Instituto Tecnológico y de Estudios Superiores de Monterrey
Campus Monterrey

Avance 1

Diseño de Compiladores

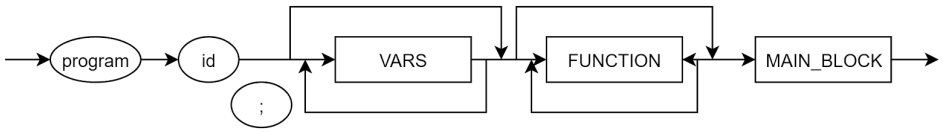
A01154891 Raúl Castellanos

01 de Octubre, 2021

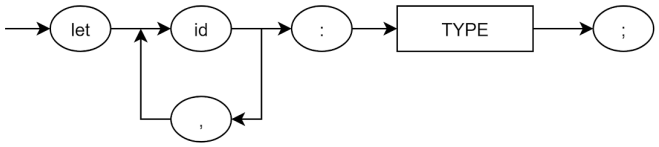
Para este primer avance se diseñó el la sintaxis del lenguaje. Se crearon los diagramas y gramática y se realizó el analizador sintáctico usando PLY, este comprueba la sintaxis de un archivo txt como input y valida que sea correcta o incorrecta.

Diagramas

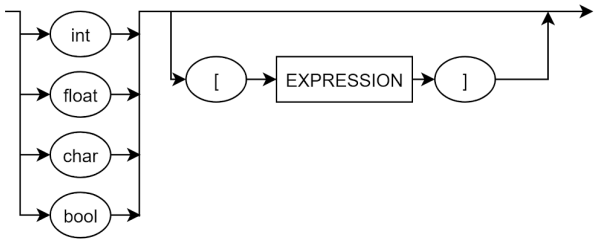
PROGRAM



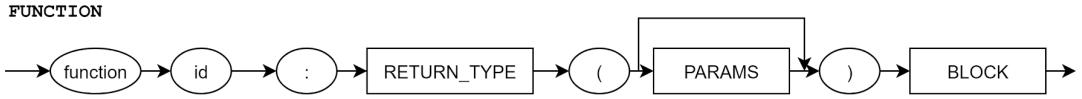
VARS



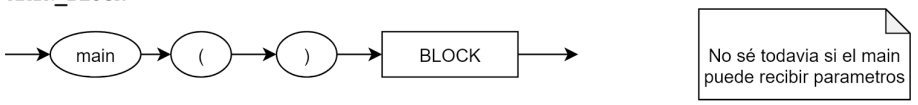
TYPE



FUNCTION

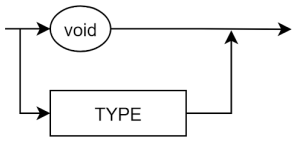


MAIN_BLOCK

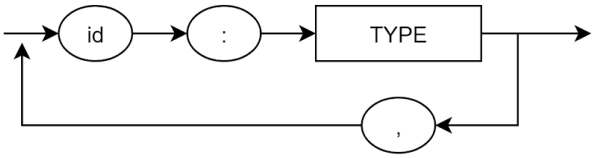


BLOCK

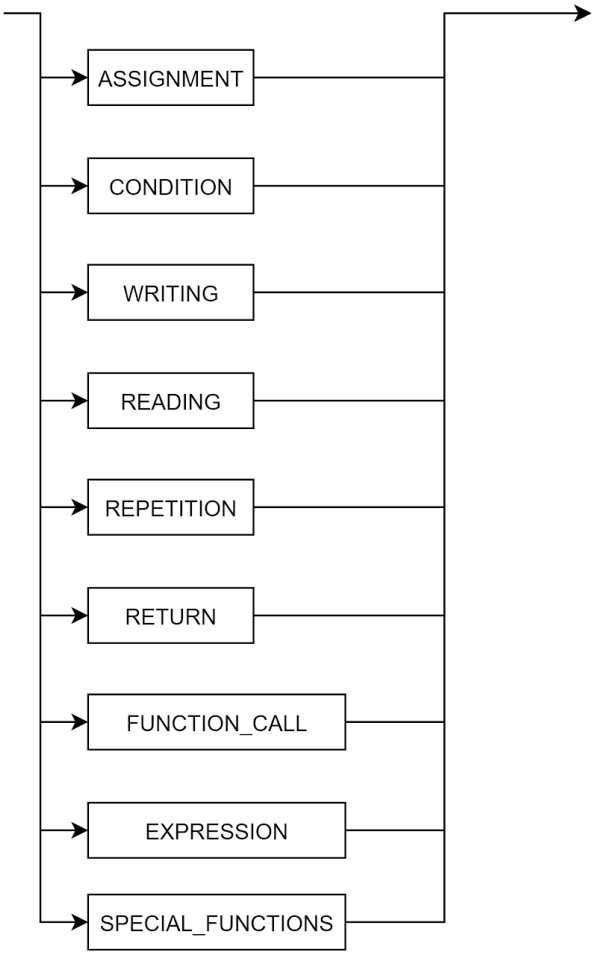
RETURN_TYPE



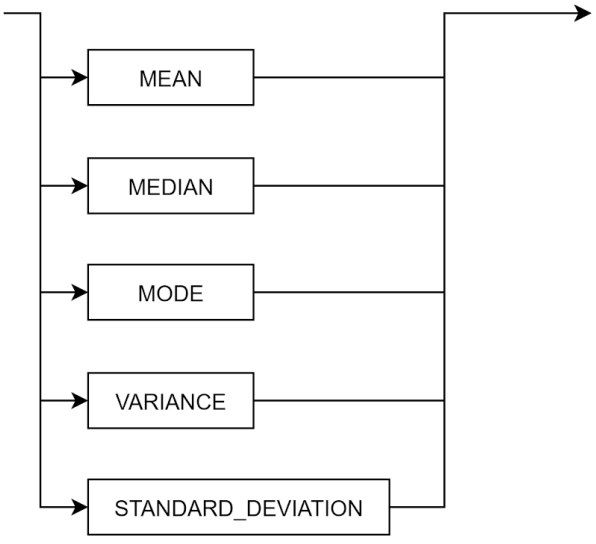
PARAMS



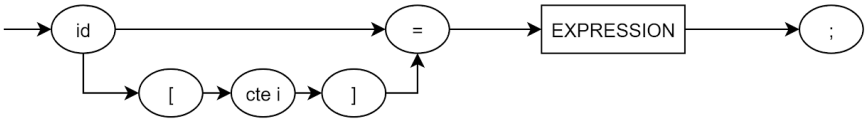
STATEMENTS



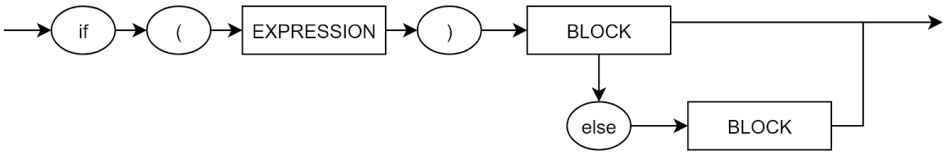
SPECIAL_FUNCTIONS



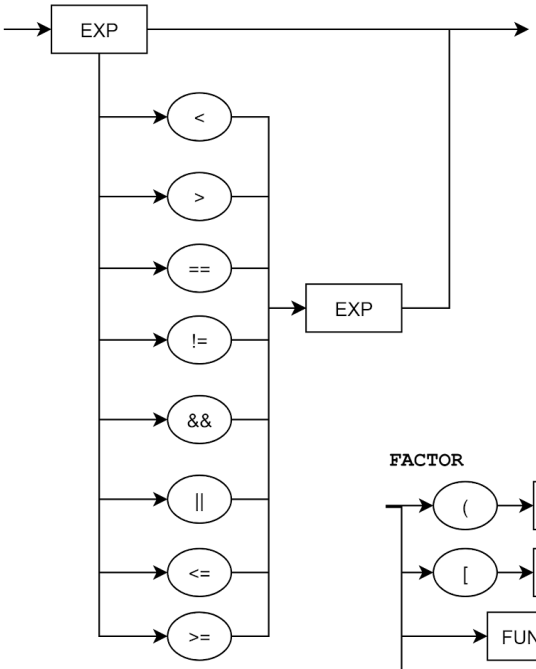
ASSIGNMENT



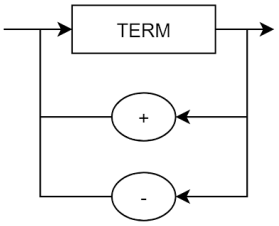
CONDITION



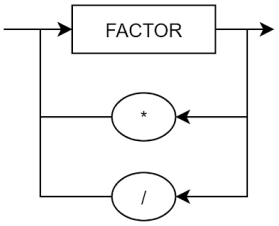
EXPRESSION



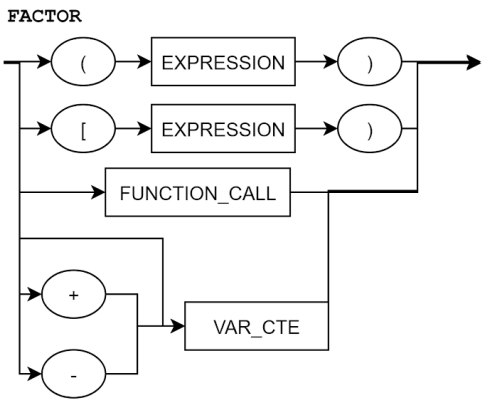
EXP



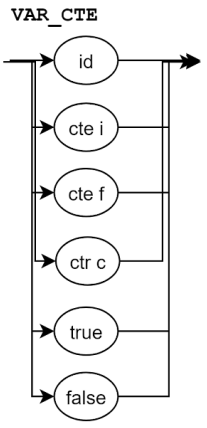
TERM



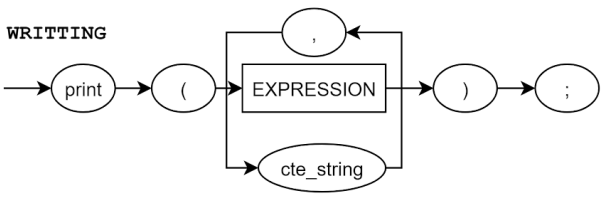
FACTOR



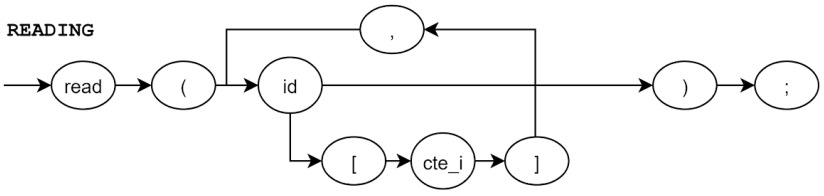
VAR_CTE



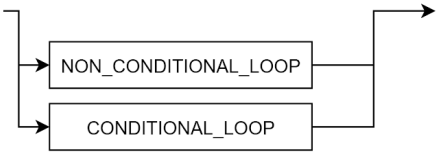
WRITTING



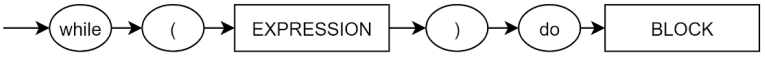
READING



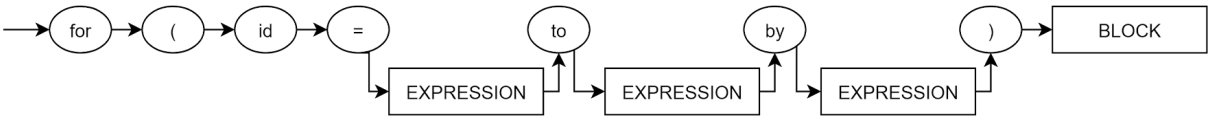
REPETITION



CONDITIONAL_LOOP

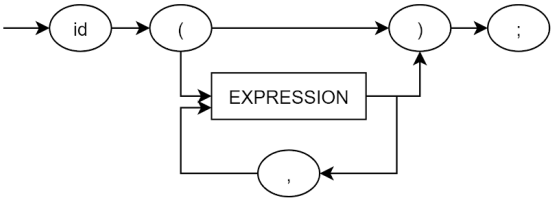


NON_CONDITIONAL_LOOP



RETURN

FUNCTION_CALL



MEAN

Deben recibir un arreglo

MEDIAN

MODE

VARIANCE

STANDARD_DEVIATION

Gramática

S → program id ; PROGRAM1

PROGRAM1 → PROGRAM1
| FUNCTION PROGRAM1
| MAIN_BLOCK

FUNCTION → function id : RETURN_TYPE (PARAMS) BLOCK
| function id : RETURN_TYPE ( ) BLOCK

MAIN_BLOCK → main ( ) BLOCK

RETURN_TYPE → void
| TYPE

PARAMS → id : TYPE , params
| id : TYPE

TYPE → int TYPE1
| float TYPE1
| char TYPE1
| bool TYPE1

TYPE1 → [EXPRESSION]
| *eps*

BLOCK → { STATEMENTS }

EXPRESSION → EXP
| EXP < EXP
| EXP <= EXP
| EXP > EXP
| EXP >= EXP
| EXP == EXP
| EXP != EXP
| EXP && EXP
| EXP || EXP

EXP → TERM | TERM EXP1

EXP1 → + EXP | - EXP

TERM → FACTOR | FACTOR TERM2

TERM2 → * TERM | / TERM

FACTOR → (EXPRESSION)
| [EXPRESSION]
| FUNCTION_CALL
| FACTOR1

FACTOR1 → + VARCTE
| - VARCTE
| VARCTE

VARCTE → id
| ctei
| ctef
| ctec
| true
| false

STATEMENTS → ASSIGNMENT
| CONDITION
| WRITING
| READING
| REPETITION
| RETURN
| FUNCTION_CALL
| EXPRESSION
| SPECIAL_FUNCTIONS

SPECIAL_FUNCTIONS → MEAN
| MEDIAN
| MODE
| VARIANCE
| STANDARD_DEVIATION

ASSIGNMENT → id = EXPRESSION ;
| id [EXPRESSION] = EXPRESSION ;

CONDITION → if (EXPRESSION) BLOCK
| if (EXPRESSION) BLOCK else BLOCK

VARS → let VARS1

VARS1 → id : TYPE ;
| id , VARS1

WRITING → print (WRITING1) ;

WRITING1 → EXPRESSION , WRITING1
| EXPRESSION
| cteString

```
        | cteString ,

READING → read ( READING1 );
READING1 → id , READING1
        | id [ EXPRESSION ] , READING1
        | id
        | id [ EXPRESSION ]

REPETITION → NON_CONDITIONAL_LOOP
        | CONDITIONAL_LOOP

CONDITIONAL_LOOP → while ( EXPRESSION ) do BLOCK

NON_CONDITIONAL_LOOP → for (id = EXPRESSION to EXPRESSION by EXPRESSION)

RETURN → return EXPRESSION ;

FUNCTION_CALL → id ( ) ;
        | id ( FUNCTION_CALL1 );

FUNCTION_CALL1 → EXPRESSION
        | EXPRESSION , FUNCTION_CALL1

MEAN → mean ( EXPRESSION ) ;

MEDIAN → median ( EXPRESSION ) ;

MODE → mode ( EXPRESSION ) ;

VARIANCE → variance ( EXPRESSION ) ;

STANDARD_DEVIATION → stdev ( EXPRESSION ) ;
```

Tokens

ID
CTEF
CTEI
CTEC
CTESTRING
COMMA
COLON
SEMI
LPAREN
RPAREN
LBRACKET

RBRACKET
LBRACE
RBRACE
EQUALS
LT
LE
GT
GE
EQ
NE
OR
AND
PLUS
MINUS
TIMES
DIVIDE
PROGRAM
LET
INT
FLOAT
CHAR
BOOL
TRUE
FALSE
FUNCTION
MAIN
VOID
RETURN
IF
ELSE
WHILE
DO
FOR
TO
BY
PRINT
READ
MEAN
MEDIAN
MODE
VARIANCE
STDEV