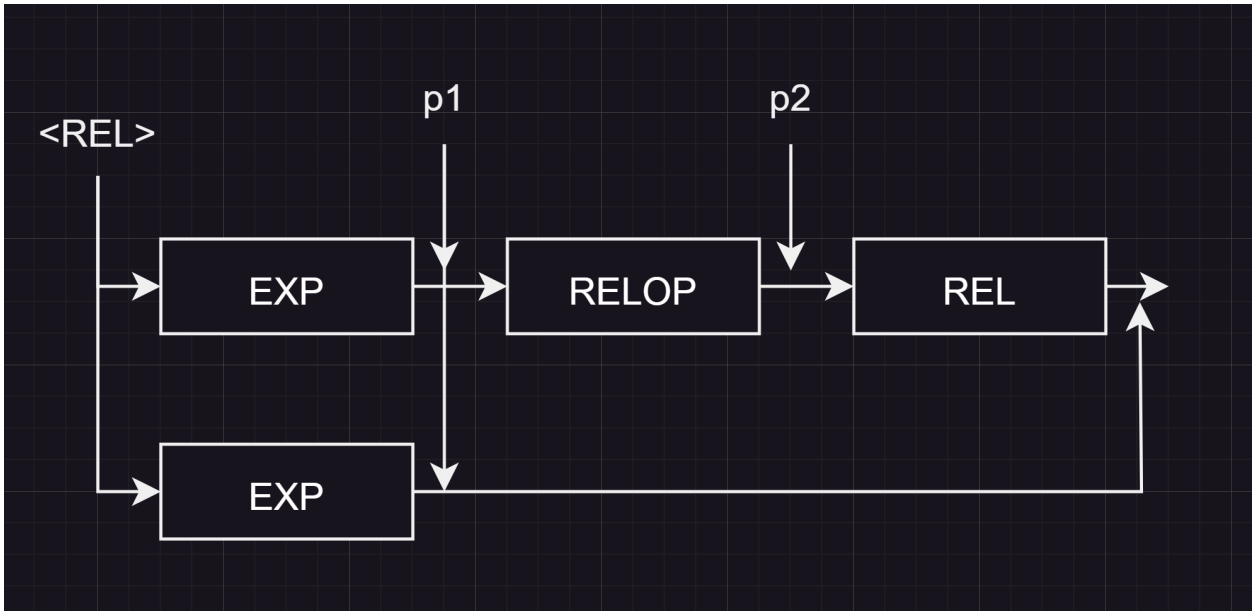

<MULTIARG>
,
ARG
MULTIARG

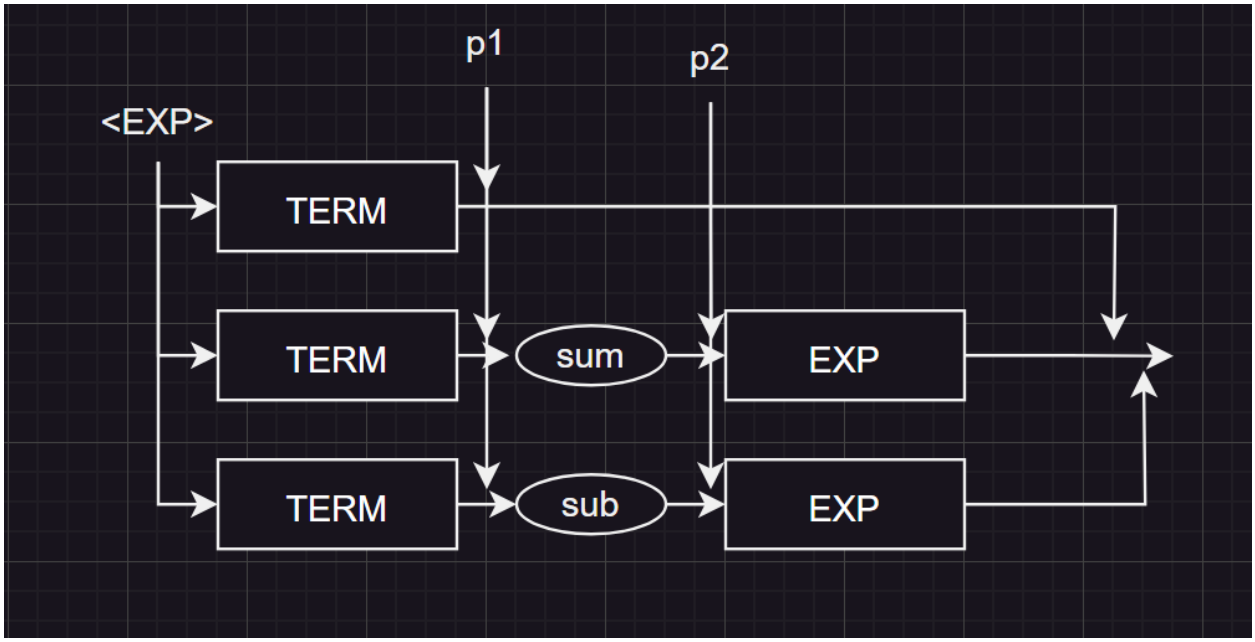

<RET_STMNT>
return
(
LOGIC
)
;

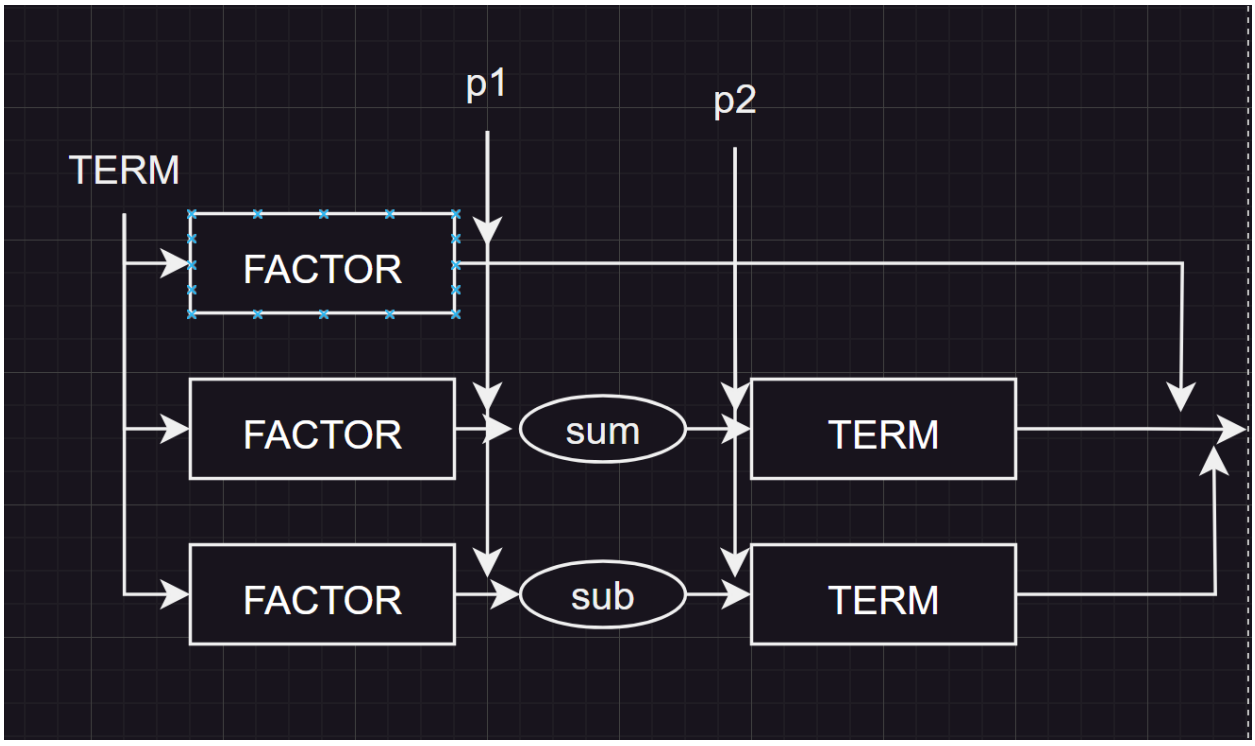

<READ>
read
(
LOGIC
MULTIO
)
;
<WRITE>
write
(
LOGIC
MULTIO
)
;


<MULTIO>
,
LOGIC
MULTIO


<LOGIC>
p1
p2
REL
REL
and
LOGIC
REL
or
LOGIC


<REL>
p1
p2
EXP
RELOP
REL
EXP





Semantic Considerations

L.Op	R.Op	+	-	*	/	>	<	<=	>=	<>	==	AND	OR	=
INT	INT	INT	INT	INT	INT	B	B	B	B	B	B	ERR	ERR	INT
INT	FLT	FLT	FLT	FLT	FLT	B	B	B	B	B	B	ERR	ERR	FLT
INT	CHAR	ERR	ERR	ERR	ERR	ERR	ERR	ERR	ERR	B	B	ERR	ERR	ERR

INT	BOOL	ERR	ERR	ERR	ERR	ERR	ERR	ERR	ERR	B	B	ERR	ERR	ERR
FLT	INT	ERR	ERR	ERR	ERR	B	B	B	B	B	B	ERR	ERR	FLT
FLT	FLT	ERR	ERR	ERR	ERR	B	B	B	B	B	B	ERR	ERR	FLT
FLT	CHAR	ERR	ERR	ERR	ERR	ERR	ERR	ERR	ERR	B	B	ERR	ERR	ERR
FLT	BOOL	ERR	ERR	ERR	ERR	ERR	ERR	ERR	ERR	B	B	ERR	ERR	ERR
CHAR	INT	CHR	ERR	ERR	ERR	ERR	ERR	ERR	ERR	B	B	ERR	ERR	ERR
CHAR	FLT	ERR	ERR	ERR	ERR	ERR	ERR	ERR	ERR	B	B	ERR	ERR	ERR
CHAR	CHAR	CHR	ERR	ERR	ERR	ERR	ERR	ERR	ERR	B	B	ERR	ERR	CHR
CHAR	BOOL	ERR	ERR	ERR	ERR	ERR	ERR	ERR	ERR	B	B	ERR	ERR	ERR
BOOL	INT	ERR	ERR	ERR	ERR	ERR	ERR	ERR	ERR	B	B	ERR	ERR	ERR
BOOL	FLT	ERR	ERR	ERR	ERR	ERR	ERR	ERR	ERR	B	B	ERR	ERR	ERR
BOOL	CHAR	ERR	ERR	ERR	ERR	ERR	ERR	ERR	ERR	B	B	ERR	ERR	ERR
BOOL	BOOL	ERR	ERR	ERR	ERR	ERR	ERR	ERR	ERR	B	B	B	B	B

Memory Management

Function Directory

Function Directory								
Name	Ret. Type	Var. Count	Param. Count	Params	Temp. Var. C	Address	Quad	Var
: string	: string	: array	: int	: array	: int	: int	: int	: pointer

- Name → Function's name
- Return Type → Type of function's return
- Variable Count → Count of total variables (Parameter count + Temporary Variable Count + Variables)
- Parameter Count → Count of parameters
- Parameters → List of parameters
- Temporary Variable Counter → Count of temporary variables generated in the quadruples
- Address → Memory address for the return
- Quad → Quadruple address
- Var → pointer to the variable table

Variable Table

Variable Table		
Name	Address	Type
: string	: int	: string

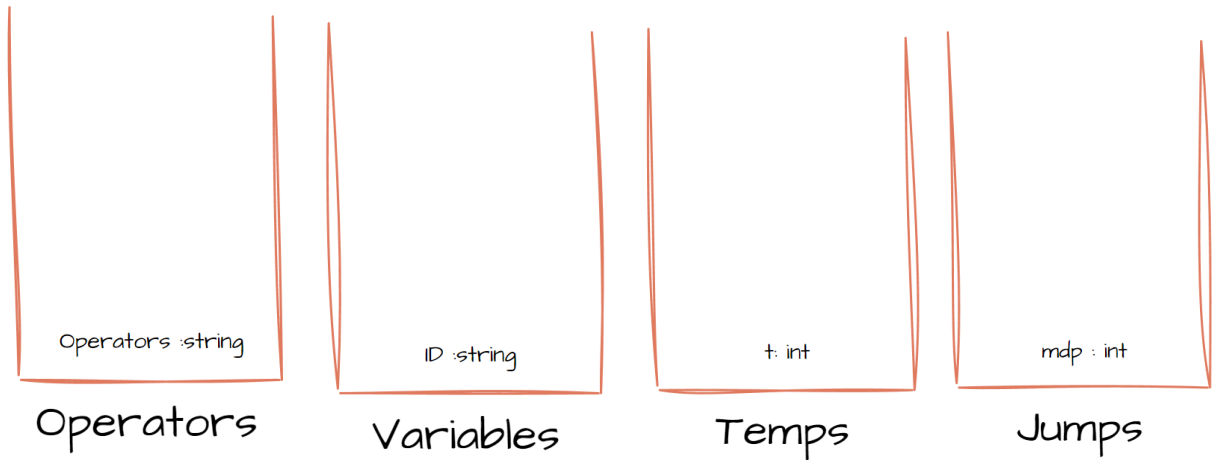
- Name → Variable name
- Address → Memory address for the variable
- Type → Variable type

Constant Table

Constants Table		
Value	Address	Type
: string	: int	: string

- Value → Variable's name
- Address → Variable's type
- Type → Variable's type

Operators **Variables** **Temporary Variables** **Jumps**



Quadruples

Quadruples			
Left Operator	Right Operator	Operand	Result

: int, : string, :bool	: int, : string, :bool	: string	: int

Memory Management

	Local	Global	Temporary	Constant
INT	0-999	4000-4999	8000-8999	12000-12999
FLOAT	1000-1999	5000-5999	9000-9999	13000-13999
CHAR	2000-2999	6000-6999	10000-10999	14000-14999
BOOL	3000-3999	7000-7999	11000-11999	15000-15999

Virtual Machine Description

PRUEBAS DEL FUNCIONAMIENTO DEL LENGUAJE :

DOCUMENTACIÓN DEL CÓDIGO DEL PROYECTO:

User's Guide
