COMPILADOR MRL14

Gerardo Guillermo Garza Tamez

A01380899 22/11/21

Índice

Descripción del proyecto

Propósito y Alcance del proyecto

El propósito de esto proyecto fue el realizar un compilador comprendiendo todas las partes que este conforma, desde como se analizan los elementos estáticos que serian los tokens y como se debe estructurar generando un código intermedio para después formar una estructura para que pueda ser analizado por una maquina virtual y poder hacer las diferentes operaciones.

Análisis de requerimientos y descripción de los principios test Cases

Como requerimientos funcionales serían los siguientes:

- El compilador debía poder reconocer diferentes tokens tales como if, else, program, int , float, print, return y otros mas especiales como circulo, punto, línea, entre otros
- El programa debe ser capaz de hacer las operaciones básicas como sumar, restar, multiplicar y dividir
- Debe guardar las funciones en algún arreglo
- Debe poder guardar las variables
- Debe generar cuadruplos

Como requerimientos no funcionales serian:

- Se debe guardar en la memoria de manera efectiva

Descripción del proceso

Para el desarrollo de este proceso se inició generando las gramáticas e identificando cuales serian los caracteres clave, así como las palabras reservadas para generar los tokens.

En las siguientes semanas se comenzó a trabajar ya en el código se creó el léxico en Python todo el proyecto se desarrolló en Python y se creo la base del parser e interprete.

Mientras iba pasando las semanas se iban incluyendo las clases extra que se fueran a necesitar como la tabla de variables, cubo semántico etc.

Se encontraron algunas cosas que estaban incorrectas debido a que la investigación antes realizada no era exactamente lo que se necesitaba y en la adaptación ocasiono fallos en la lógica.

Como reflexión seria darle su tiempo y respeto a lo que se necesita si ocupas ayuda con algo no te quedes callado muévete y búscala con compañeros o profesores.

Descripción del lenguaje

Nombre del lenguaje: Andrea

Este lenguaje genera los cuádruplos básicos de suma resta multiplicación etc. A su vez puede realizar operaciones como condicionales, también tiene dos tipos de iteraciones el for loop y el while loop, el lenguaje que interpreta es muy similar a C con pocas diferencias como que solo se puede declarar variables globales al principio después de indicar el nombre del programa o declarar las variables locales al principio de cada función ya sea el main o cualquier otra función, permitirá arreglos pero de máximo 2 dimenciones, no hay variables tipo string ni double, long o bool pero si es capaz de interpretar FALSE, TRUE y condicionales.

Descripción del compilador

Se género en Windows con un procesador dual core y 16Gb de ram, el lenguaje en el que se desarrollo el compilador fue Python y de utilerías se usaron las librerías de ply para el léxico y parser, la librería de os para poder navegar en el sistema, la librería de códecs para abrir un archivo y la librería de re para generar las expresiones regulares

Descripción del análisis de léxico

se guardaron las siguientes palabras como tokens:

```
reservadas = [
  'PROGRAM', 'program',
    'ID', 'id',
    'INT', 'int',
    'FLOAT', 'float',
    'CHAR', 'char',
    'IF', 'if',
    'ELSE', 'else',
    'READ', 'read',
    'VOID', 'void',
    'RETURN', 'return',
    'FOR', 'for',
    'WHILE', 'while'
]

tokens = reservadas + [
```

```
        # ; { } , = ( ) [ ]
        'SEMICOLON','L_BRACE', 'R_BRACE',
        'COMMA', 'ASSIGN', 'L_PARENTHESIS','R_PARENTHESIS',
        'L_BRACKET','R_BRACKET',

        #operators + - * /
        'PLUS',  'MINUS', 'MULT', 'DIV',

        #bool > < != == && ||
        'GREATER_THAN', 'LESS_THAN', 'DIFF_THAN',
        'EQUALS_TO', 'AND', 'OR',

        #Constants
        'CONST_ID', 'CONST_INT', 'CONST_FLOAT', 'CONST_CHAR','LETRERO'
]
```

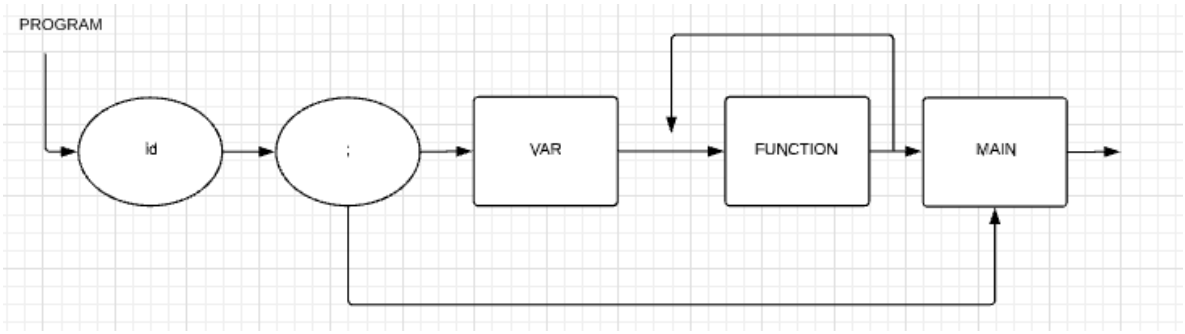
Para estos se generaron las siguientes expresiones regulares:

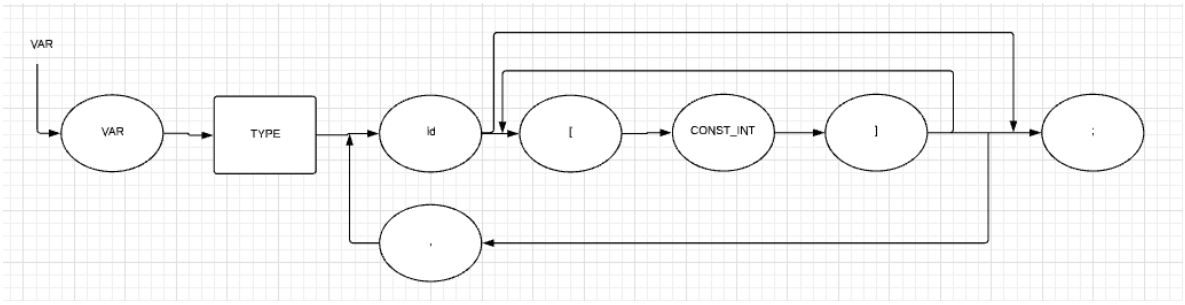
```
t_ignore = '\t\n'
t_ignore_space = '\s'
t_SEMICOLON = r';'
t_L_BRACE = r'\['
t_R_BRACE = r'\]'
t_COMMA = r','
t_ASSIGN = r'='
t_L_PARENTHESIS = r'\('
t_R_PARENTHESIS = r'\)'
t_L_BRACKET = r'\{'
t_R_BRACKET = r'\}'
```

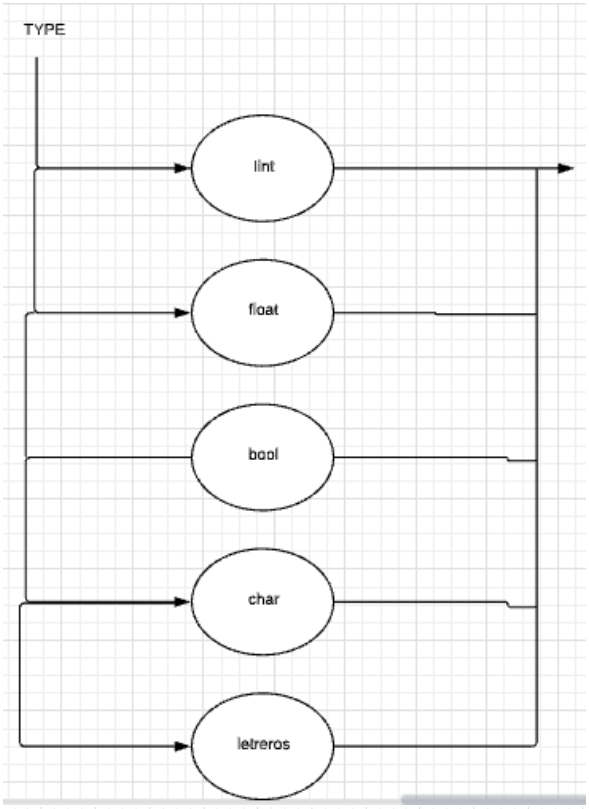
```
t_PLUS = r'\+'
t_MINUS = r'\-'
t_MULT = r'\*'
t_DIV = r'/'

t_GRATHER_THAN = r'\>'
t_LESS_THAN = r'\<'
t_DIFF_THAN = r'\!='
t_EQUALS_TO = r'=='
t_AND = r'&&'
t_OR = r'\|\|'
t_CONST_ID = r'[a-zA-Z_][a-zA-Z_0-9]*'
t_CONST_INT =  r'\d+'
t_CONST_FLOAT= r'\d\.\d+'
t_CONST_CHAR = r'[a-zA-Z_0-9]'
t_LETREROS=  r'[a-zA-Z_][a-zA-Z_0-9]*'
```

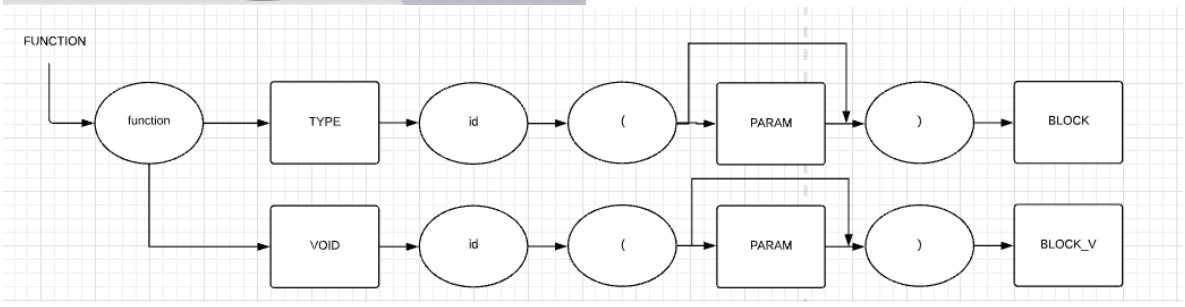
Descripción del análisis de sintaxis

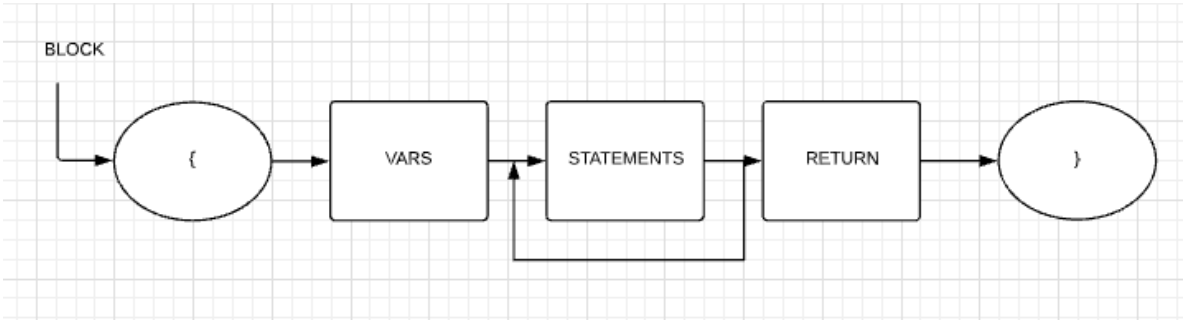
Para la generación de la sintaxis se tuvo que evaluar la estructura de como es que serian interpretado en el lenguaje y se generaron los siguientes diagramas

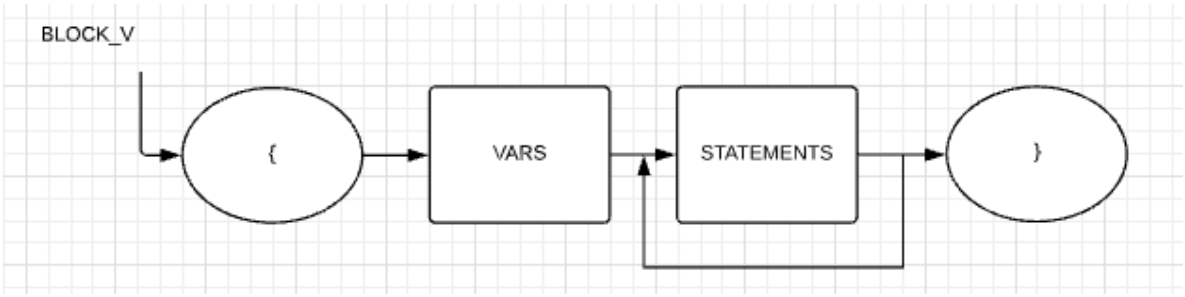


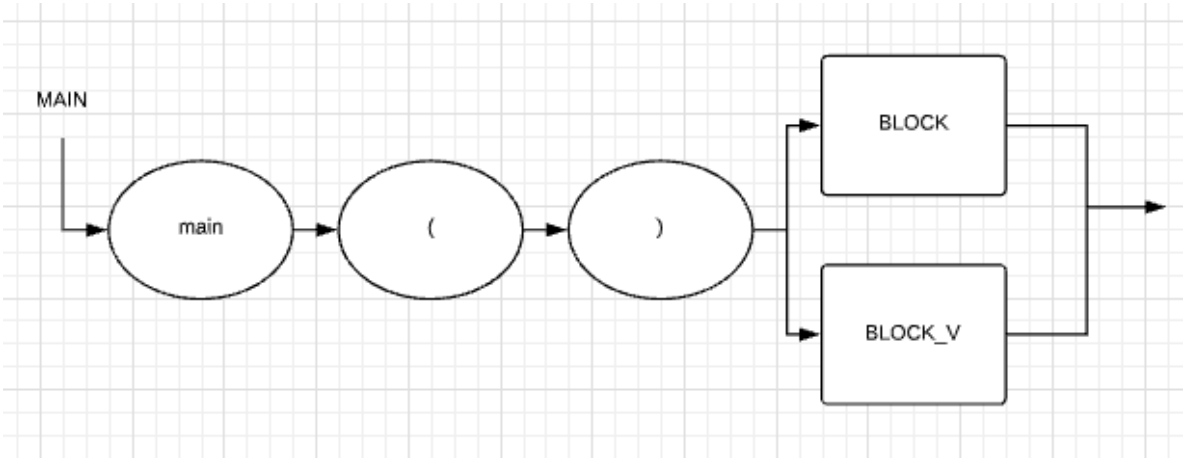

VAR
VAR
TYPE
id
[
CONST_INT
]
;
,

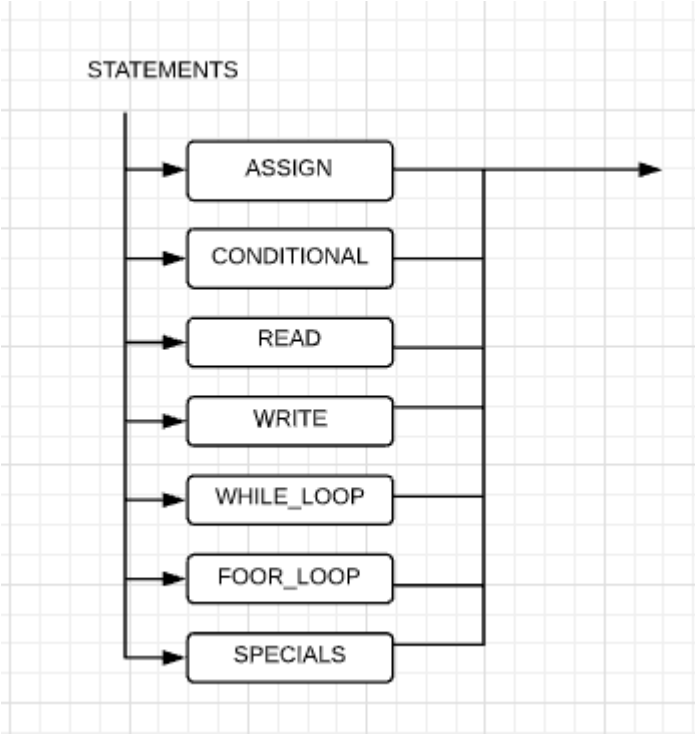

TYPE
int
float
bool
char
letreros

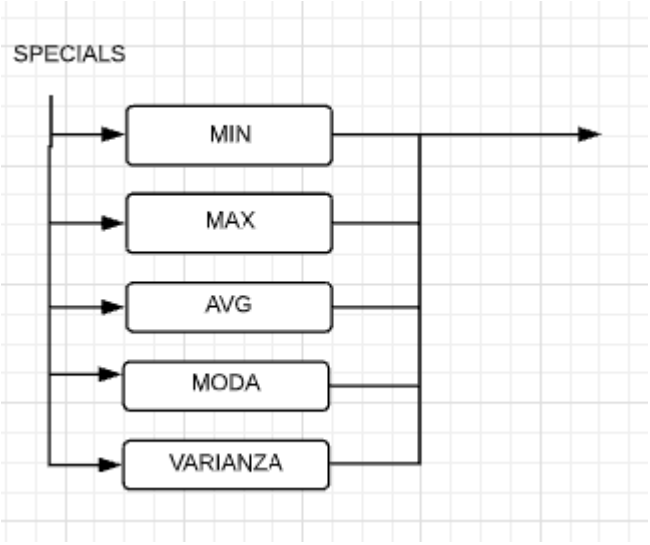

FUNCTION
function
TYPE
id
(
PARAM
)
BLOCK
VOID
id
(
PARAM
)
BLOCK_V

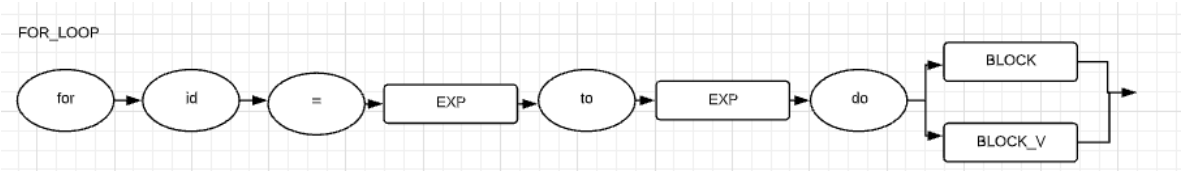

BLOCK
{
VARS
STATEMENTS
RETURN
}

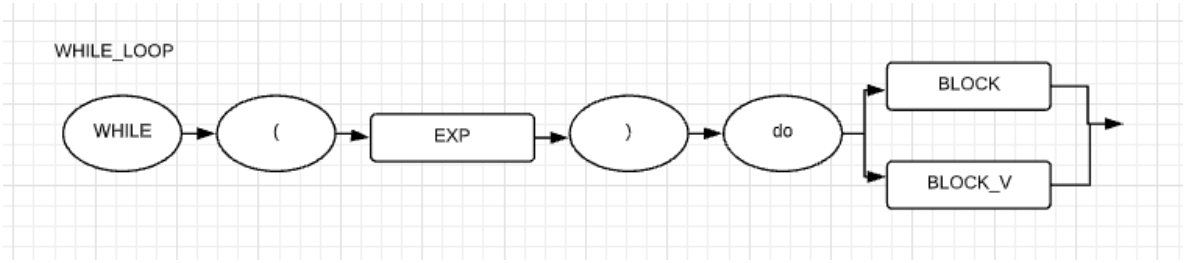

BLOCK_V
{
VARS
STATEMENTS
}

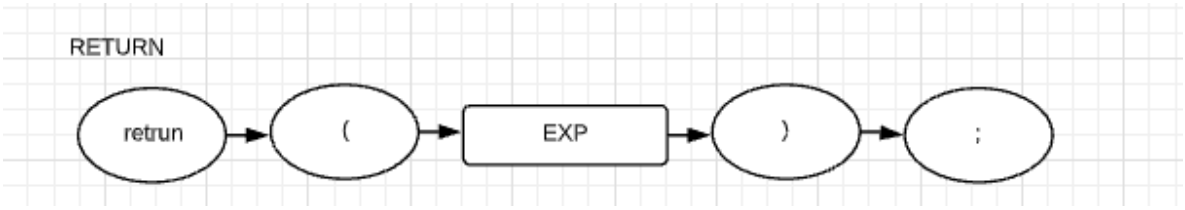

MAIN
main
(
)
BLOCK
BLOCK_V

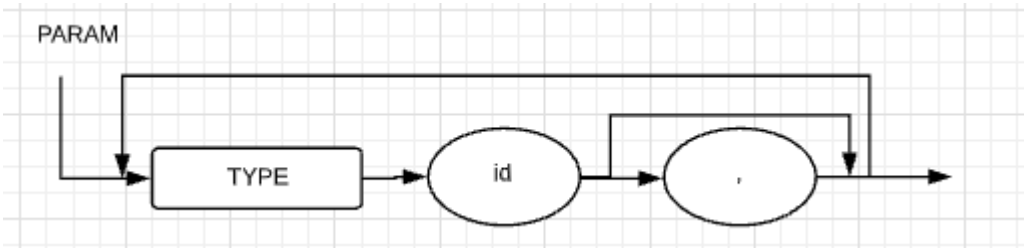

STATEMENTS
ASSIGN
CONDITIONAL
READ
WRITE
WHILE_LOOP
FOOR_LOOP
SPECIALS


SPECIALS
MIN
MAX
AVG
MODA
VARIANZA


FOR_LOOP
for
id
=
EXP
to
EXP
do
BLOCK
BLOCK_V







Descripción de generación de código intermedio y análisis Semántico

Para la generación de código intermedio se generaron cuadruplos y se asignaron ciertas direcciones para la memoria siendo los caracteres y tokens reservados son números fijos que van desde el [- al -] para palabras reservadas como int, float char, read, for, if etc. Y para los caracteres especiales como “,”, “;”, “=”,”*”, “+” etc se les asigno un numero del [- al -] siendo estos caracteres que nunca podrán cambiar de posición estos fueron los números que les asigne, separe la memoria en 10 secciones la primera seria para los enteros que designe los números 500 en adelante para los enteros los enteros temporales serían los 600 en adelante los números flotantes del 750 en adelante cabe recalcar que para los números enteros flotantes separe mas ya que lo use como los temporales los números flotantes temporales son a partir del 850, los caracteres en el 900 los caracteres temporales en el 1000 y los strings o letreros en 1100 y los temporales de estos en el 1200.

Use la lista que seria lo mismo que usar una pila ya que era algo sencillo y al tenerlos todos separados por cierta cantidad los vuelve fácil de buscar o filtrar.

Descripción de la Máquina virtual

La maquina virtual es el programa que se encargara de analizar los cuadruplos generados por el parser así como tener acceso a la tabla de funciones y la tabla de variables para obtener los datos de cada tabla para poder realizar las operaciones usa la misma “clase” memoria como simulación de la memoria requerida del interprete a la computadora.

Pruebas del funcionamiento del lenguaje

Documentación del código del proyecto

```
def t_vars(t):
    global numInt, numFloat, numTxt
    if t[1] == 'INT':
        tabla.addVar(t[1], t[2], numInt)
        memo.enteros.append(numInt)
        numInt += 1
    elif t[1] == 'FLOAT':
        tabla.addVar(t[1], t[2], numFloat)
        memo.flotantes.append(numFloat)
        numFloat += 1
    else:
        tabla.addVar(t[1], t[2], numTxt)
        memo.text.append(numTxt)
        numTxt += 1


def t_varst(t):
    global numIntT, numFloatT, numtxtT
    if t[1] == 'INT':
        tabla.addVar(t[1], t[2], numIntT)
        memo.ent_temp.append(numIntT)
        numIntT += 1
    elif t[1] == 'FLOAT':
        tabla.addVar(t[1], t[2], numFloatT)
        memo.flot_temp.append(numFloatT)
        numFloatT += 1
    else:
        tabla.addVar(t[1], t[2], numtxtT)
        memo.text_temp.append(numtxtT)
        numtxtT += 1
```